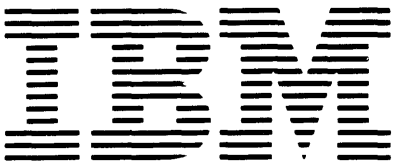
IBM

8100 Information System

Principles of Operation



8100 Information System

Principles of Operation

GA23-0031-4
File No. 8100-00

Fifth Edition (March 1984)

This is a revision of, and obsoletes, GA23-0031-3. The changes include both technical and editorial clarification to improve accuracy and usability of the publication, and are indicated by a vertical line in the left margin. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 Bibliography*, GC20-001, for the editions that are applicable and current.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

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, Department 52Q, Neighborhood Road, Kingston, New York 12401. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

The IBM 8100 Information System provides a coordinated set of information processing equipment. The logical structure of the 8100 system, including the processing and control element (PCE), is unique and permits several levels of performance with the preservation of program compatibility.

This publication provides, for reference purposes, a detailed definition of the machine functions performed by the PCE within an IBM 8100 Information System processor. For purposes of this publication, the PCE is defined as the logical entity that is the controlling center of the system. Additional definitions are included in Chapter 1 and in the Glossary.

The manual describes each function to the level of detail that must be understood to prepare a program that relies on that function. It does not, however, describe all the notation and conventions employed in preparing such a program; for this information, the user must instead refer to the appropriate assembler language manual, such as *IBM 8100 DPPX Assembler Programming: Language Reference and Guide,* SC27-0412.

The information in this publication is provided principally for use by assembler language programmers, although anyone concerned with the functional details of the IBM 8100 Information System processors will find it useful.

This manual is written as a reference document and should not be considered an introduction or a textbook for the IBM 8100 Information System. It assumes the user has a basic knowledge of data processing systems and, specifically, the IBM 8100. Such basic knowledge can be derived by selecting the appropriate hardware and/or software publications from the IBM 8100's system library.

All facilities discussed in this publication are not necessarily available on every processor model in the IBM 8100 system. Furthermore, in some instances the definitions are structured to allow certain extension capabilities to be described even though they are not offered on any currently available processor model. An example is the provision for the number of bits in an adjunct register. The allowance for this type of extension should not be construed as implying any intention by IBM to provide such capabilities. Appendix G provides a summary of processor-specific parameters and functions identified as such within this manual. For information about the characteristics and availability of features on a specific processor model, refer to the appropriate publications for that model. The availability of features on processor models is summarized in *An Introduction to the IBM 8100 Information System,* GA27-2875.

Although processor models of the IBM 8100 system may differ in implementation and physical capabilities, logically they are compatible. That is, within the limitations described below, any program gives identical results on any model.

The compatibility rule has four limitations:

1. The system's facilities used by the program should be the same in each case. For example, the optional processor features and the storage capacity, as well as the quantity and type of input/output equipment, should be equivalent.

2. The program should be independent of the relation between instruction execution times, input/output data rates, storage access times, and elapsed time values.

3. The program should not depend on functions identified in this manual as processor-model dependent, on operations explicitly not defined, on results defined to be unpredictable, or on special-purpose functions that are not described in this manual.

4. The program should not use or depend on reserved fields unless they are explicitly made available for program use. Additionally, the program should not be designed to cause interruptions by means of format errors, such as the use of undefined operation codes.

The information presented in this manual is grouped into 10 chapters and several appendixes. The 10 chapters are organized in three parts: Part I, the first chapter, is an overview description of the processing facilities provided by the IBM 8100 system. Part II consists of the next four chapters and pertains to functions that are useful to all assembler language programmers. Part III, the last five chapters, contains information normally used by programmers who develop or maintain supervisory-type programs.

Part I. Overview

Chapter 1, Introduction to the Logical Structure, describes the processing facilities available with the IBM 8100 Information System. It summarizes the information provided in the remaining nine chapters.

Part II. Information Processing Facilities

Chapter 2, Storage and Registers, describes how the program refers to instruction-operand information in main storage or a general register.

Chapter 3, Program Execution, explains the role of instructions and their formats in program execution. It provides a detailed description of sequence of the instructions and storage references. Program exceptions and the action taken by the PCE as a result of these exceptions are also described.

Chapter 4, General Instructions, describes in detail the instructions available to all programs for general use.

Chapter 5, Floating-Point Instructions, describes in detail the instructions provided with the floating-point feature. Program exceptions relating to floating-point operations are also described in this chapter.

Part III. Supervisory Facilities

Chapter 6, Register Organization, explains the organization and use of the groups of register sets available to the program, and contains the descriptions of the instructions for retrieving or storing information in any register set.

Chapter 7, Dynamic Address Relocation and Translation, explains the operation of the machine facilities, including dynamic address relocation and dynamic address translation, that ease the task of controlling the allocation and use of main storage. The instructions used to manipulate the translation table are also described.

Chapter 8, Input/Output Operations, explains the programmed control of I/O devices by the channel and by the PCE. It includes detailed descriptions of the I/O instructions and other I/O control formats.

Chapter 9, PCE Control, describes in depth the facilities for the switching of system status, for protecting the system against the unauthorized modification of system status, for defining a relative processing priority among programs, and for temporarily disallowing the processing of one or more programs. It deals specifically with PCE states, program modes, the program status vector, the floating-point status vector, priority levels, control vectors, interruptions, and the dispatching mechanism. It includes the detailed descriptions of the instructions available for reading or altering PCE status information, and for directly controlling certain attached system facilities.

Chapter 10, Dual-Mode Processing, describes the differences between processors containing one PCE and processors containing two PCEs. It also describes dual-mode processing and how it is implemented in those models that contain two PCEs.

The Appendixes include:

- Lists of the instructions arranged in several sequences (App A)
- Explanation of assembler language notation used in this manual (App B)
- Summaries of instruction operations and result conditions (App C)
- List of instruction formats arranged by operation code (App D)
- Summaries of important formats (App E)
- Summary of permanently assigned register sets (App F)
- Summary of processor-specific functions (App G)
- Table of the powers of 2 (App H)
- Tabular information helpful in dealing with hexadecimal numbers (App I)
- An EBCDIC chart (App J)
- Information about number representation (App K)

A glossary of important terms used in this publication is provided at the back of the manual, preceding the Index.

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

Prerequisite Publications

An Introduction to the IBM 8100 Information System, GA27-2875

Corequisite Publications

IBM 8130 Processor Description, GA27-3196

IBM 8140 Processor Description, GA27-2880

IBM 8150 Processor Description, GA23-0122

Related Publications

IBM 8100 DPPX Assembler Programming: Language Reference and Guide, SC27-0412

IBM 8130 and 8140 Processors Operator's Guide, GA27-3197

IBM 8150 Processor Operator's Guide, GA23-0123

Contents

Figures

Summary of Changes

Fifth Edition (March 1984)

This edition includes information that relates to:

- Separation protection, which uses translation locks and protection keys.
- Exception block index (EBI) registers.
- The LOAD FROM TRANSLATION LOCK TABLE (LATL) and STORE TO TRANSLATION LOCK TABLE (STATL) instructions.
- The control immediate WRITE DIAGNOSTIC CONTROL VECTOR (KI 192) and READ DIAGNOSTIC CONTROL VECTOR (KI 193) instructions.
- The 8130 Model B and the 8150, where applicable.
- Corrections and clarifications of the previous revision level.

PART I. OVERVIEW

Chapter 1. Introduction to the Logical Structure

Chapter 1. Introduction to the Logical Structure

Note: *Before using this manual, review the Preface that precedes the table of contents. The Preface (1) describes the purpose and content of this manual, (2) defines some assumptions made, (3) identifies the intended user, and (4) indicates the prerequisite knowledge needed by the user.*

Also, be aware that not all 8100 processor models implement every architectural function. For example, the floating-point feature, exception block index (EBI) registers, separation protection, two channels in dual-PCE processors, and certain instructions are not available on all models. Refer to "Appendix G" for processor-specific functions.

This chapter serves as an introduction to the major information formats and processing facilities provided by the IBM 8100 Information System. The description is mainly tutorial; it is not intended as a rigorous specification. The remaining chapters and appendixes provide such a specification. Chapter 1 should be read before using this publication as a reference.

Available on all processors are control facilities that provide system functions, such as I/O interrupt request identification, programmable assignment of I/O devices to priority levels, execution of direct-control (KDO) instructions, initial program load (IPL), and system and I/O reset. These facilities for system control as they apply to 8100 operation are described in the respective processor description manuals listed in the Preface under "Corequisite Publications".

Logical Structure

The logical structure of an IBM 8100 Information System consists of main storage, a processing and control element (PCE), a channel, and input/output devices attached to the channel through adapters. In IBM 8100 Information Systems having the dual-mode capability, the logical structure includes an interrupt control element (ICE) and a second PCE. Figure 1-1 shows this logical structure.

The PCE is the logical entity that is the controlling center of the system. It contains the sequencing and processing controls for instruction execution, interruption action, dynamic address transformations, and other control or processing functions. The physical makeup of the PCE in the processor models of the 8100 system may be different, but the logical function remains the same.

Some processor models contain two PCEs (primary and secondary) and also an interrupt control element (ICE) that enables communication between the PCEs. These models are referred to as dual-PCE processors, which can operate either in dual or single mode. Dual mode is the normal operational mode and uses both PCEs; single mode uses only the primary PCE.

Each PCE includes 32-bit registers used as general-purpose registers. Also included are registers that are permanently assigned to hold control information. Floating-point registers having 64 bits each are optionally available.

Three distinct types of processing are provided by the PCE: (1) logical manipulation of bits, fixed-length information units, and character strings; (2) fixed-point binary arithmetic; and (3) (optionally) floating-point arithmetic.

The 8100 system is designed for use with a supervisory program that coordinates the use of the system's resources. The PCE includes facilities for protection, dynamic address transformations, interruption handling, and PCE control.

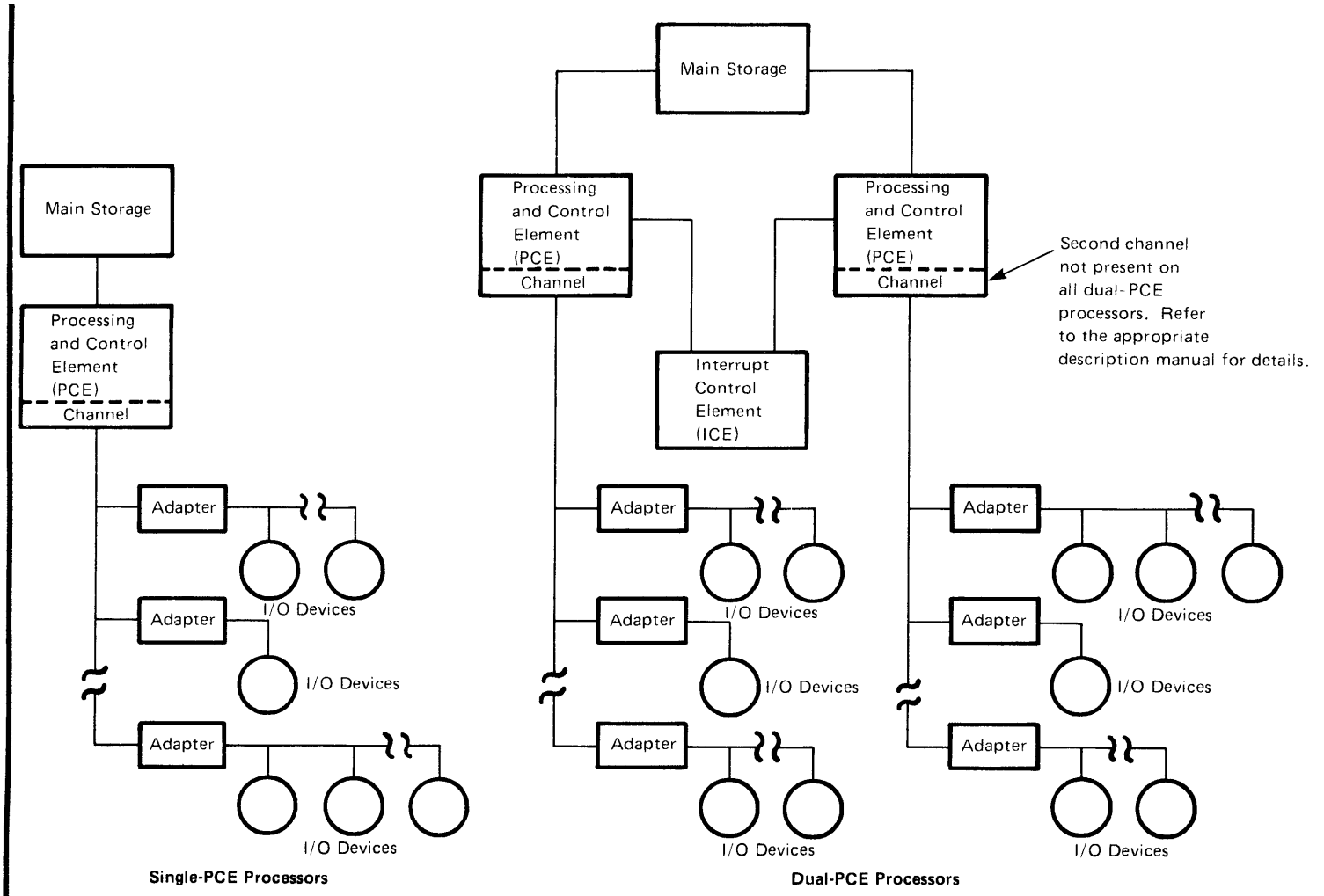


Figure 1-1. IBM 8100 Information System Logical Structure

Information Formats

Within any information format in the 8100 system, the bits making up the format are numbered consecutively left to right, starting with 0. The basic building block of all formats is an 8-bit unit of information. Fixed-length fields of one, two, four, and eight consecutive 8-bit units are called *bytes, halfwords, words,* and *doublewords,* respectively. For instructions operating on fixed-length fields, the operation implies one of these four lengths as the operand length. When the length of a field is not implied by the operation but is stated explicitly, the information format is said to have *variable field length.* Variable field lengths are variable by increments of a byte or a halfword.

The location of a field in main storage is identified by the address of the leftmost byte of the field. Except for doubleword information formats, fixed-length fields must be aligned in main storage on an *integral boundary.* That is, the field's location must have an address that is a multiple of its length in bytes. For example, a halfword integral boundary has an address which is a multiple of 2; a word integral boundary has an address which is a multiple of 4. Doubleword information formats must be aligned on a word boundary. Halfwords are the basic building blocks of instructions. Instructions, thus, must be located at addresses that are a multiple of 2.

Variable-length fields that are variable in increments of 1 byte may start at any byte address. On the other hand, variable-length fields that are variable in increments of 1 halfword must start at an address that is a multiple of 2. Figure 1-2 shows some examples of the information formats and alignment possibilities.

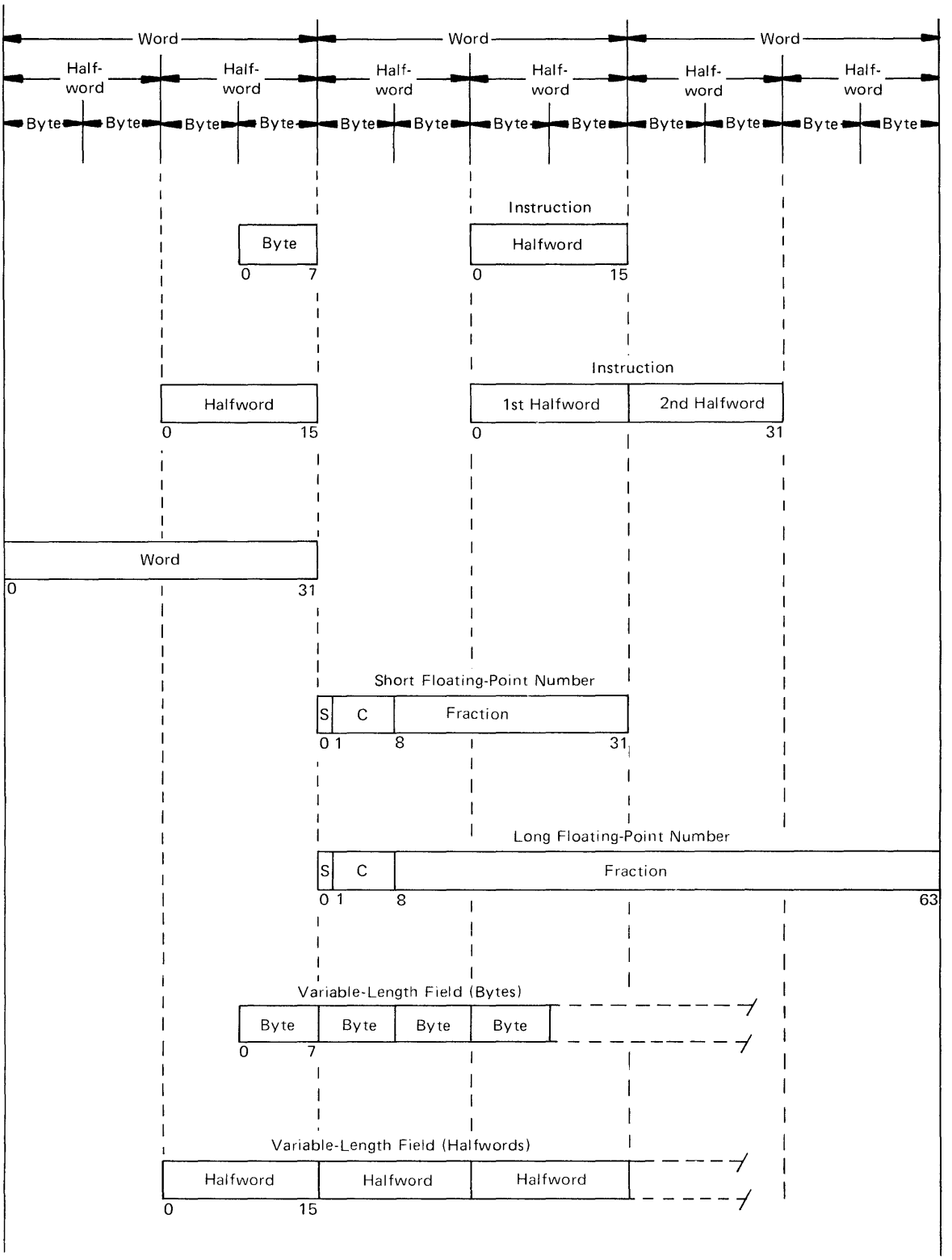


Figure 1-2. Examples of Information Formats and Alignment

Main Storage Addressing

Main storage is addressed in units of bytes. Byte locations in storage are numbered consecutively, left to right, starting with 0. Each number is considered the address of the corresponding byte location. Storage addresses are represented by unsigned 32-bit positive binary integers (see Figure 1-3).

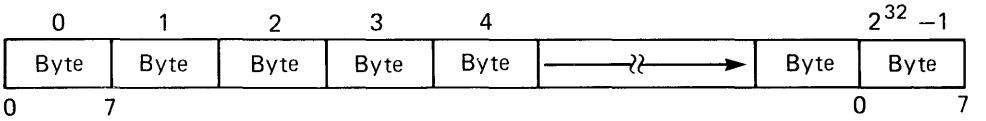


Figure 1-3. Storage Addresses

For purposes of addressing main storage, two types of addresses are recognized: real and logical. *Real addresses* are the addresses assigned to physical main storage locations. The addresses used by a program, or in a channel I/O operation, to refer to storage locations are called *logical addresses.* Logical addresses are transformed into real addresses by two means: dynamic address relocation and dynamic address translation. Dynamic address relocation is always applied to all logical addresses; dynamic address translation is controlled by a supervisory program. These facilities are described under "Dynamic Address Transformations" in this chapter.

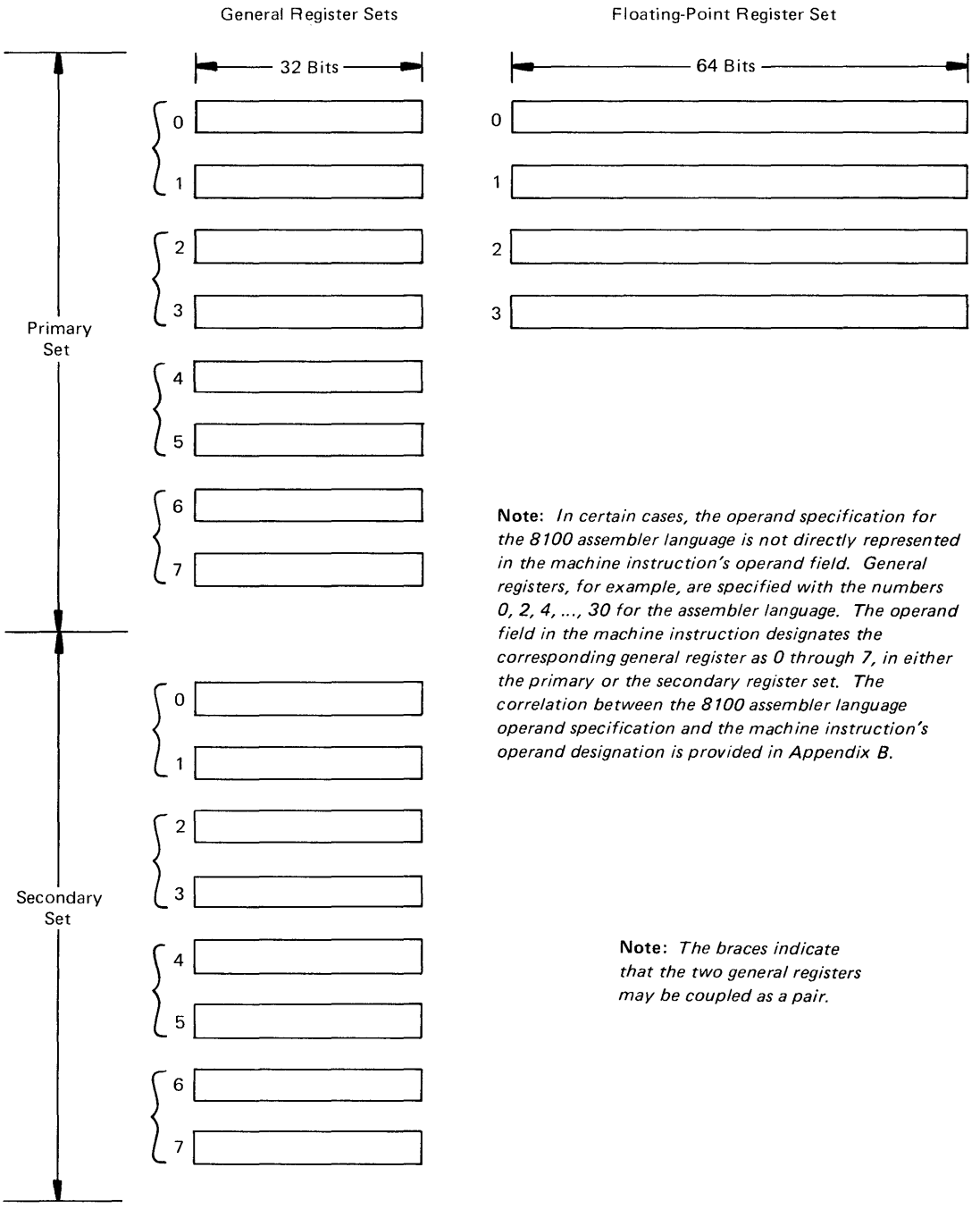
Registers

Registers are organized in *register groups* and *register sets.* Sixty-four sets of registers are provided in each of two register groups; each set consists of eight 32-bit registers numbered 0-7. One group of 64 register sets is known collectively as the *principal register group.* Of these, 12 sets are permanently assigned to hold control information, 4 are reserved, and the remaining 48 sets are available for use by programs as general-purpose registers. The second group of 64 register sets is known as the *adjunct register group.* Of these, 24 sets are permanently assigned to hold control information, and the remaining sets are reserved.

A program has two sets from the principal register group assigned to it and, thus, can address information in 16 general-purpose registers. General-purpose registers can be used for addressing and to hold operands and results in arithmetic and logical operations. Of the two register sets assigned to a program, one is designated the *primary set* and the other the *secondary set.*

For processor models having the optional floating-point feature installed, programs can address data in floating-point registers. Eight sets of floating-point registers are provided in the *floating-point register group.* All eight sets are available for assignment to programs. A program may address one set; each set consists of four 64-bit registers numbered 0-3.

The assignment of register sets to programs is discussed further under "Program-Environment Definition" in this chapter. Figure 1-4 shows the organization of registers assigned to a program. A description of all register groups is found under "Register Organization" in this chapter.



Note: *In certain cases, the operand specification for the 8100 assembler language is not directly represented in the machine instruction's operand field. General registers, for example, are specified with the numbers 0, 2, 4, ..., 30 for the assembler language. The operand field in the machine instruction designates the corresponding general register as 0 through 7, in either the primary or the secondary register set. The correlation between the 8100 assembler language operand specification and the machine instruction's operand designation is provided in Appendix B.*

Note: *The braces indicate that the two general registers may be coupled as a pair.*

Figure 1-4. Organization of Registers Assigned to a Program

Instructions

Operations performed by the PCE are controlled by sequences of instructions, which are the building blocks of programs. Each instruction consists of two major parts: (1) operation-code fields which specify the operation to be performed by the PCE, and (2) fields which specify the operands that participate in the operation.

In the following sections, operands, operand specifications, and instruction formats for the general instructions are discussed. Floating-point numbers, operand specifications, and instruction formats are described under "Floating-Point Instructions" in this chapter. PCE-control instructions are summarized in other sections of this chapter where the associated control functions are also described.

During instruction execution, the PCE monitors the existence of certain program exceptions, including those resulting from improper specification or use of instructions and data. These program exceptions normally result in an interruption of the program. (See "PCE Control" in this chapter for a discussion of interruptions.)

Note: *Instructions are described in Chapters 4 through 10, and Appendix A contains three lists that summarize these descriptions. These lists arrange the instructions by name, by mnemonic, and by instruction type.*

Operands

Operands can be grouped into three classes according to their location: operands in general registers, immediate operands, and operands in main storage.

Operands located in general registers may be 1-byte, 2-byte (halfword), or 4-byte (word) information units. The length of an operand in a general register is implied by the operation-code fields of the instruction. The distinction between the primary and secondary register sets applies in general to processing of byte operands and is explained in detail later in this chapter.

A general register can be used to hold multiple information units, each of which can be processed independently. Figure 1-5 shows the allocation of operands in a general register. Bit positions 0-15 are referred to as the *upper halfword* of a general register and bit positions 16-31 as the *lower halfword.* Byte operands may be located in bit positions 16-23 (upper byte) and 24-31 (lower byte) of a general register. Thus in the 16 registers assigned to a program there are a maximum of 32 halfword-operand locations and a maximum of 32 byte-operand locations. For operations which place the result in a general register, if the result is a byte or halfword, only the indicated bit positions of the register are used for the result; the remaining register bit positions are not changed.

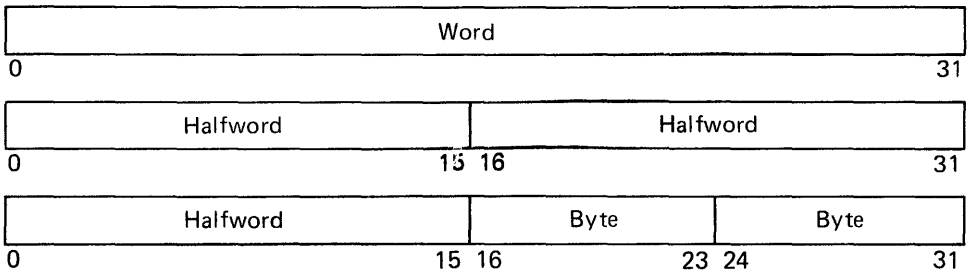


Figure 1-5. Data Unit Allocation for a General Register

Immediate operands are contained in a field within the instruction. The immediate data may be 4 bits or 1 byte, depending on the instruction.

Operands in main storage may have either a fixed length implied by the operation-code fields of the instruction, or a variable length specified by the contents of a general register.

Operation Classes and Instruction Formats

Instruction formats differ primarily in the method of operand specification and, in most cases, also in the allocation of fields for operation-code bits. Some instructions contain fields that vary somewhat from the general format descriptions.

An instruction is 1 or 2 halfwords long. Each instruction is in one of eight general formats. Six of the eight formats are used for all instructions other than floating-point instructions. The two remaining formats are used only for instructions provided with the optional floating-point feature.

The first four bit positions of all instructions are an operation-code field which identifies a general form for the instruction. Specification of the complete operation code generally requires more than 4 bits. Bits 12-15 of most instruction formats are used for this purpose; other fields may also be used in certain formats to completely specify the operation code.

Differences among the operand-specification parts of the various instruction formats depend on two factors: the number of operands required for the operation, and their locations. For purposes of description, operands are designated as first and second operands, and in some cases, third operands. In general, two operands participate in an operation and the result replaces the first operand. An exception is instructions with STORE in the name, where the result replaces the second operand.

Operation classes are identified by the locations of the operands that participate in the operation. The classes are: register-to-register, register-and-immediate, register-and-storage, storage-to-storage, and sequencing. Operation classes and the various instruction formats that are used for each operation class are discussed in the following sections. Note that instruction-format names express, in general terms, the form of operand specification provided, not the operation class that may employ a particular format. An instruction format is used for more than one operation class when these classes require similar forms of operand specification.

Register-to-Register

Operations in which the operands and result are held in general registers are called *register-to-register operations*. All fixed-point arithmetic and logical operations fall in this class. These operations generally use two operands and produce a result which always replaces the first operand. Except for multiplication and division, the first and second operands and the result all have the same length. In certain operations, such as data movement, only the second operand is used as input to the operation; the first operand is treated as an explicit result field having the same length as the second operand.

For most arithmetic and logical operations on halfword data, both operands are located in the lower halves of general registers. Several operations are also provided in which operands are located in the upper halves (arithmetic) or in either half (data movement). The operation code implies the register halves (upper or lower) that contain the operands. Figure 1-6 shows arithmetic and logical halfword operations, which can be described by the following general expressions:

Arithmetic and Logical:

$(R_1<16..31>) \leftarrow (R_1<16..31>)\ \textcircled{Y}\ (R_2<16..31>)$lower half
$(R_1<0..15>) \leftarrow (R_1<0..15>)\ \textcircled{Y}\ (R_2<0..15>)$upper half

Data Movement:

$(R_1<16..31>)$	$\leftarrow (R_2<16..31>)$	lower half to lower
$(R_1<0..15>)$	$\leftarrow (R_2<16..31>)$	lower half to upper
$(R_1<16..31>)$	$\leftarrow (R_2<0..15>)$	upper half to lower
$(R_1<0..15>)$	$\leftarrow (R_2<0..15>)$	upper half to upper

where:

$\leftarrow$	means "is replaced by"
$\textcircled{Y}$	represents an arithmetic or logical operation
( )	denotes "the contents of the register designated by"
R_1, R_2	designate general registers containing the first and second operand, respectively
<a..b>	designates bit positions a through b of the general register.

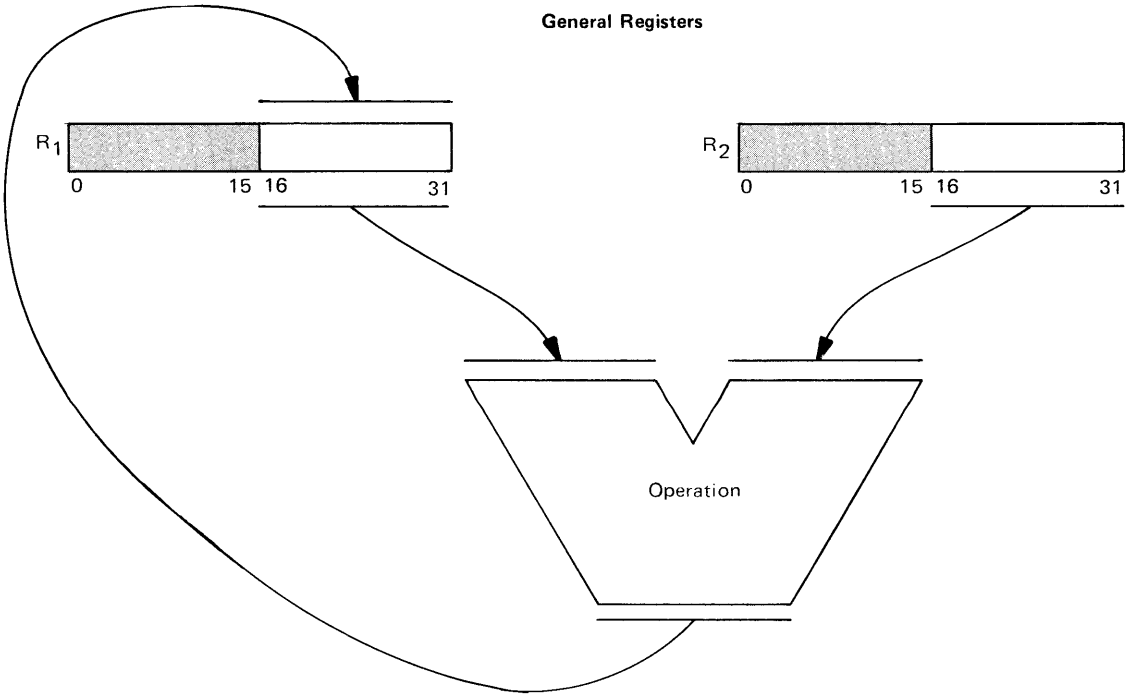


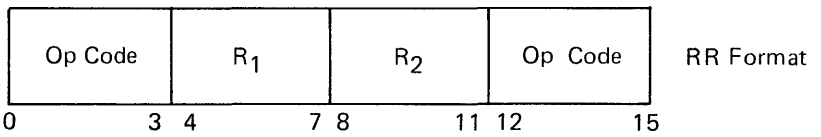
Figure 1-6. Example of Register-to-Register Operation (lower halfwords)

Register-to-register operations on byte operands are also provided. For arithmetic and logical operations on byte operands (see Figure 1-7), both the first and second operand must be in the same register set. Two instructions are provided for each such operation on byte operands: one obtains both operands from the primary register set; the other obtains both from the secondary set.

Data-movement operations are provided in which the operand locations may be in either set. Operation codes indicate the register set(s) that contains the operands. General expressions for register-to-register operations on byte data, where *r* is used to indicate specification of a byte operand, are:

$(r_1) \leftarrow (r_1)$ Ⓨ (r_2) arithmetic, logical
$(r_1) \leftarrow (r_2)$ data movement

Register-to-register operations require specification of two register operands. The RR format is used for these operations. For this, as well as all other formats, the format name expresses, in general, the types of operand specification, not the operation class. In the format shown below, two registers are specified and thus the name is RR. The RR format is also used for other classes of operations, such as register-and-storage, where specification of two registers is required for a particular instruction.



For register-to-register operations on halfword operands, each R field designates one of the general registers that contains an operand: R_1 designates the first-operand register and R_2 designates the second-operand register.

Operations on byte operands in general registers are also specified by instructions in the RR format. In this case, the register-specification fields of the instruction designate one of the 16 byte-operand positions in either the primary or the secondary register set. The particular set is designated by an operation-code field of the instruction.

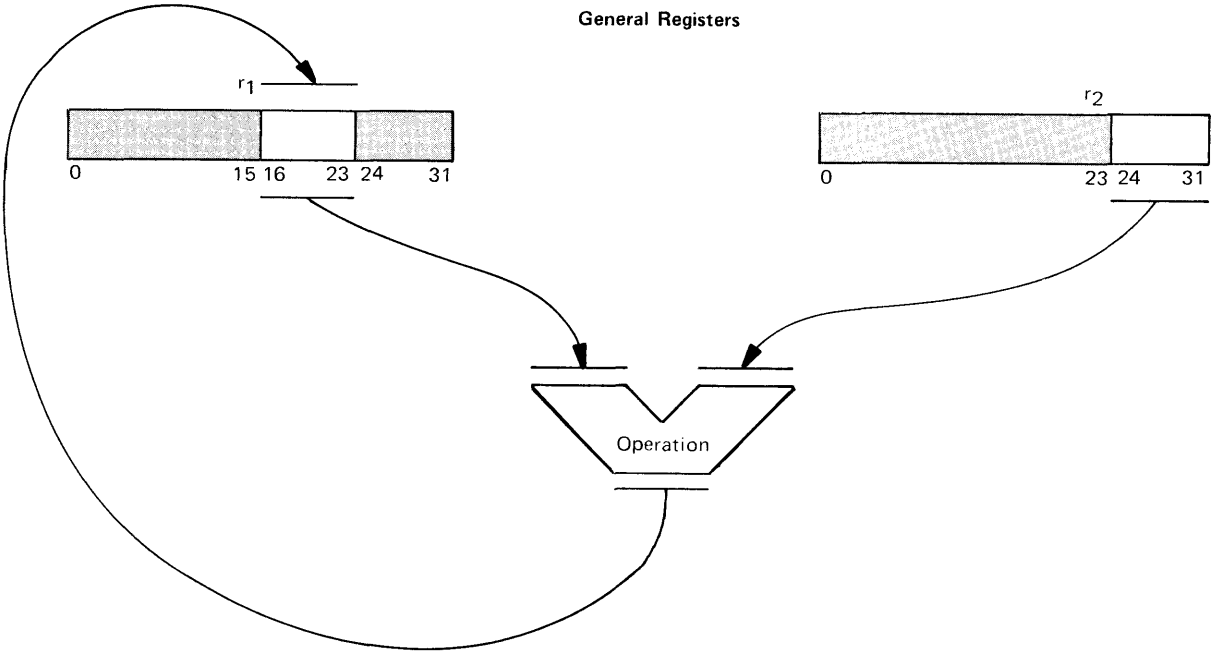
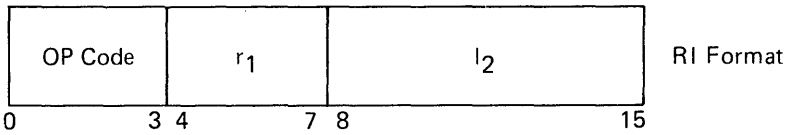


Figure 1-7. Example of Register-to-Register Operation (bytes)

Register-and-Immediate

Operations in which the first operand is located in a register and the second operand is an immediate field in the instruction are called *register-and-immediate operations*. The result always replaces the first operand. Figure 1-8 shows an example of this type of operation.

For register-and-immediate operations on byte operands, the instruction format shown below is used. Because a register and an immediate field are specified, this format is called the RI format. In the RI format, the r_1 field designates one of the 16 byte-operand positions in the primary register set.



Certain register-and-immediate arithmetic operations are specified with RR format instructions in which a 4-bit immediate-data field is defined in place of one of the register-specification fields.

The RI format is also used for other classes of operations, such as programmed input/output (PIO) and PCE-control. For PIO operations, the immediate field contains a command code for an I/O device; for PCE-control instructions, it is used as an extension to the operation code. For PCE-control instructions, instruction bit positions 4–6 designate the first operand byte (located in register bit positions 16–23) or word of a general register in the primary register set. When a PCE-control operation does not use a register operand, the r_1 field should contain 0's.

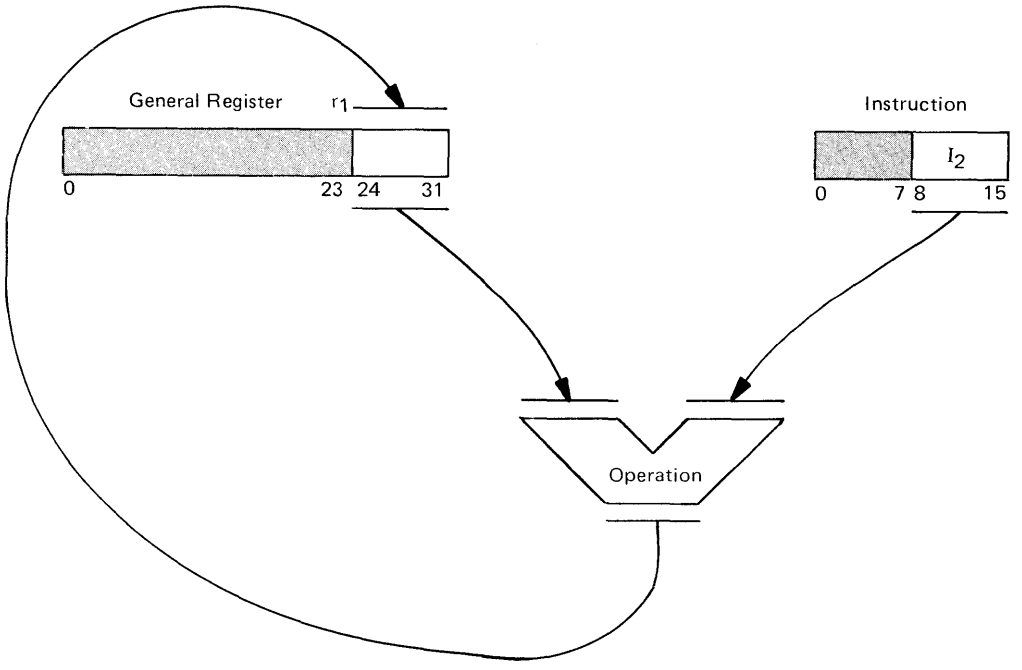


Figure 1-8. Example of Register-and-Immediate Operation

Register-and-Storage — Address Generation

All operations that refer to information in main storage (register-and-storage, storage-to-storage), or that alter the sequence of instructions executed, require specification of logical main-storage addresses. Addresses are specified by means of instruction formats that designate the contents of a general register as all or part of the address. All logical addresses used by the program to refer to main storage are treated as unsigned 32-bit positive binary integers.

Operations in which the first operand is located in a register and the second operand is located in main storage are called *register-and-storage operations.* Except for floating-point arithmetic, all register-and-storage operations are used only for data movement between a register and main storage.

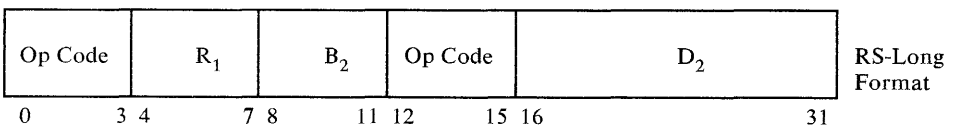
For register-and-storage operations, the simplest address specification is obtained when the address is contained in one of the general registers. When this address specification is used, the instruction has the RR format. The register containing the first operand is specified by the R_1 field; the R_2 field designates a register that contains the address of the second operand. The address so specified is usually considered an index; that is, the address is used to select an element from a one-dimensional array of like elements. Several instructions are provided that include modification of the address as part of the operation.

More generally, address specification involves generation of an effective storage address. An *effective storage address,* E, is an unsigned 32-bit positive binary number given by E = B + D. Here, B represents an unsigned 32-bit positive number called the *base address.* Base addresses can be used for independently addressing different areas of storage. In many types of processing, the base address is useful for locating a data structure (such as an array or record). The base address may also be used for indexing purposes; for example, to select a record from an array of records having a common format.

D represents a binary number, usually signed, called the *displacement,* which is taken from a field in the instruction. The displacement provides for addressing relative to the location specified by the base address; for example, for addressing an elementary unit or field within a record.

The base and displacement are added as binary integers with the displacement logically expanded to 32 bits. The result of this addition is used as the effective address (see Figure 1-9).

The principal instruction format for register-and-storage operations using a base and displacement is RS-Long, which is shown below. In this format, the R_1 field designates the register containing the first operand; the B_2 and D_2 fields designate the components of the second-operand address. The B_2 field designates the general register containing the base address; any of 15 general registers may be used to hold the base address. The D_2 field is used to represent the displacement as a signed binary integer.



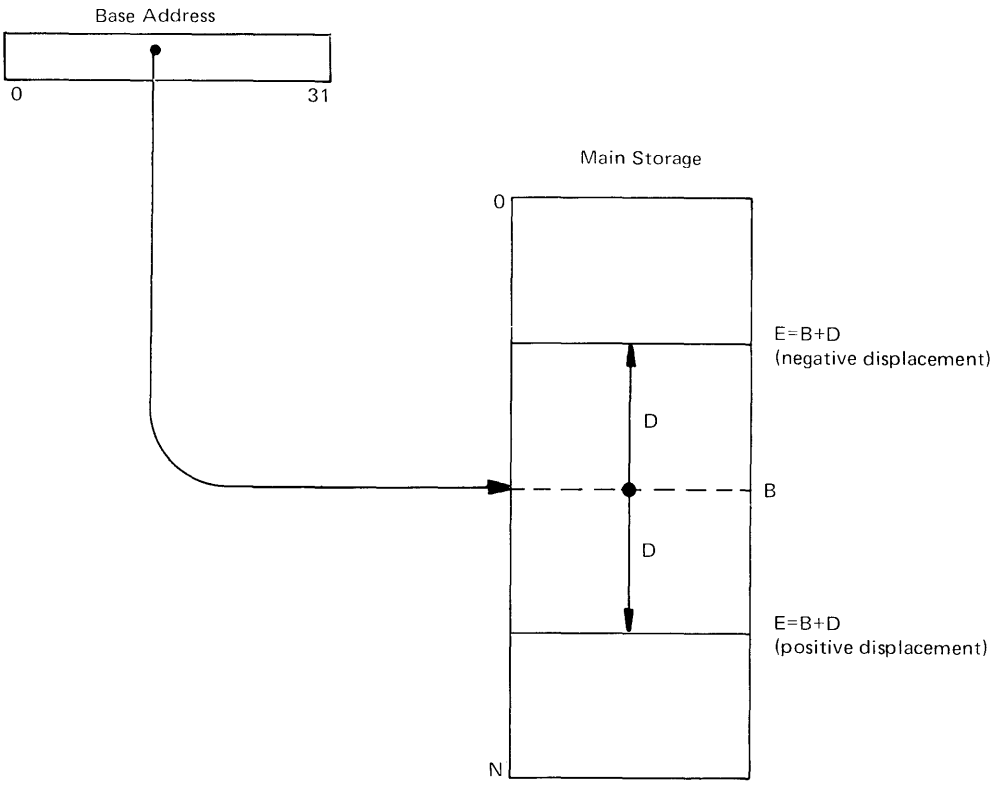
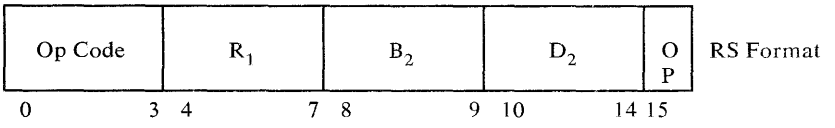


Figure 1-9. Address Generation Using Base and Displacement Values

A special case of base-and-displacement address generation is provided when the B_2 field of an RS-Long format instruction contains all 0's. In this case, the updated instruction address (address of the next sequential instruction) is used as the base, instead of the contents of a general register. This provides for addressing instructions and data relative to the current instruction's address. It is particularly useful for branch addresses and references to local data because a register is not needed to hold the base.

Base-and-displacement address generation for certain register-and-storage operations is provided in an abbreviated format, called the RS format, which is shown below. The B_2 field designates one of four general registers that can contain the base address. The D_2 field is used to represent an unsigned binary integer.



Initialization, modification, and testing of addresses in general registers can be performed using the operations for data movement and fixed-point arithmetic. Further, an instruction can designate the same general register as containing an address and as the location of an operand. Address generation is completed before the register is used for an operand.

One instruction that provides several functions is the LOAD ADDRESS instruction, which has the RS-Long format. The operation of LOAD ADDRESS is given by the following expressions:

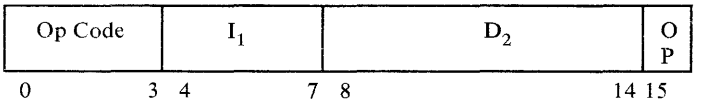
$(R_1) \leftarrow (B_2) + D_2$	where B_2 designates a general register
	or
$(R_1) \leftarrow IA + D_2$	where IA is the updated instruction address (the B_2 field is all 0's)

This instruction performs the calculation E = (B) + D and places the 32-bit value, E, in the general register specified by R_1 LOAD ADDRESS is also useful as an add-immediate operation in which the D_2 field is used as signed immediate data. When the same general register is specified for R_1 and B_2, a convenient update of that register's contents is obtained.

Sequencing

Sequencing operations are used to alter the sequential order of instruction execution. These operations require specification of a main storage address that designates the new sequence of instructions to be executed. Most sequencing operations are specified in both RR and RS-Long format instructions. In the RR format, the R_2 field designates a general register containing the address. When specified in the RS-Long format, the address is formed by a base and displacement calculation.

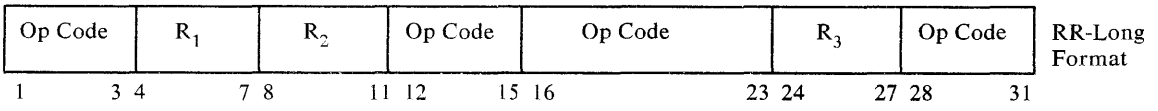
A special case of address specification in sequencing operations applies to instructions with JUMP in the name. These operations are specified with J format instructions (shown below). The D_2 field is used as a signed displacement which is added to the updated instruction address to form the new instruction address.



Storage-to-Storage

Operations on variable-length fields are called *storage-to-storage operations.* Operand specification for these operations requires two addresses and a field length (both fields are the same length). The length may vary from 1 to 256 units; the unit is a byte or halfword, depending on the operation.

Storage-to-storage operations are specified in the RR-Long instruction format which is shown below. The starting addresses of the first and second operands are contained in the registers designated by the R_1 and R_2 fields, respectively. The length is contained in bit positions 24-31 of the register designated by the R_3 field.



In storage-to-storage operations, the data units are processed proceeding from left to right. As the operands are processed, the storage addresses in the R_1 and R_2 registers are increased accordingly. The length in the R_3 register is used as a count of the units remaining to be processed and is reduced accordingly until a count of 0 is reached. Because address and count information is maintained in general

registers, these operations are interruptible and can be resumed automatically following an interruption. This capability prevents delaying the execution of other, high-priority programs while a long storage-to-storage operation is being performed.

General Instructions

The following sections briefly summarize most general instructions, grouped according to the operations they cause the PCE to perform. The types of general instructions are: data movement, fixed-point arithmetic, logical operations, and sequencing. In this discussion, the general instructions are described in terms of these types as well as in terms of the classes of operations and the instruction formats discussed in the previous section. Certain specialized operations are not discussed.

Data Movement

Operations are provided that allow data to be moved unchanged (1) from one general register to another, (2) between main storage and a general register, and (3) from one main storage location to another.

Register-to-Register: In register-to-register load operations, the unchanged second operand replaces the first operand. Four LOAD (halfword, register) operations are provided, one for each combination of operand positions in upper and lower halves of two general registers (see Figure 1-10).

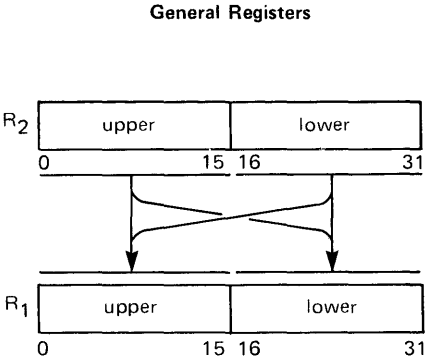


Figure 1-10. LOAD (halfword, register) Operations

Movement of byte operands is accomplished by the LOAD (byte, register) operation. Four instructions of this type are provided, one for each combination of movement among primary and secondary register sets (see Figure 1-11).

Register-and-Storage: Operations for data movement from main storage to a register (LOAD) and from a register to main storage (STORE) are provided. For all LOAD or STORE operations that move 1 byte of information, the register operand is located in the primary register set.

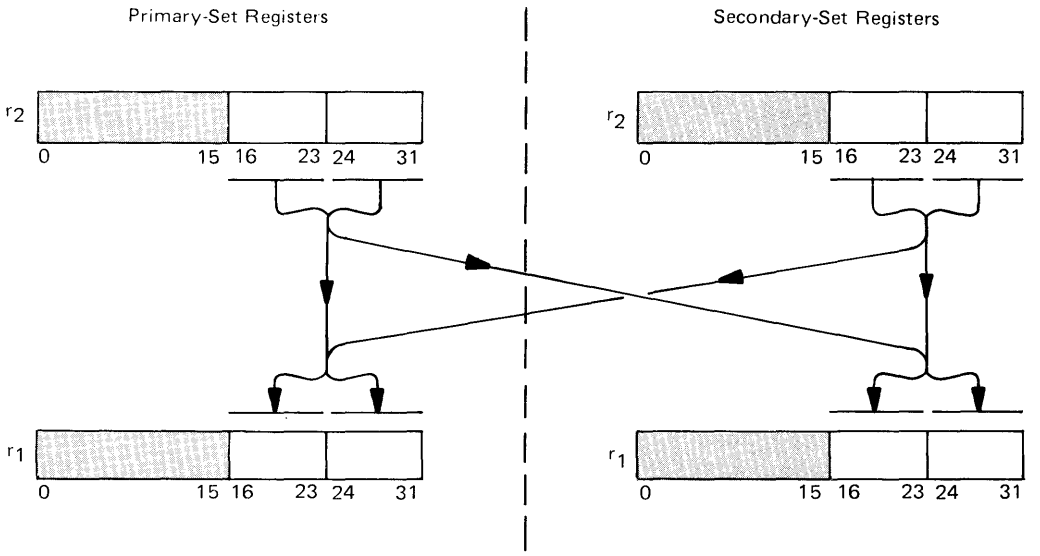


Figure 1-11. LOAD (byte, register) Operations

The following are general expressions for data-movement operations specified in the RS-Long format:

(R_1) <— $MS[(B_2) + D_2]$	LOAD (word)
$(R_1<16..31>)$ <— $MS[(B_2) + D_2]$	LOAD (halfword)
(r_1) <— $MS[(B_2) + D_2]$	LOAD (byte)
$MS[(B_2) + D_2]$ <— (R_1)	STORE (word)
$MS[(B_2) + D_2]$ <— $(R_1<16..31>)$	STORE (halfword)
$MS[(B_2) + D_2]$ <— (r_1)	STORE (byte)

where $MS[(B_2) + D_2]$ represents the contents of a main-storage location with an address specified by the base and displacement.

These instructions provide for relative addressing of up to 32,768 bytes preceding the base address and 32,767 bytes beyond it. LOAD (halfword) and STORE (halfword) instructions are also provided in an abbreviated specification that uses the RS format. These instructions provide for addressing data structures consisting of up to 32 contiguous halfwords.

Register-and-storage data movement operations are also provided with RR format instructions. These LOAD and STORE operations comprise a set of 12 instructions useful for processing one-dimensional arrays (or stacks) of halfword or byte elements. The storage address contained in the register designated by R_2 is considered to be an index (or stack pointer). Operations are provided that include increasing or decreasing the address by the length of the data unit (1 for bytes, 2 for halfwords). See Figure 1-12 for an example from this group of instructions.

The 12 instructions are obtained from all combinations of the following specifications:

Operation	**Data Unit**	**Addressing**
LOAD	byte	index unchanged
STORE	halfword	index post-incremented
		index pre-decremented

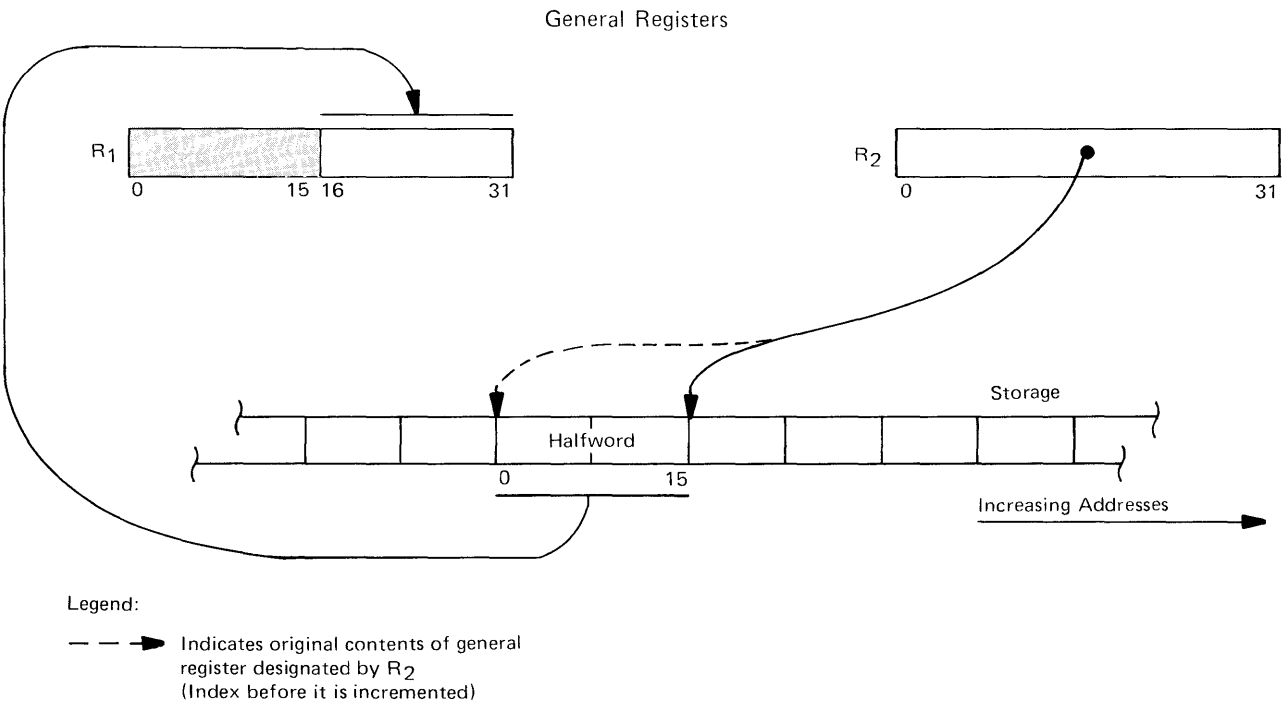


Figure 1-12. LOAD (halfword, with index incremented) Operation

It is often convenient to transfer information between main storage and multiple registers in a single operation (for example, as part of subroutine linkage). Groups of eight halfword operands may be addressed as a single unit — the *quadrant.* In the 16 general registers assigned to a program, there are four quadrants as illustrated in Figure 1-13.

The instructions LOAD (halfwords, quadrant) and STORE (halfwords, quadrant) transfer 8 halfwords between the consecutive operand positions in a quadrant and consecutive locations in main storage.

The instruction LOAD (byte, register-immediate) is provided to place a byte of immediate data into a general register in the primary set.

Storage-to-Storage: Movement of variable-length fields from one main storage location to another is accomplished with MOVE operations. Two RR-Long format instructions are provided: one that moves a field of consecutive byte data units, and one that moves a field of consecutive halfword data units (see Figure 1-14).

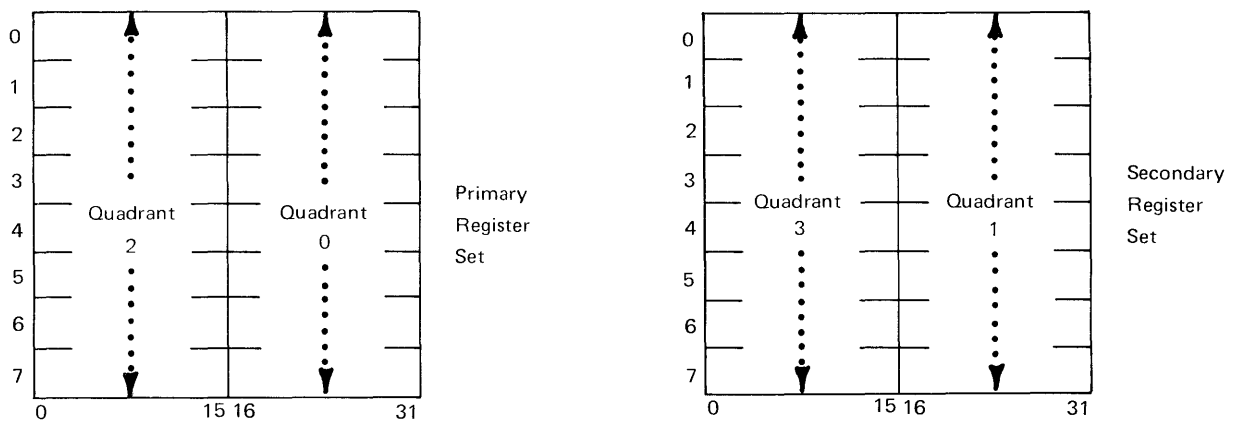


Figure 1-13. General Register Quadrants

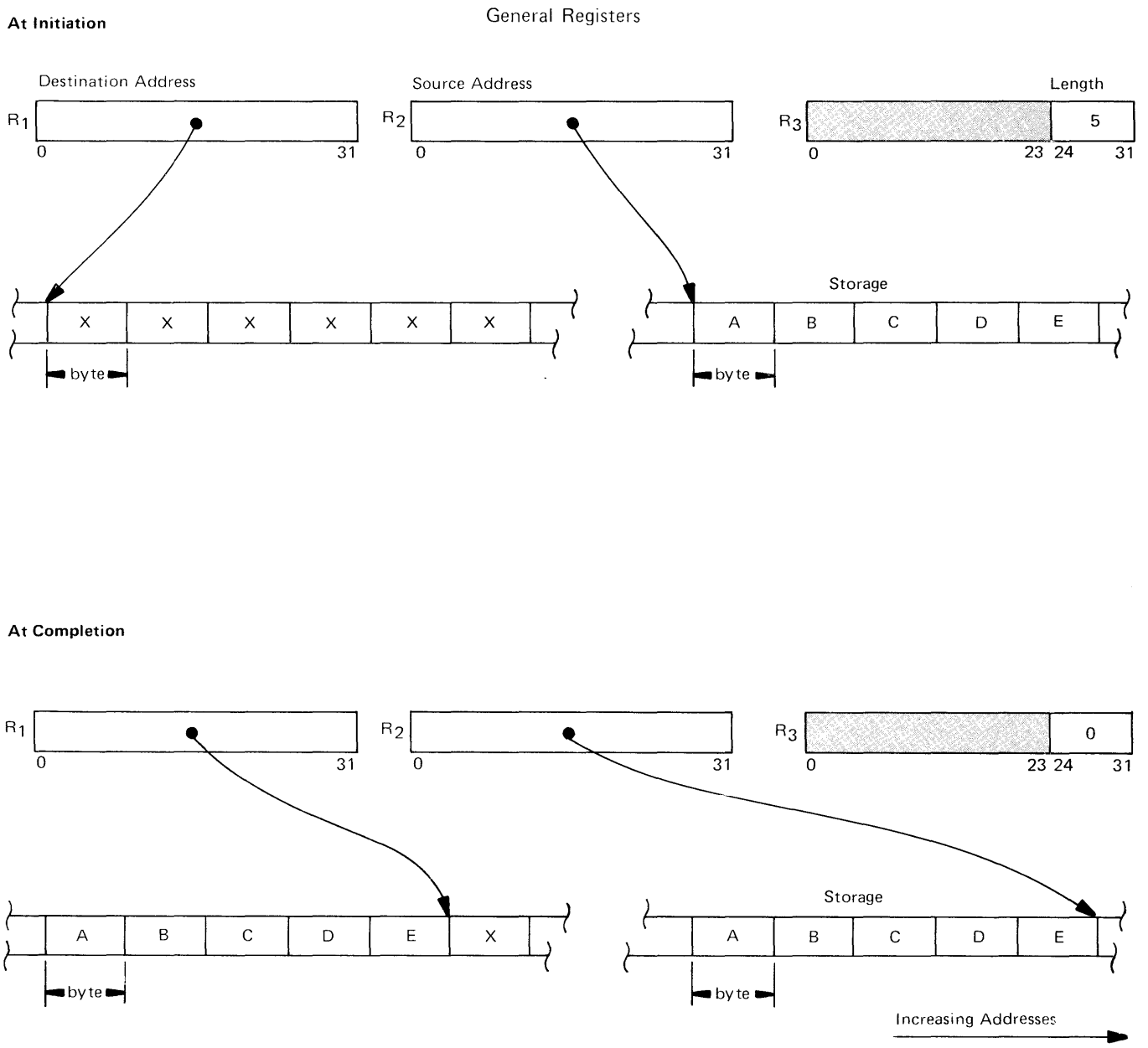


Figure 1-14. Example of MOVE (bytes, storage)

Fixed-Point Arithmetic

The basic arithmetic operand is the 16-bit fixed-point signed binary number. Byte operands (8-bit fixed-point signed binary numbers) may also be specified for most operations.

Fixed-point numbers may be signed or unsigned integers. In an unsigned number, all bits are used to represent the magnitude of the number. In signed numbers, the leftmost bit indicates the sign. Positive numbers are represented in true binary notation with the sign bit set to 0. Negative numbers are represented in twos-complement binary notation with a 1 in the sign-bit position. The value zero is represented by all bits being 0.

Most fixed-point arithmetic is performed with register-to-register operations specified in the RR instruction format. Unless otherwise stated, the halfword operands occupy the lower half (bit positions 16-31) of a general register. Byte operands may be located in either the primary or the secondary register set; however, both operands are located in the same set.

ADD and SUBTRACT operations are provided for both 8-bit and 16-bit binary numbers. MULTIPLY and DIVIDE operations are provided for unsigned 16-bit multipliers, multiplicands, and divisors. Products and dividends are unsigned and occupy 32 bit positions. Quotients and remainders are unsigned 16-bit binary numbers.

Extended-and-mixed-precision fixed-point arithmetic is made convenient by use of twos-complement representation and by provision for recognition and retention of the carry from one field to another. One example of extended-precision arithmetic, useful particularly for address modification, applies to 32-bit binary numbers contained in general registers. ADD WITH CARRY and SUBTRACT WITH CARRY operations are provided that combine operands in the upper halves (bit positions 0-15) of two general registers with the carry resulting from a previous operation on the lower halves (see Figure 1-15).

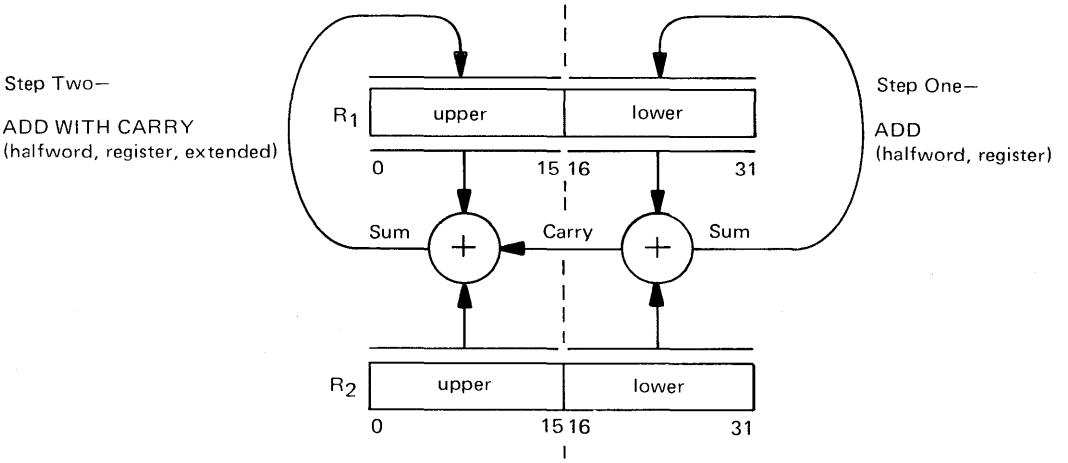


Figure 1-15. Example of Extended-Precision Addition

ADD WITH CARRY and SUBTRACT WITH CARRY operations are also provided that combine operands in the lower halves of general registers with the value of the carry from a previous operation. Operations on both 8-bit and 16-bit fields are provided.

Addition and subtraction of halfword and byte operands may also be performed with register-and-immediate operations.

All arithmetic operations indicate certain result conditions that reflect the outcome of the operations (equal to 0, greater than 0, less than 0, overflow, or carry). These result conditions can be tested by operations that control the sequence of instructions executed.

Arithmetic-compare operations are provided for comparing 8-bit, 16-bit, and extended-precision fixed-point binary numbers. COMPARE operations are similar to the corresponding SUBTRACT operations except that the first operand is not replaced (no result is stored).

Logical Operations

A set of instructions is provided for the logical manipulation of data. The set of logical instructions includes comparing, boolean, bit testing, shifting, and rotating operations.

As in fixed-point arithmetic, the boolean operations (AND, OR, EXCLUSIVE OR) are provided as register-to-register operations. Fixed-length logical data may be processed in 8-bit or 16-bit lengths. The boolean operations are applied bit-by-bit. All boolean operations indicate certain result conditions reflecting their outcome (all 0's, all 1's, or mixed 0's and 1's).

Halfword operands are taken from the lower half of a general register. Byte operands may be located in either the primary or the secondary register set; both operands are located in the same set. Boolean operations on byte operands in the primary register set are also provided as register-and-immediate operations.

Shifting and rotating operations are provided for use in isolating, concatenating, and aligning groups of contiguous bits. SHIFT LEFT and ROTATE LEFT operations are provided in the RR instruction format for both 8-bit and 16-bit fixed-length fields. In this format, the number of bits moved is specified with immediate data in place of one R field.

SHIFT operations cause bits shifted out of the high-order bit positions of the operand to be lost; 0's are supplied in vacated low-order bit positions (see Figure 1-16). ROTATE operations wrap the operand; that is, bits shifted out of the high-order bit positions are entered consecutively in vacated low-order bit positions.

Operations are provided for testing and comparing logical data. TEST (byte, register-immediate) is provided for testing byte operands in the primary register set. The operation selects for testing those bits in the byte that are designated by 1's in the corresponding bit positions of a mask. The collection of bits so selected are tested for three conditions: all 0's, all 1's, and mixed 0's and 1's. If the mask and operand are identical, this is also indicated.

Logical comparison of two variable-length fields in storage is provided by the COMPARE LOGICAL operations. Two RR-Long format instructions are provided: one that compares fields of consecutive byte data units, and one that compares fields of consecutive halfwords (see Figure 1-17). Fields are logically compared by treating each byte or halfword as an unsigned positive binary number. The operation ends when an inequality between two corresponding units is found or when the end of the fields is reached. Three result conditions may be indicated: equal, low, or high.

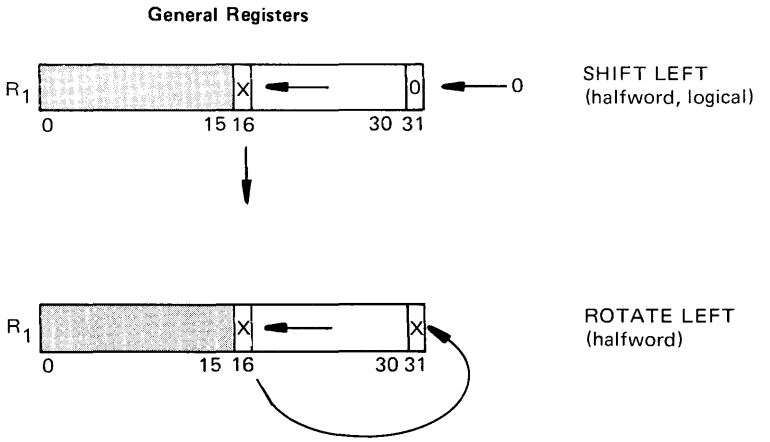


Figure 1-16. SHIFT and ROTATE (halfwords) Operations

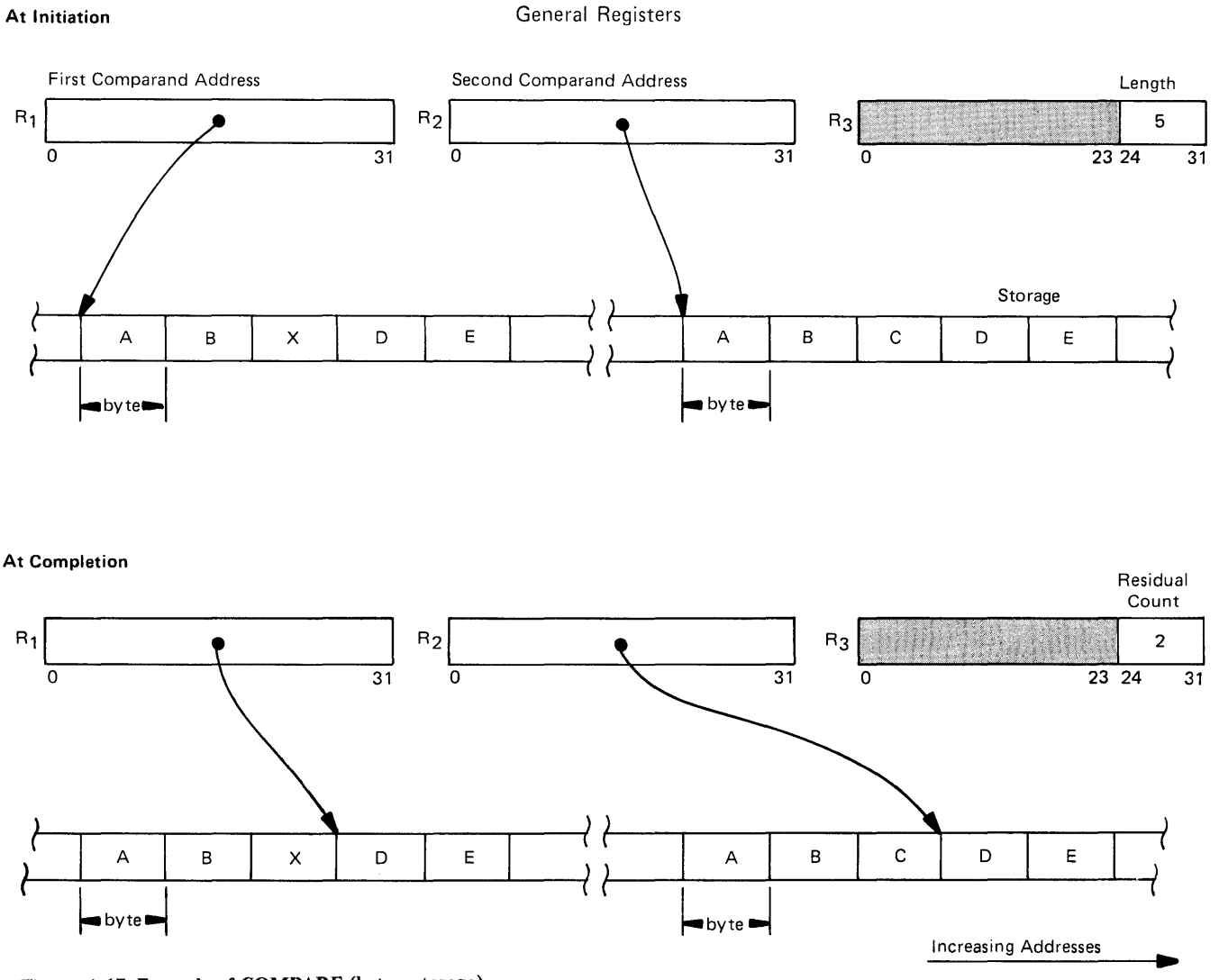


Figure 1-17. Example of COMPARE (bytes, storage)

Sequencing

Normally the PCE takes instructions in sequence. After an instruction is fetched from the location specified by the instruction address, the instruction address is increased by the number of bytes in the instruction. This addition is effectively performed before the fetched instruction is executed.

The normal sequence of instruction execution may be changed by use of branching operations to perform decision making, loop control, and subroutine linkage.

Conditional branching is accomplished by BRANCH ON CONDITION and JUMP ON CONDITION operations. These operations test five logical entities, called *result conditions,* which describe the outcome of arithmetic, logical, and I/O operations. Each of the five result conditions can be set in one of two possible states: *indicated* or *not-indicated.* A conditional branch or jump operation can select combinations of result conditions as a criterion for branching.

The specific meaning associated with any of the five result conditions depends on the particular instruction for which result conditions are specified. For example, results such as 0 sum, first operand high, equal, overflow, and non-0 may be indicated. Once set, the states of the result conditions remain unchanged until modified by a subsequent operation. Each operation that indicates result conditions sets the state of all five conditions. The states are derived from condition-indicator bits held in the PCE.

The instructions for conditional branching include a mask field. The mask is used to select result conditions to be tested. If a selected result condition is indicated, the branch occurs. An unconditional branch can also be specified. A branch may be made if any one of several result conditions is indicated. This is accomplished using multiple mask positions to select the pertinent conditions for the branch.

The branch address for BRANCH ON CONDITION specified with an RS-Long format instruction is formed by base and displacement address calculation. This instruction provides for addressing 65,536 bytes preceding the base address and 65,534 bytes beyond it. In the RR format, the branch address is taken from the general register designated by the R_2 field.

JUMP ON CONDITION is specified with a J format instruction. The range covered by the displacement, which is added to the updated instruction address, is 126 bytes preceding the JUMP ON CONDITION instruction and 128 bytes beyond it.

A more specialized conditional branch instruction is BRANCH ON COUNT (byte, register), which reduces a byte-operand count field by 1 and branches if the count is not 0. Another specialized conditional branch operation is provided by JUMP ON BIT ZERO (halfword). In this instruction, a specified bit position (any of positions 16-31) of an implied general register is tested. If the specified bit position contains a 0, a jump is taken. The jump address is formed in the same way as for JUMP ON CONDITION.

Subroutine linkage is provided by a BRANCH AND LINK operation (both RR and RS-Long formats). The updated instruction address is saved in a designated general register. The saved address may be used directly to effect a return from a subroutine. After the link address is saved, control is transferred to the specified

branch address. In the RR format, if the R_2 field contains 0's, the link address is saved but no branch takes place. This provides a convenient means of initializing a base register. In the RS-Long format, the new address is calculated from the specified base and displacement, similar to BRANCH ON CONDITION.

Floating-Point Instructions

The optionally available floating-point feature provides instructions used in calculations with numbers having a wide range of magnitude. These instructions yield results scaled to preserve precision.

Floating-point numbers are specified in either of two fixed-length formats, short or long, which are illustrated in Figure 1-2. The fraction is expressed in 4-bit hexadecimal digits having a radix point to the left of the high-order digit. The sign of the fraction is indicated in the leftmost bit of the representation. The fraction is represented in true binary notation.

The proper magnitude is determined by considering the fraction as multiplied by a power of the fraction radix (16). The characteristic expresses this power and is represented as a 7-bit excess-64 binary number. The power can range from -64 to 63.

Floating-point data may be normalized or unnormalized. Unnormalized numbers have at least one high-order 0 digit (4 bits) in the fraction. The range covered by the magnitude (M) of a floating-point number is:

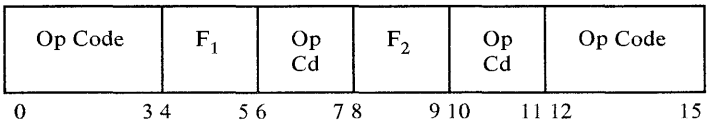
$$16^{-65} \leq M \leq 16^{63}$$

or, in decimal terms, approximately:

$$5.4 \times 10^{-79} \leq M \leq 7.2 \times 10^{75}$$

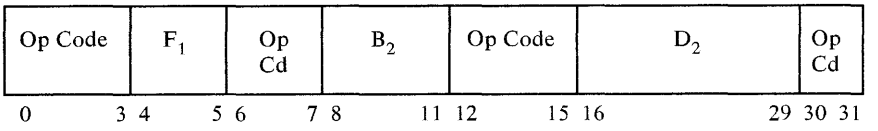
The format of floating-point numbers (short or long) is determined by the value of a *precision-mode* bit held in the PCE. A short-format operand occupies bit positions 0-31 of a floating-point register; bit positions 32-63 are ignored and remain unchanged in operations calling for short operands. A long-format operand occupies all 64 bit positions of a floating-point register. The instruction SET PRECISION MODE is provided to define the length of operands used in all subsequent floating-point operations.

Register-to-register floating-point operations are specified in the FF instruction format shown below. In the FF format each F field designates one of the floating-point registers; F_1 designates the register containing the first operand and F_2 designates the register containing the second operand. In register-to-register operations, the result always replaces the first operand.



FF Format

Register-and-storage floating-point operations are specified in the FS instruction format shown below. In this format the F_1 field designates the floating-point register containing the first operand. The result of these operations always replaces the first operand.



FS Format

The second-operand address is generated from a base and displacement. The floating-point instructions provide for addressing 32,768 bytes preceding the base and 32,764 bytes beyond it. Any of the 16 general registers may be used to hold the base address. Use of the instruction address as a base address is not provided by these instructions.

Data movement operations are provided that transfer floating-point numbers from one floating-point register to another, or between floating-point registers and main storage.

A register-to-register LOAD operation transfers the floating-point number (unchanged) from one register to another. Other register-to-register LOAD operations (1) allow changes to the sign of the fraction as well as movement of data, (2) test the sign of the fraction in the number moved, or (3) round a long-format number to a short-format number.

Data is transferred unchanged between main storage and the floating-point registers by two instructions, LOAD and STORE, both specified in the FS format.

Each arithmetic operation is provided in two forms: register-to-register and register-and-storage. General expressions for these two forms are:

$(F_1) \leftarrow (F_1) \textcircled{Y} (F_2)$ register-to-register
$(F_1) \leftarrow (F_1) \textcircled{Y} MS[B+D]$ register-and-storage

ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPARE operations are provided. All operations except MULTIPLY generate a result that has the same length as the operands. MULTIPLY produces a long-format result for both short- and long-format operands. The result conditions are set to reflect the outcome of addition, subtraction, and comparison operations.

The initial operands of any operation need not be normalized. Automatic normalization of the result is provided for the arithmetic operations. ADD UNNORMALIZED and SUBTRACT UNNORMALIZED operations are provided (in both FF and FS formats), which produce the result without normalization.

Intermediate results in all addition, subtraction, comparison, and multiplication operations may have one additional low-order digit. This low-order digit, the *guard digit*, increases the precision of the final result.

Monitoring of three exceptions — exponent underflow, exponent overflow, and lost significance (vanishing fraction) — is provided under program control. A SET MASK operation is provided for each monitored exception to allow it to be controlled separately. Division by 0 is monitored at all times.

Program—Environment Definition

An environment for program execution is defined in terms of the processing resources and the status information required for proper execution. Processing resources required for the program include registers and the set of addresses used for references to main storage. A program environment is described by two vectors of control information: a program status vector (PSV) and an address control vector (ACV).

In general, the PSV is used to control instruction sequencing, to define general register assignments, and to hold the status of the PCE in relation to the program. The PSV that describes the program being executed is called the *current PSV.* If a different PSV is introduced as the current PSV, the state of the PCE is changed; execution then proceeds using the description contained in the new PSV. When the current PSV is stored, the state of the executing program is preserved so that execution may later be resumed.

All addresses used by the program to refer to main storage are called *logical addresses.* These addresses are not used directly to refer to physical main-storage locations. The set of logical addresses that may be used by a program is called its *logical address space.* Each program can be assigned a distinct logical address space. The size of the logical address space is defined by an ACV.

An ACV is associated with each PSV; when a new PSV is introduced, a new ACV is introduced also. The new ACV describes the logical address space available to the program. The introduction of a new PSV/ACV pair and the storing of PSV information are discussed under "PCE Control" in this chapter.

The dynamic address relocation and translation facilities are used to associate addresses in a logical address space with physical locations in main storage. ACV fields are used to control these address transformations. Further details are given under "Dynamic Address Transformations" in this chapter.

When the optional floating-point feature is installed, a third control vector is used in defining a program environment: the floating-point status vector (FSV). The FSV provides additional information for the proper execution of programs using the floating-point instructions. In general, the FSV is used to define floating-point register assignments, to specify data formats, to control the reporting of exceptions, and to indicate exceptions and equipment malfunctioning.

The definition of a program-execution environment by a PSV/ACV pair and an FSV is illustrated in Figure 1-18. The PSV is 64 bits long. The instruction address field of the PSV contains the logical address of the next instruction to be executed. The primary and secondary general-register sets assigned to the program are identified by two fields in the PSV. Other fields in the PSV include the condition indicators, the program mode, and the program information code.

The program information code is used primarily to report information resulting from the monitoring of program exceptions by the PCE. The condition indicators reflect the results of arithmetic, logical, and I/O operations.

To ensure the integrity of a supervisory program, instructions that alter PCE-control information are not available for general use. Only those programs whose PSV specifies the appropriate authorization may use these instructions. A hierarchy of instruction-use authorization is provided by four modes specified in the program-mode field of the PSV: master, supervisor, input/output, and application.

When the FSV is present, the floating-point register set available to the program is identified by the register-set field. The precision-mode field specifies the data format (short or long) used for floating-point operations. The remaining two fields (exception mask and exception indicators) are used for controlling, and for reporting the results from, the PCE's monitoring of program exceptions such as exponent overflow and lost significance.

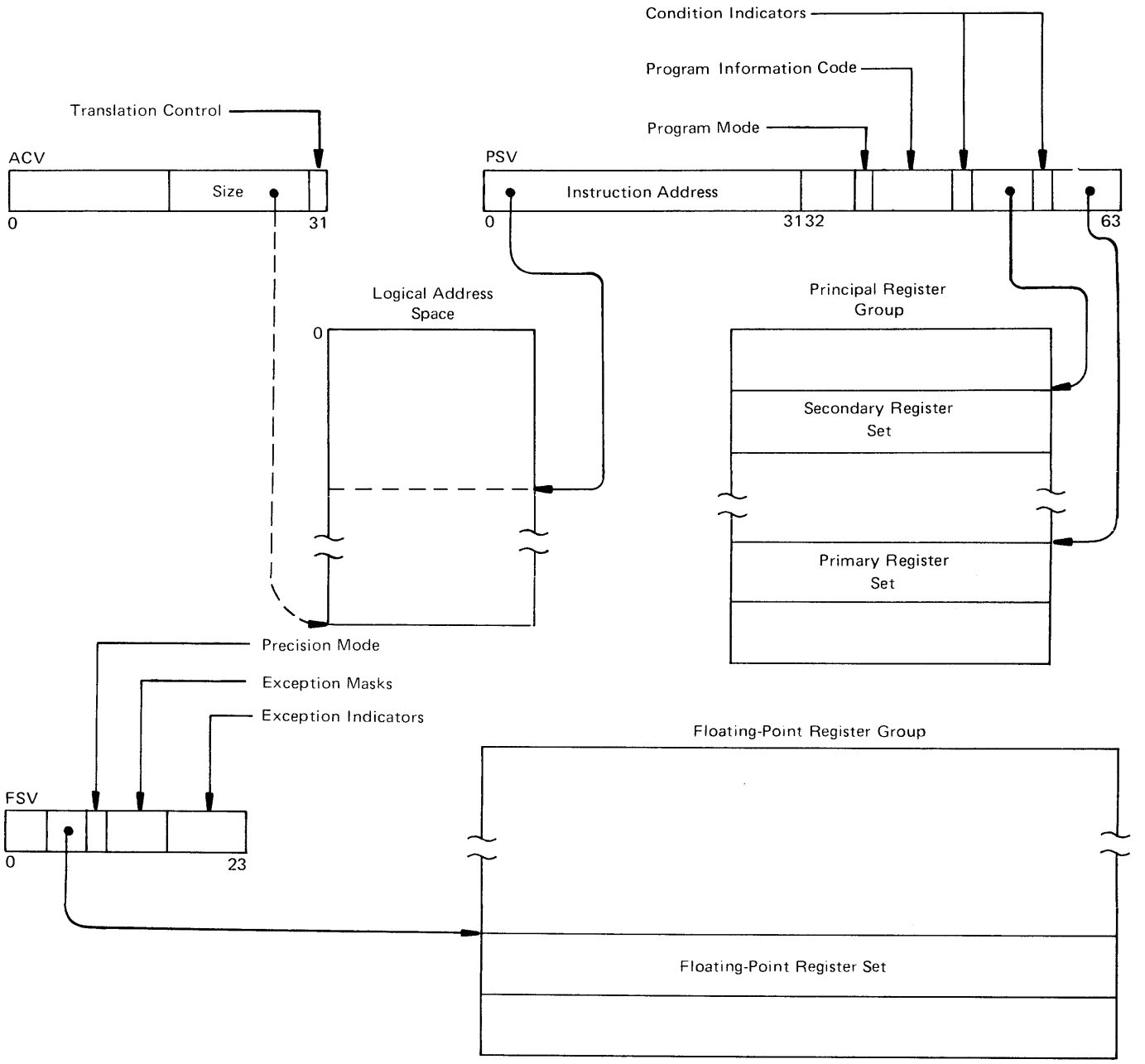


Figure 1-18. Program-Execution Environment

Dynamic Address Transformations

The dynamic address transformations performed by the IBM 8100 system PCE are provided as aids for storage management by a supervisory program. A supervisory program must deal with many functions concerning the management of main storage. Among these are:

- Allocation of storage space among programs with varying sizes and periods of activity
- Provision for sharing programs and data
- Protection of both shared and nonshared areas

Logical Addressing of Main Storage

The main-storage addressing arrangement is based on a logical separation of the addresses used by the program from the addresses assigned to the physical locations in main storage. That is, the addresses used by the program are not used directly to refer to main storage. This separation can provide extensive storage protection and ease the task of storage management. The dynamic address transformation facilities are provided to associate logical addresses used by the program with physical locations in main storage.

An *address space* is a set of addresses used to refer to main storage. Byte locations are numbered consecutively starting with 0 and continuing to the largest address defined for the address space.

Each program can have assigned to it a logically distinct address space. Similarly, a logically distinct address space can be assigned to each channel I/O operation. These address spaces are called *logical address spaces.* All main-storage addresses used by the program or channel are treated as logical addresses.

The *real address space* is the set of addresses assigned to the physical locations in main storage. The largest address in the real address space corresponds to the highest-numbered installed physical location.

Also defined is an address space which comprises the complete set of logical addresses provided by the processor. This address space is called the *PCE address space.* The PCE address space is not the real address space which corresponds to physical storage locations. The size specified for a logical address space cannot exceed the size of the PCE address space.

The information that defines a logical address space is contained in an ACV. An ACV is associated with each PSV; the ACV describes the logical address space available to the program. When a new PSV is introduced as the current PSV, a new ACV is introduced also. (See "Program-Environment Definition" and "PCE Control" in this chapter.) An ACV is also associated with an entity called a *channel pointer.* The ACV describes the logical address space available for the channel I/O operation using the logical address in the corresponding channel pointer.

During program execution and channel I/O operations, the addresses in a logical address space are always dynamically relocated in the PCE address space. When dynamic address translation is not active, the relocated addresses are used as real addresses. When translation is active, a translation table is used by the PCE to translate the relocated addresses into real addresses. The translation facility allows contiguous blocks of relocated addresses to be assigned to noncontiguous blocks of physical main storage. The address spaces and dynamic relocation and translation facilities are illustrated in Figure 1-19 and discussed in the following sections.

Dynamic Address Relocation

The basic transformation of logical addresses is dynamic address relocation. Each logical address space is assigned a beginning location, called the *origin address,* in the PCE address space. The assigned size of the logical address space determines the maximum logical address that may be used by the program or channel I/O operation. The origin and size are represented by information placed in an ACV by the supervisory program.

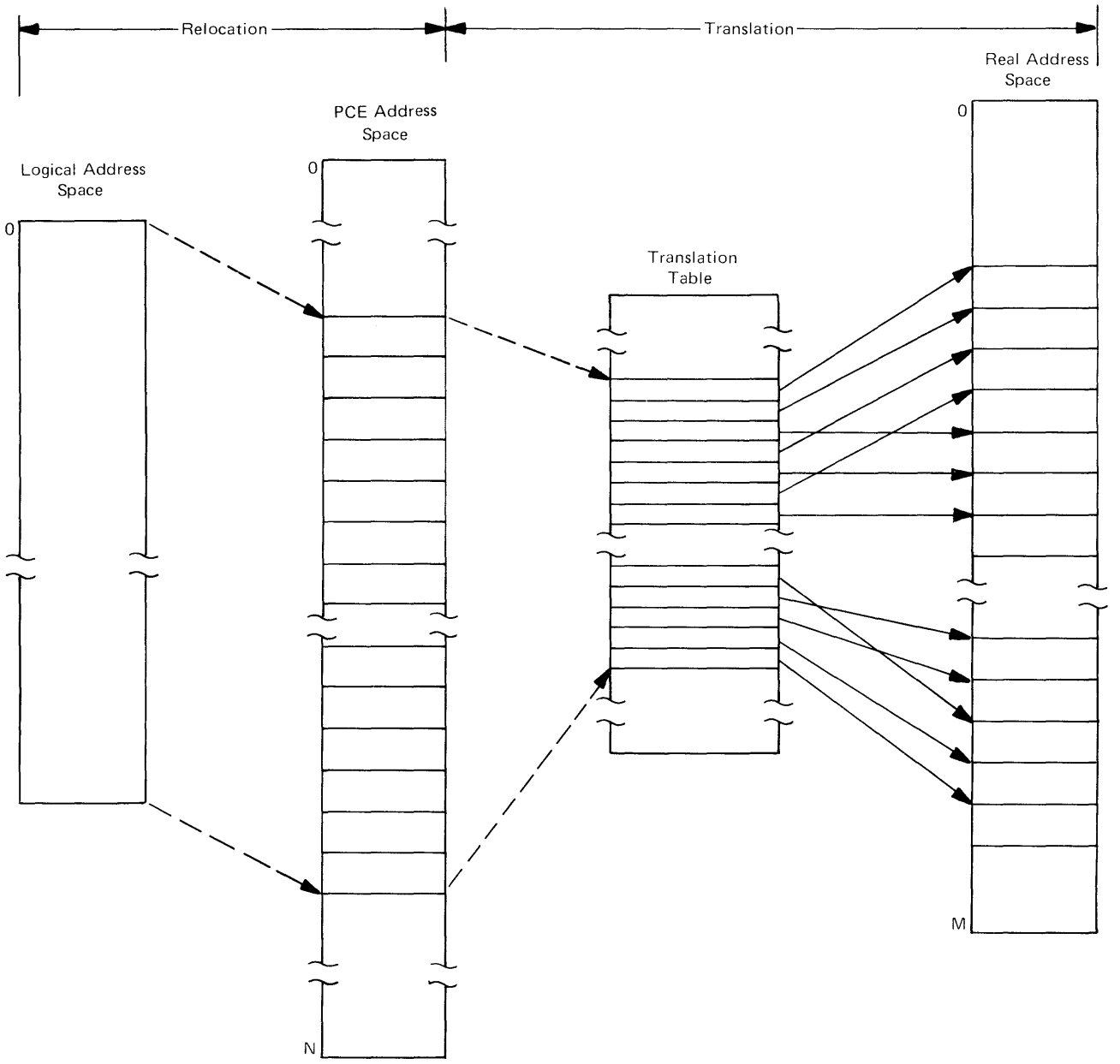


Figure 1-19. Dynamic Address Relocation and Translation

During each use of a storage address, the dynamic address relocation facility tests the logical address. If it exceeds the maximum address in the logical address space, a program exception is indicated. If the logical address does not exceed the maximum, it is combined with the origin address to produce an address in the PCE address space. This process maps the contiguous addresses in a logical address space into a set of contiguous addresses in the PCE address space, as illustrated in Figure 1-20.

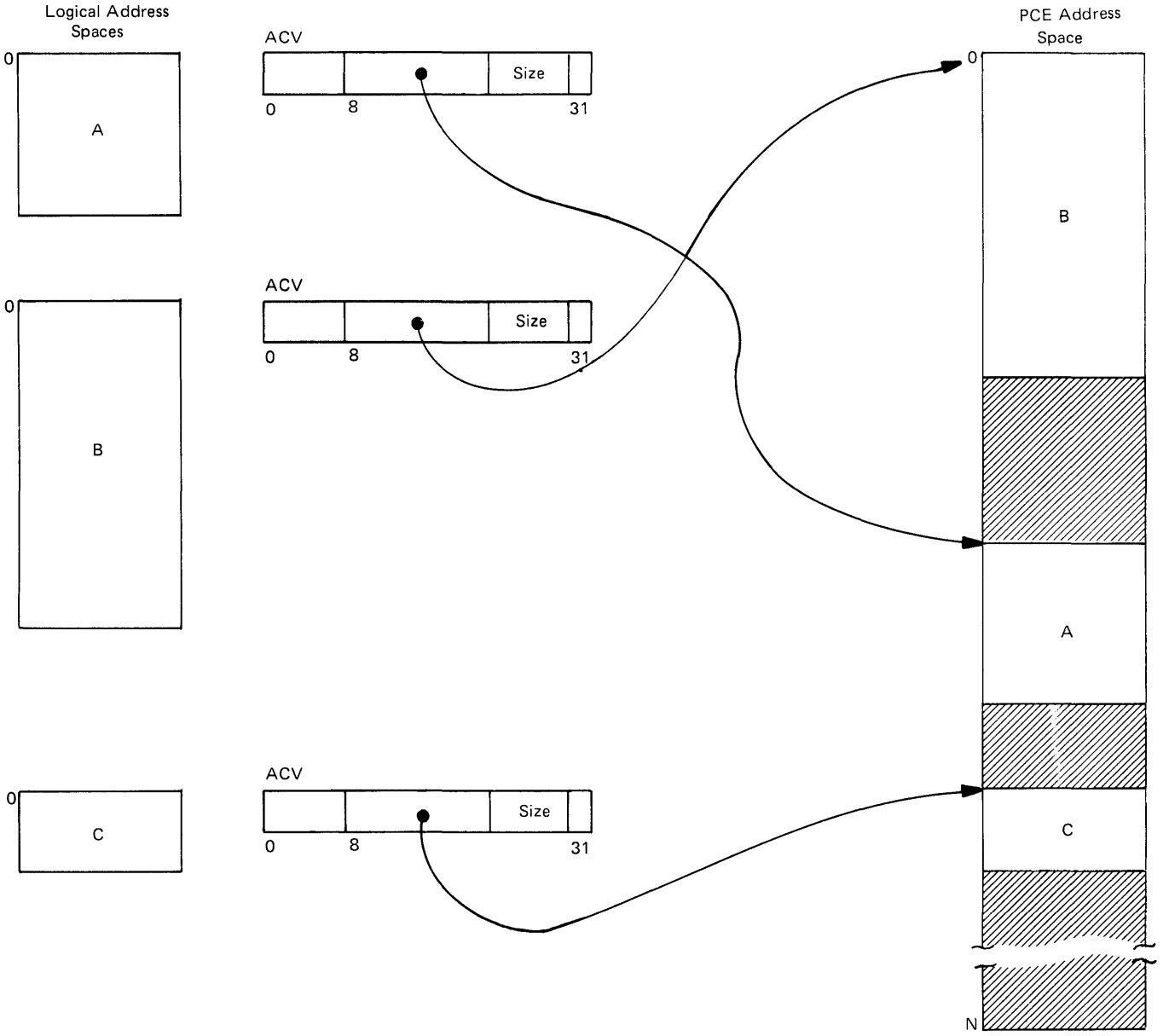


Figure 1-20. ACV Control of Dynamic Address Relocation

Because dynamic address relocation is applied on each storage reference, an address space may be logically moved to a different location in the PCE address space by changing its assigned origin. Further, the origin addresses may be chosen so that address spaces are nested for sharing of programs and data (see Figure 1-23).

The ACV is 32 bits long and contains an origin field, a size field, and a translation-control bit. The size in bytes is defined as an integral power of 2 that is not less than 2^{11} and not greater than 2^{32}. Thus, logical address space sizes may be 2,048, 4,096, 8,192, 16,384, and so on, up to the size of the PCE address space. The number of bits needed in the ACV to express the size varies depending upon the size of the logical address space. Accordingly, the number of bits needed to specify the origin is also determined by the size of the space.

Specifically, the origin address of a logical address space within the PCE address space must be an integral multiple of the size of the logical address space. For example, a logical address space of 32,768 bytes may begin at any of the addresses 0, 32768, 65536, 98304, and so on, in the PCE address space.

The binary representation of the origin address necessarily has a number of low-order 0's which is not less than the number of low-order 0's in the binary representation of the size of the logical address space. For example, the allowable origin addresses for a 32,768-byte logical address space have the following hexadecimal representations:

Hexadecimal	**(Decimal)**
00000000	(0)
00008000	(32768)
00010000	(65536)
00018000	(98304)
•	•
•	•

All of these addresses have at least 15 low-order 0's in the corresponding binary representation. Further, the number of significant bits required to represent a logical address is not greater than the number of low-order 0's in the binary representation of the size. For example, the largest address in a 32,768-byte address space is 32767 (00007FFF in hexadecimal), which has 15 low-order 1's in its binary representation.

Low-order 0's in the origin address are not included in the origin field of the ACV. Relocation, which logically consists of adding the origin address to a logical address, is accomplished simply by concatenating the origin field from the ACV with the significant bits of the logical address supplied by the program or channel. The significant bits of the logical address are those low-order 1's in the binary representation of the largest address in the logical address space. (The boundary of significant bit positions is indicated by *m* in Figure 1-21.)

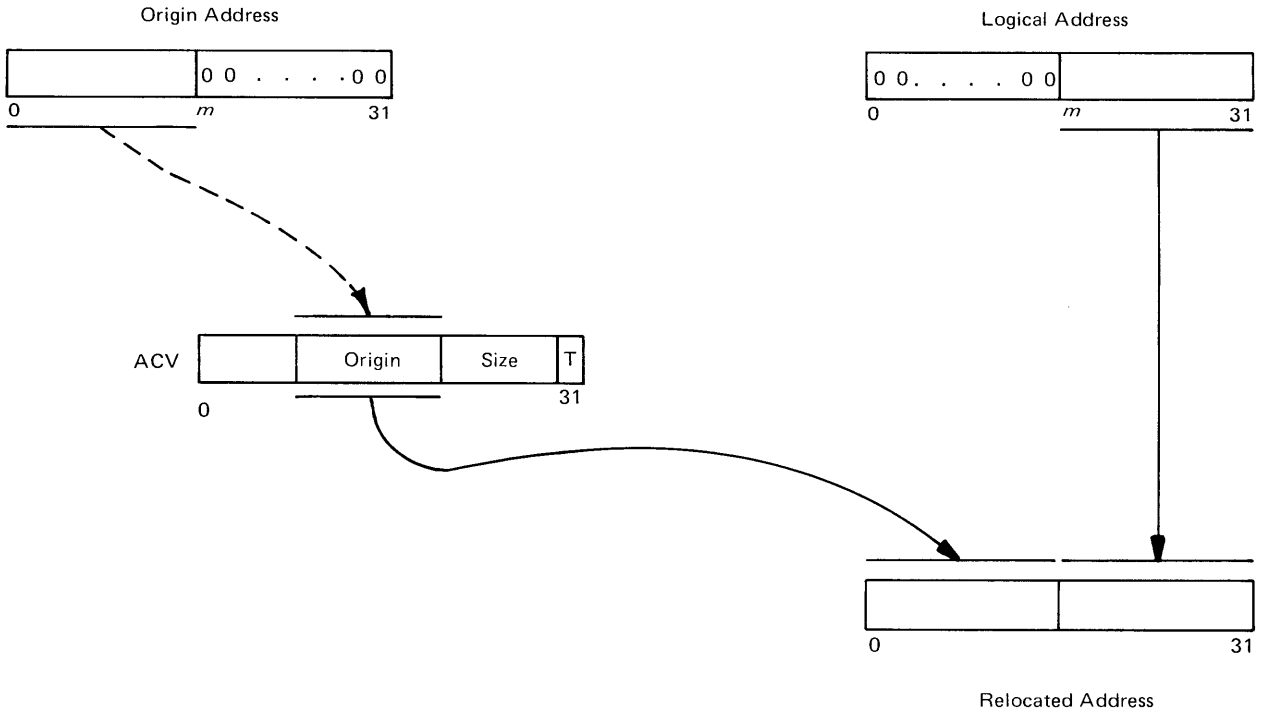


Figure 1-21. Relocation Process

The low-order bit position of the ACV (bit 31) controls the dynamic address translation facility. When address translation is not specified, the relocated address is used directly to refer to a location in physical main storage (see Figure 1-22). An attempt to refer to a location in the PCE address space corresponding to an uninstalled physical location causes a program exception to be indicated. If address translation is specified, the relocated address is transformed into a real address by the dynamic address translation facility, which is described later.

Figures 1-23 and 1-24 illustrate storage-management functions aided by dynamic address relocation. Figure 1-23 illustrates nesting of logical address spaces for sharing information.

Figure 1-24 illustrates how the logical separation of address spaces provides a means for protecting information in main storage. When the ACV for program A is the current ACV, the information in main storage assigned to program B cannot be referred to by A. Similarly, program B cannot refer to information in main storage assigned to program A.

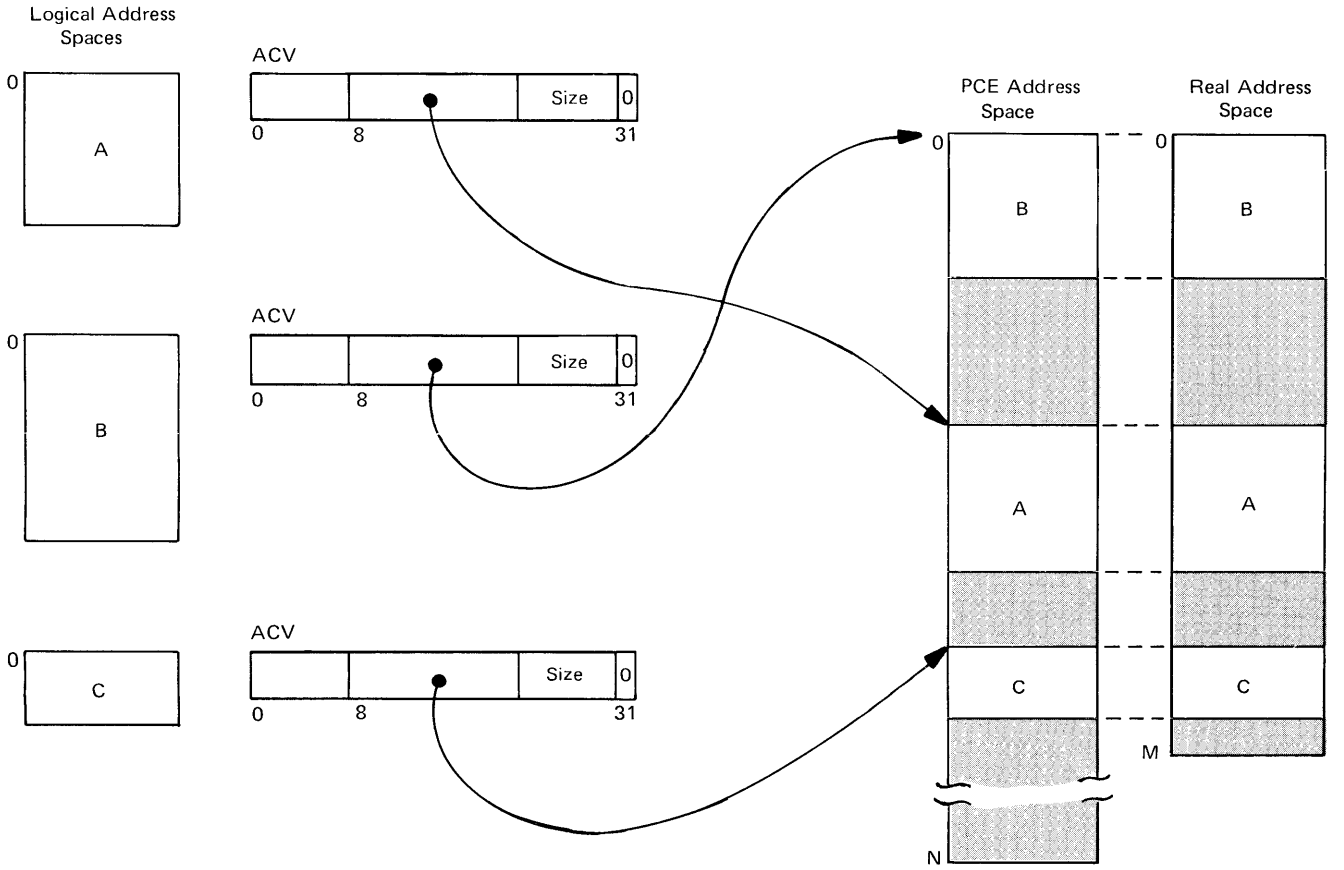


Figure 1-22. Dynamic Address Relocation (Translation Not Specified)

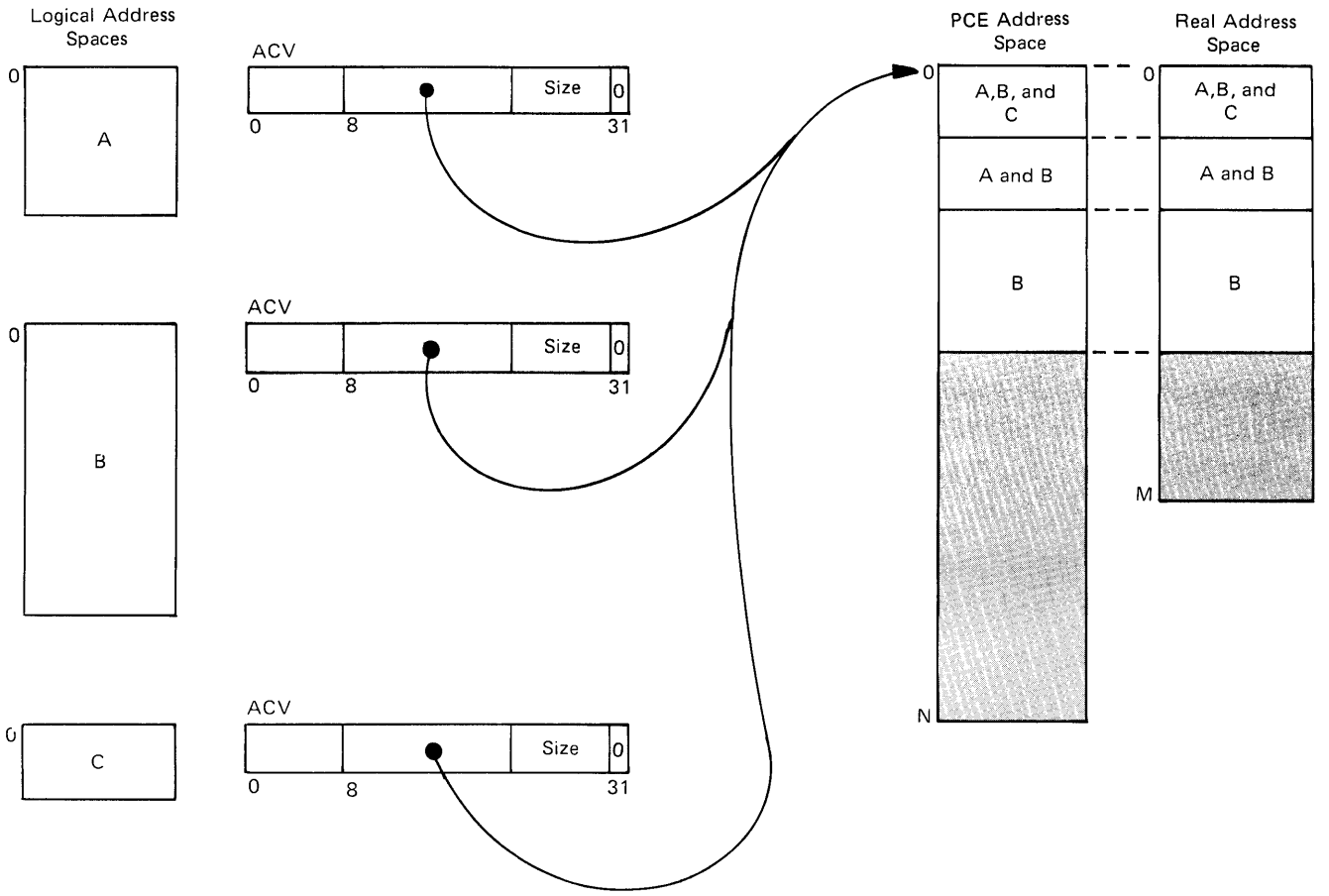


Figure 1-23. Nested Logical Address Spaces

Dynamic Address Translation

Storage management is further aided by the dynamic address translation facility. This facility allows a supervisory program to manage assignment of contiguous logical addresses to noncontiguous areas of main storage. More efficient use of installed main storage can be obtained by allocating fragments of real address space to one contiguous logical address space. Also provided with this facility are means for controlling access to storage.

For the purpose of address translation, the PCE address space and the real address space are both logically partitioned into equal-sized blocks. A *block* comprises 2,048 (2K) byte locations and begins at an address that is a multiple of 2048. When dynamic address translation is specified in the ACV, addresses in the PCE address space are not used directly to refer to locations in main storage. Instead, a translation table is provided which allows a supervisory program to associate blocks in the PCE address space with blocks of real addresses (see Figure 1-25).

One translation-table entry is provided for each block in the PCE address space. Each table entry is 32 bits long and contains a block-address field and a field for access control. The translation-table entries are organized in the same sequence as the blocks in the PCE address space; contiguous blocks in the PCE address space are associated with contiguous entries in the translation table. When dynamic address translation is active (bit position 31 of the ACV is 1), each address (after relocation) is used to locate the corresponding entry in the translation table. The block address in the table entry designates the block of main storage associated with the block of PCE address space.

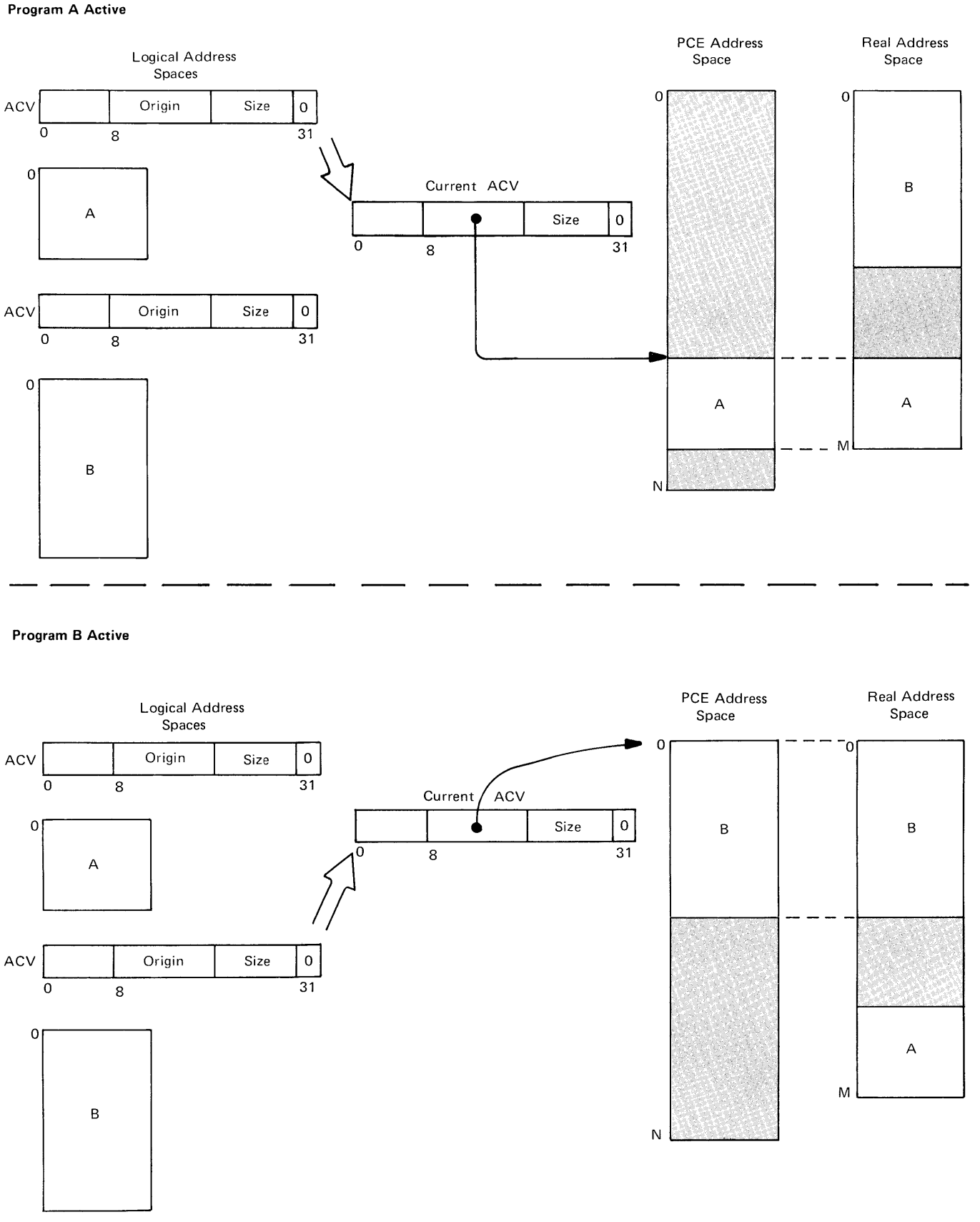


Figure 1-24. Information Protection in Main Storage

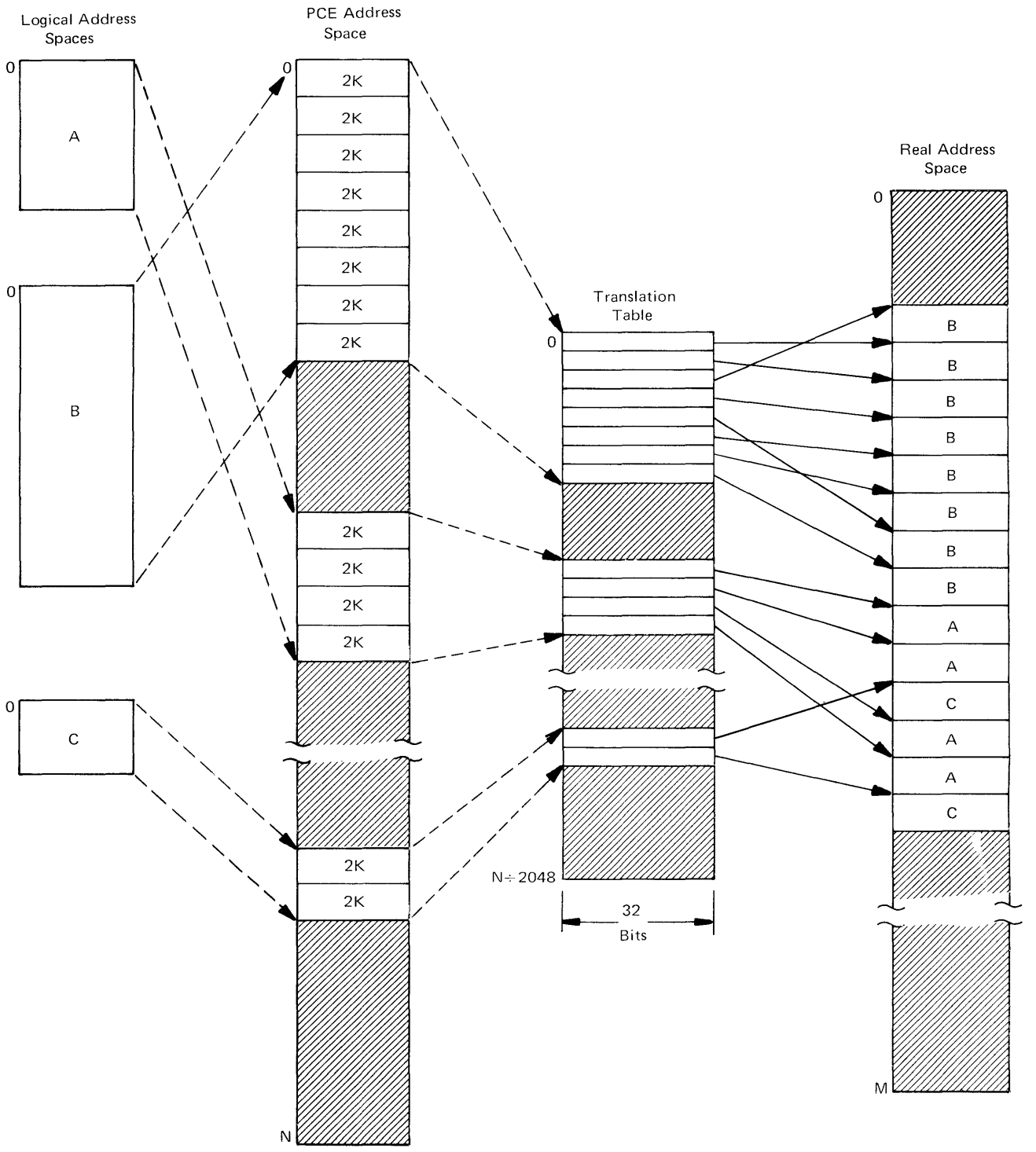


Figure 1-25. Dynamic Address Translation

The instructions LOAD FROM ADDRESS TRANSLATION TABLE and STORE TO ADDRESS TRANSLATION TABLE are provided to access or modify table entries. These instructions may be executed only by a program that has the proper authorization indicated in the program-mode field of its PSV.

During the translation process, each relocated address is treated as having two parts: a block index and a byte index. The block index portion is used as an index to select an entry from the translation table. The table entry contains the high-order bits of the corresponding real address, which are then concatenated with the byte index to form the complete real address (see Figure 1-26).

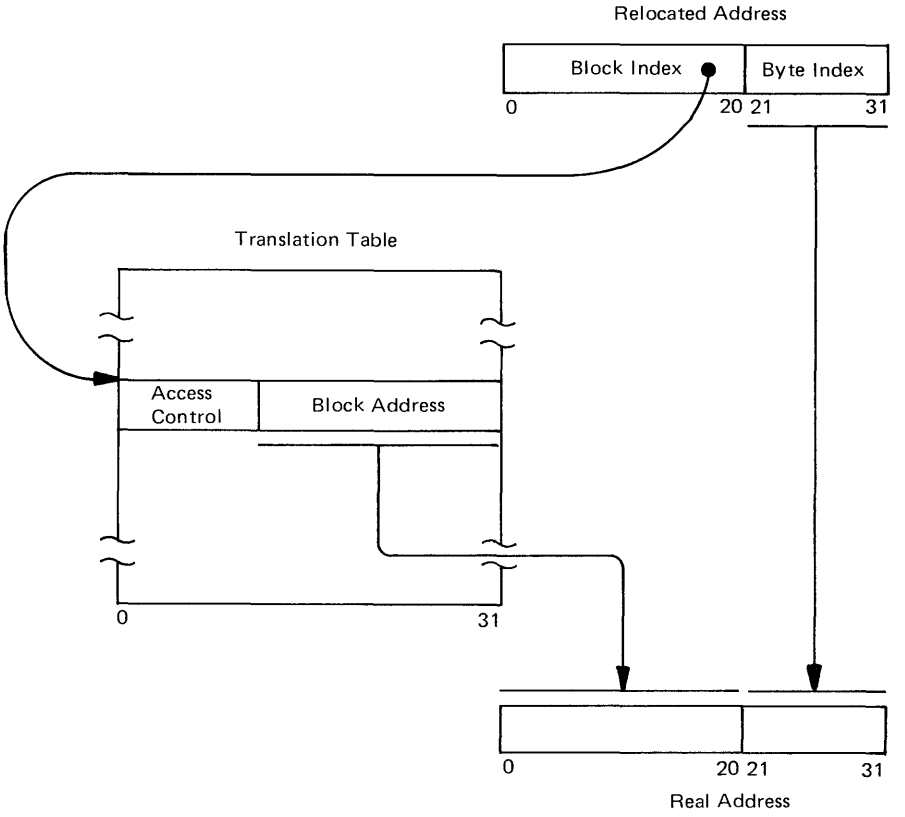


Figure 1-26. Translation Process

Figure 1-27 shows information sharing by several programs that are otherwise separate. In this figure, programs A and B share information having the same addresses in both of their logical address spaces. Programs B and C share information that has different addresses in both of their logical address spaces. Note that in the latter case, shared data should not contain location-dependent information (for example, address values).

When dynamic address translation is active, its translation facility provides protection against improper storage access by using the access-control field in the translation-table entries. Each 2048-byte block in the PCE address space may be assigned access protection. The access types allowed for the block are checked during each address translation. When main storage is accessed by a program and the access type is not allowed, a program exception occurs. The program exception code in the stored PSV is set to 1 to indicate an access exception, and depending on processor model, the exception block index (EBI) register associated with the active PSV contains the block index of the PCE address in error. When main storage is accessed by a channel I/O operation and the access

type is not allowed, a channel exception occurs. EBI registers are not used for channel exceptions. The types of access control that can be specified are: (1) no accesses permitted, (2) store access not permitted, (3) instruction-fetch access not permitted, or (4) store by channel I/O operations not permitted. Combinations can also be specified.

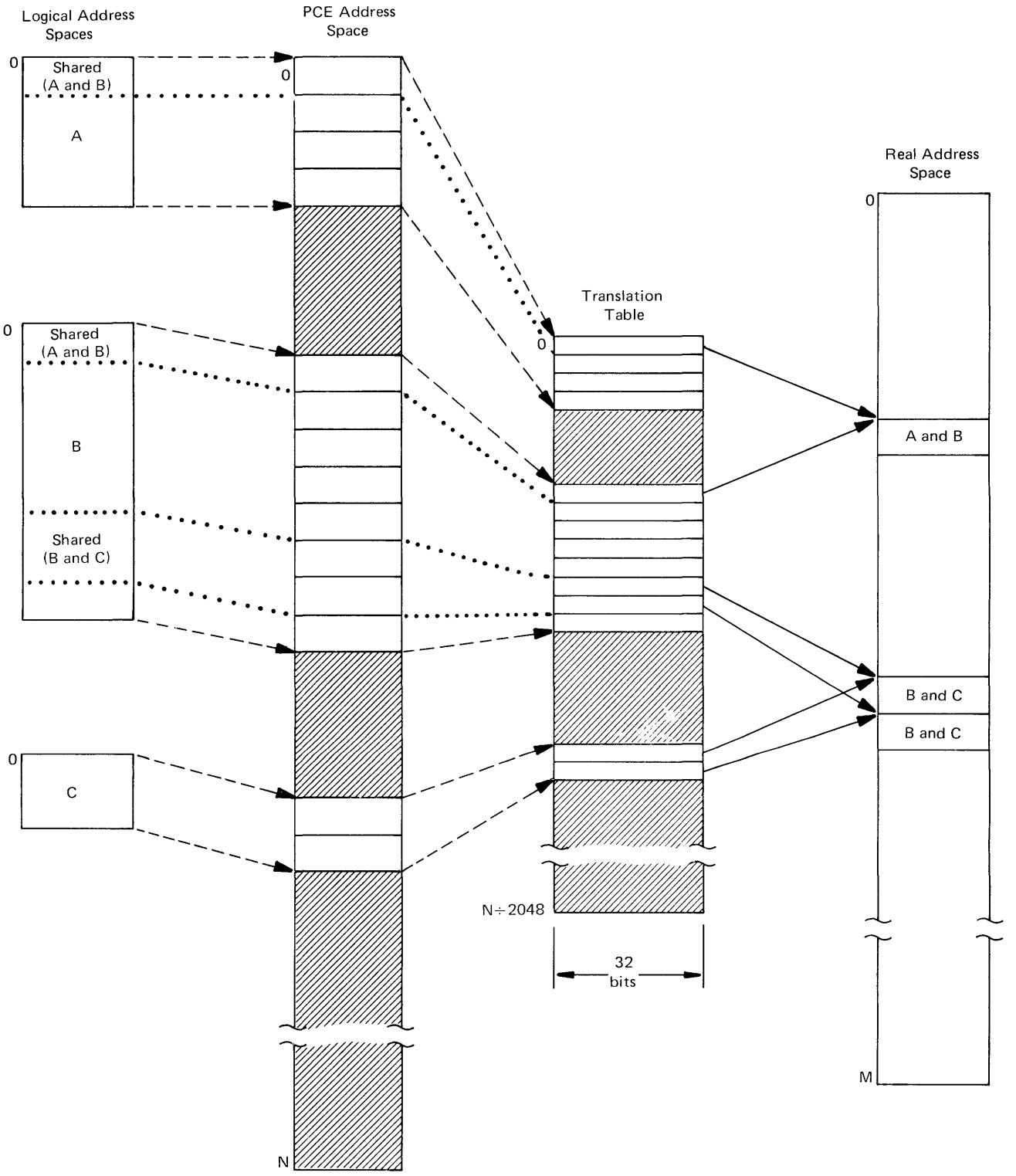


Figure 1-27. Information Sharing

Depending on processor model, the dynamic address translation facility also provides separation protection by means of translation locks and protection keys when dynamic address translation is active. Separation protection is in addition to access-control protection. With the translation lock/protection key mechanism, multiple programs and/or channel I/O operations may coexist, but are logically separated within a common logical address space defined by a particular ACV value.

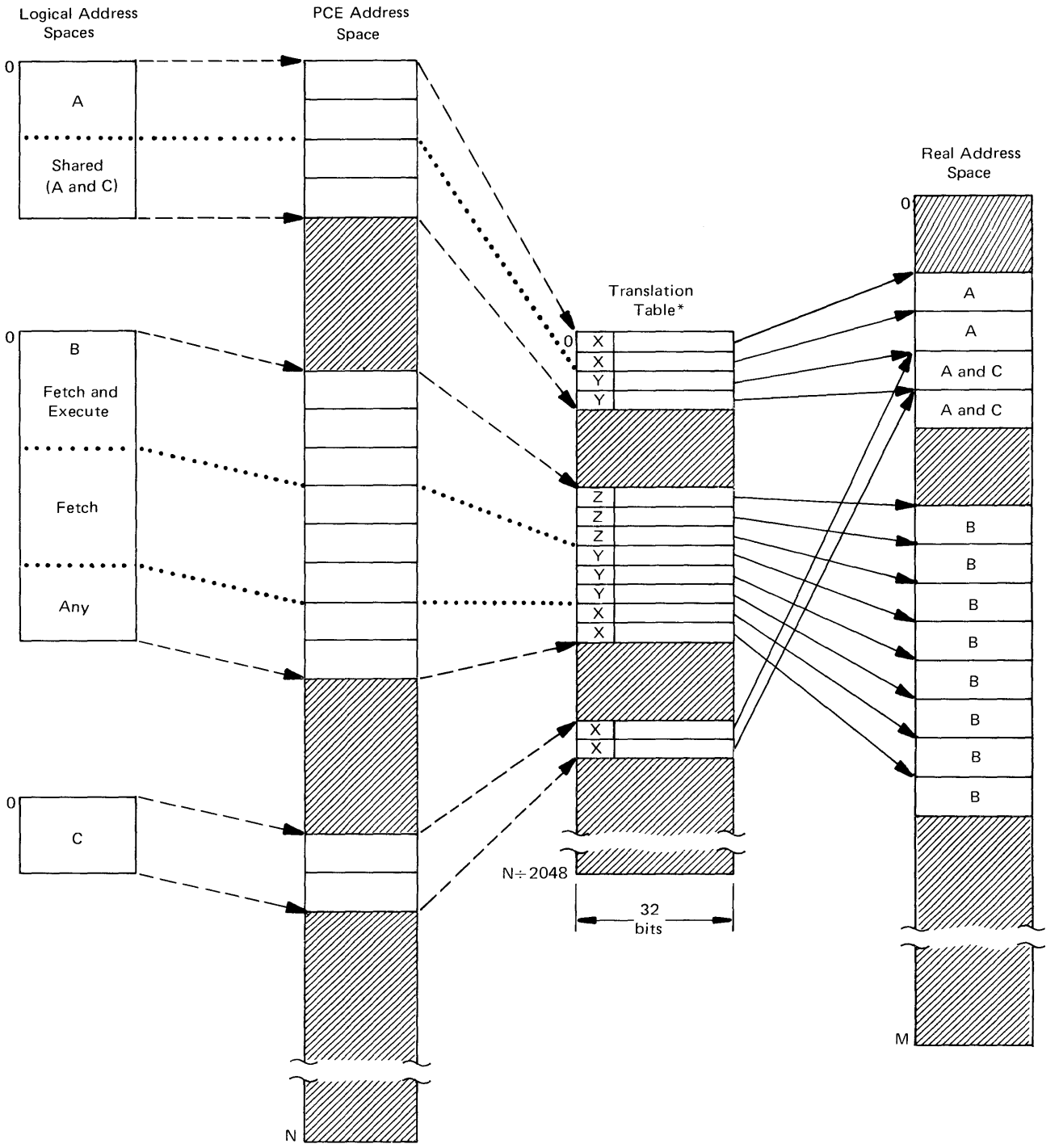
A translation lock table contains an entry for each 2K-byte block in the PCE address space; each entry contains a translation lock. Whenever a program or CHIO operation becomes active, a protection key associated with the active ACV is also activated. When a main-storage block is accessed, the corresponding translation lock is compared with the active protection key. If the translation lock and protection key values are either identical or at least one value is zero, access is allowed.

If a program attempts to access main storage and access is not allowed, a program exception occurs. The program exception code in the stored PSV is set to 3 to indicate a separation exception, and the block index of the PCE address in error is placed in the exception block index (EBI) register associated with the active ACV.

If a CHIO operation attempts to access main storage and access is not allowed, a channel exception occurs. EBI registers are not used for channel exceptions.

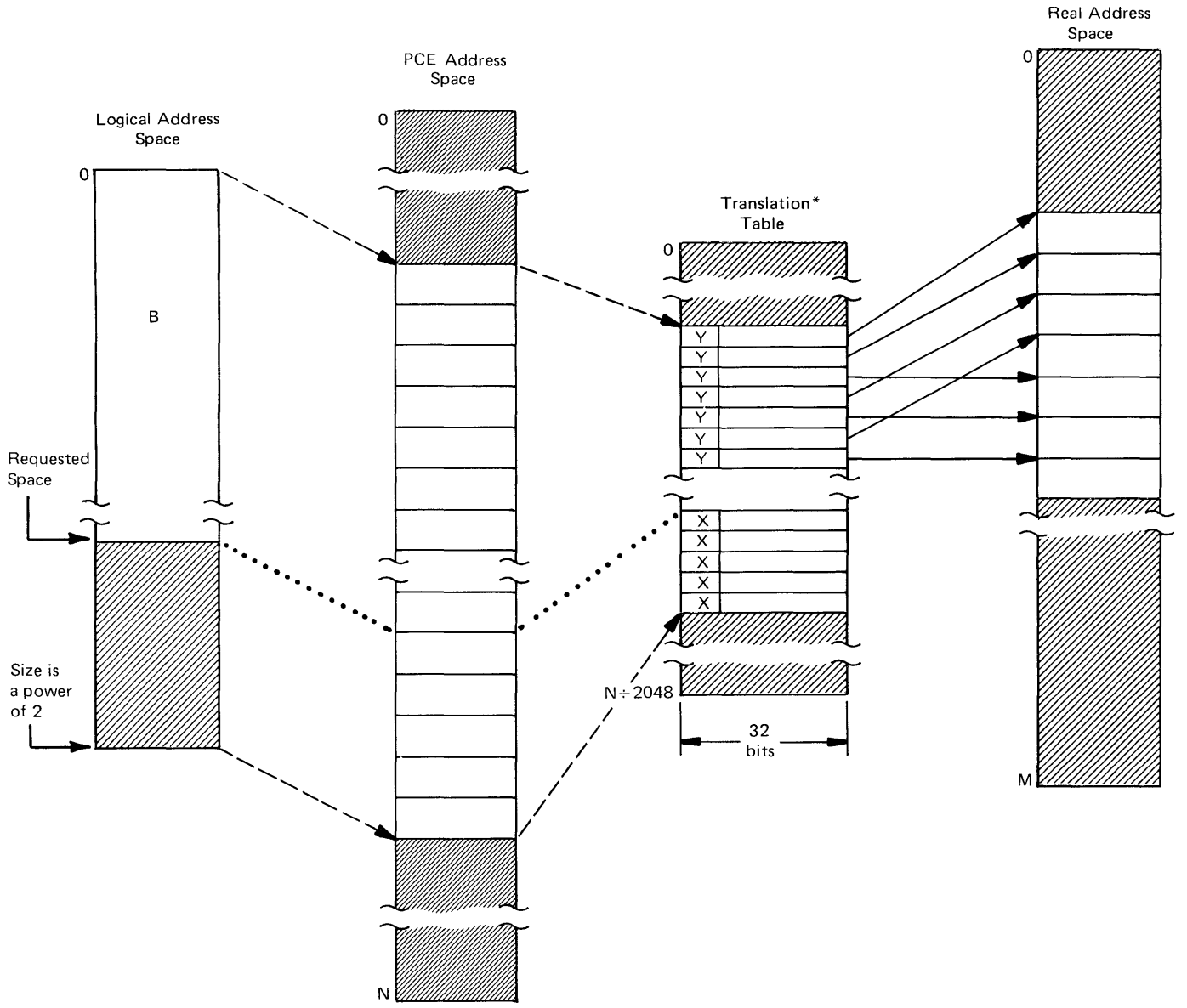
The LOAD FROM ADDRESS TRANSLATION LOCK TABLE and STORE TO ADDRESS TRANSLATION LOCK TABLE instructions are provided to access or modify translation-lock-table entries. These instructions may be executed only by a program that has the proper authorization indicated in the program-mode field of its PSV.

Figures 1-28 and 1-29 show some examples of access control with dynamic address translation. Figure 1-28 illustrates how programs may be allowed different types of access to shared information. Partitioning of a program's instruction areas from data areas within an address space is also illustrated in this figure. An example of conservation of physical main storage is shown in Figure 1-29. In this example, program B requires a smaller amount of storage than the next largest logical-address-space size (which is an integral power of 2). Only those main-storage blocks required by B are allocated; program B is not allowed to use logical addresses for which no main storage is assigned. Figure 1-30 shows an example of how separation protection is used to control storage access.



*Legend: X = all access types valid
Y = fetch access valid
Z = fetch and execute access types valid

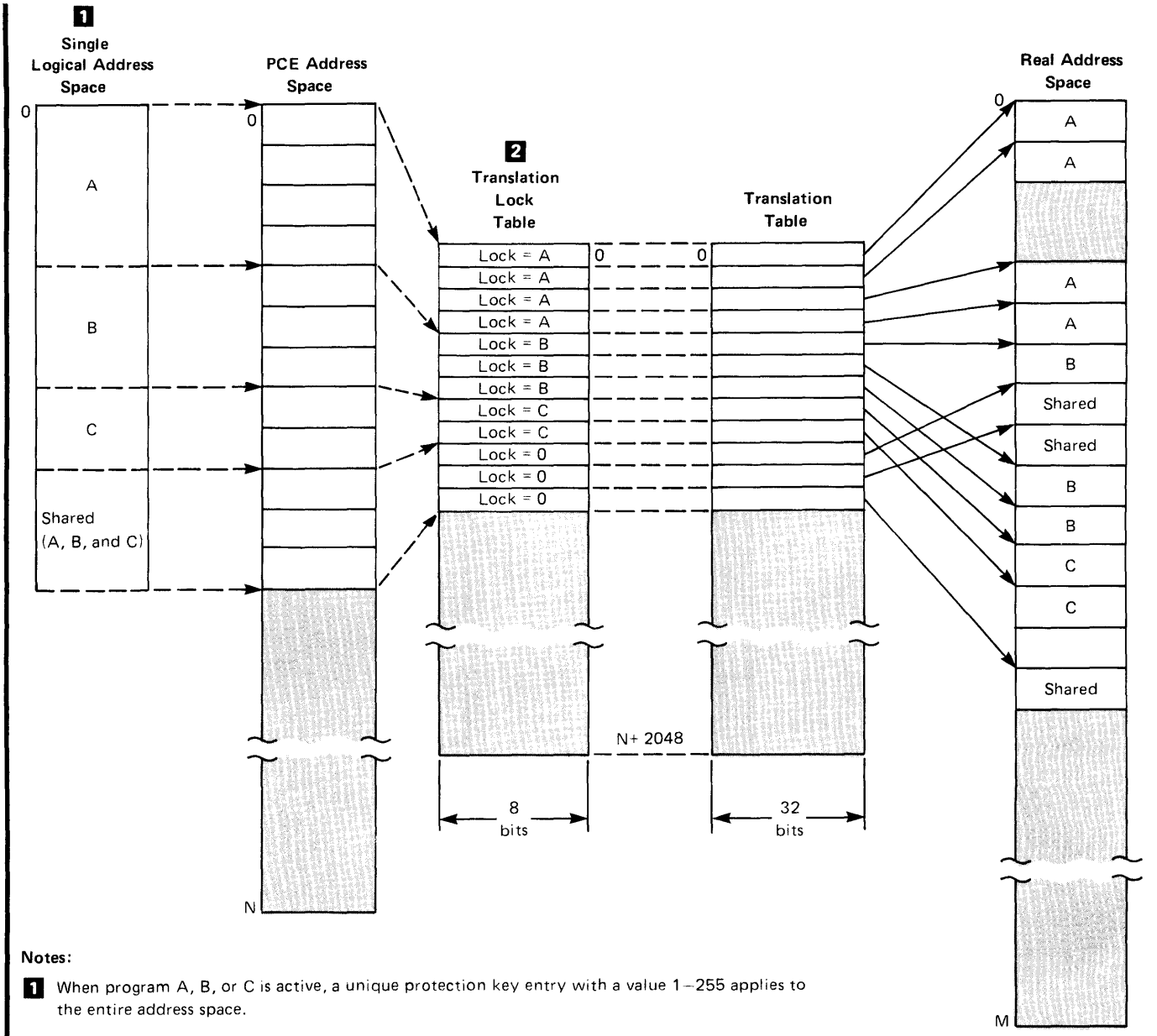
Figure 1-28. Information Sharing with Different Types of Access



*Legend: X = All access types are invalid; real storage is not allocated.

Y = All access types are valid; real storage is allocated.

Figure 1-29. Example of Conservation of Main Storage



Notes:

1 When program A, B, or C is active, a unique protection key entry with a value 1–255 applies to the entire address space.

2 Translation lock values A, B, and C are unique values 1–255. A value of 0 allows access by any protection key.

Figure 1-30. Example of Separation Protection

Register Organization

The PCE provides three groups of registers: principal registers, adjunct registers, and (optionally) floating-point registers. All registers are organized in register sets. Each principal or adjunct register set consists of eight 32-bit registers numbered 0-7. The organization of principal and adjunct registers is illustrated in Figure 1-31. Each floating-point register set consists of four 64-bit registers numbered 0-3.

Principal Registers

The group of principal registers consists of 64 sets numbered 0-63. Principal register sets 0, 1, 4, 5, and 8-15 are permanently assigned to hold system control information; sets 16-63 are available for assignment by a supervisory program to programs for use as general registers.

Principal register sets 0, 1, 4, and 5 are used to hold the PSV information. Sets 8-15 are assigned as channel pointers and are used during channel I/O operations to address storage (see "Input/Output Operations" in this chapter). Sets 2, 3, 6, and 7 are reserved and should not be used.

Adjunct Registers

The group of adjunct registers consists of 64 sets numbered 0-63. Certain adjunct register sets are permanently assigned as follows: sets 0, 1, 4, and 5 contain ACVs and EBIs, sets 8-15 contain ACVs, and sets 16, 17, 20, 21, and 24-31 contain protection keys. Because ACVs are logically associated with PSVs and channel pointers, adjunct register sets 0, 1, 4, 5, and 8–15 are logically associated with the correspondingly numbered principal register sets. The remaining adjunct register sets are reserved and should not be used.

Floating-Point Registers

The group of floating-point registers consists of eight sets numbered 0-7. All floating-point register sets are available for assignment to programs.

Access to Register Groups

The contents of any register in both the principal and adjunct register groups may be referred to using register-indirect operations. Information may be transferred in byte or halfword units between an active general register and any principal register or any adjunct register, including those permanently assigned. These instructions can be used only by a program that has the proper authorization indicated in the program-mode field of its PSV.

PCE Control

The PCE gives control to programs in response to requests for program execution. The functions performed by the PCE to determine which program is to be given control are called *dispatching functions.* The PCE performs dispatching functions automatically to provide fast response to requests for program execution. Requests come from three sources:

- Requests created by a program
- Signals from I/O devices
- Requests generated by the PCE as a result of detecting certain errors

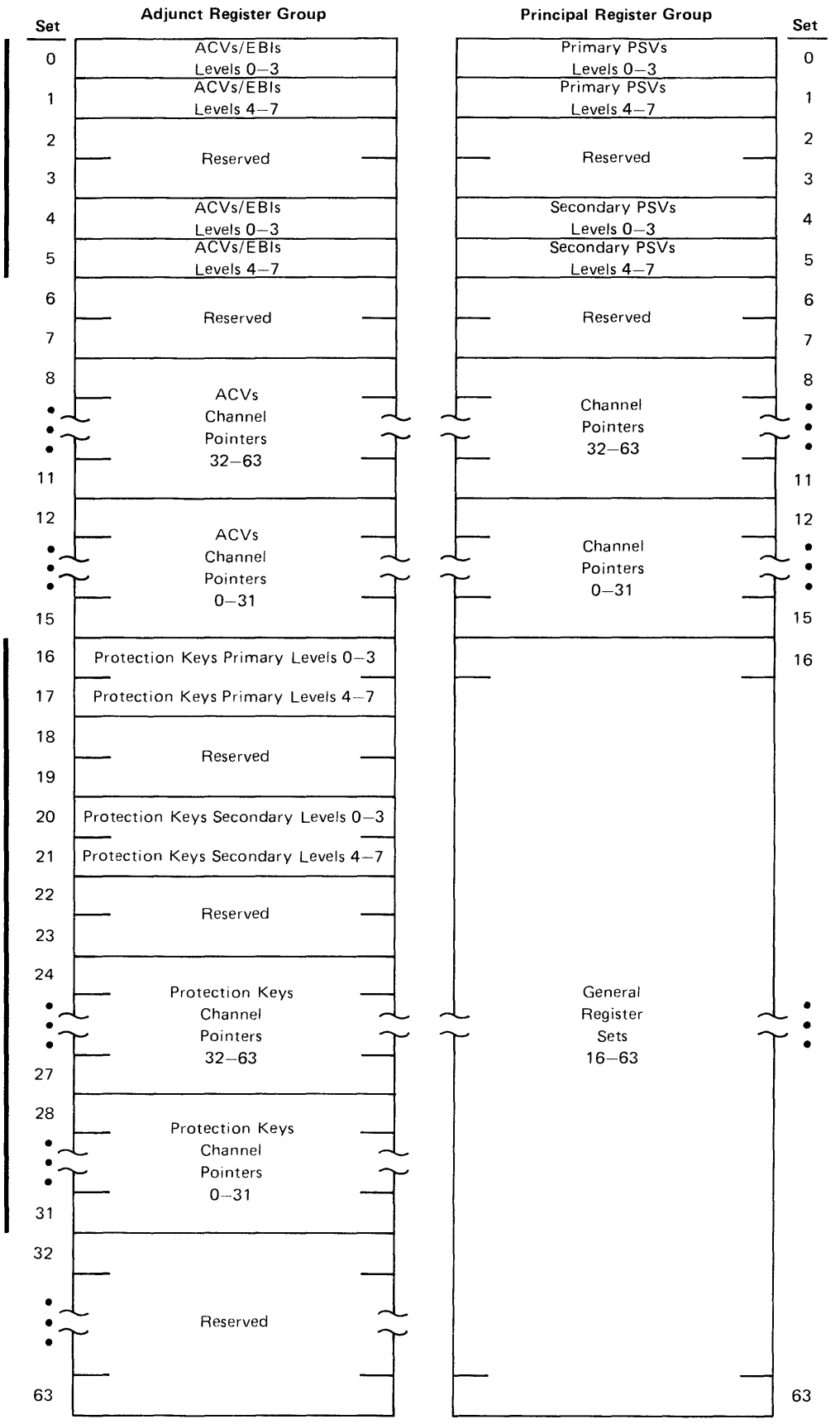


Figure 1-31. Principal and Adjunct Register Assignments

The allocation of unique general-register sets from the group of principal register sets, and the definition of a logical address space, allow a supervisory program to establish a distinct execution environment for a program. Because a distinct environment is defined, the state of the program is automatically preserved when program execution is suspended and restored when the program is subsequently resumed.

Priority Levels

Requests for program execution are serviced by the PCE according to their relative priorities. Eight levels of priority are defined. The eight priority levels are numbered 0-7. Level 0 is defined as the highest priority, level 1 is defined as the next highest priority, and so on to level 7, which is defined as the lowest priority. The priority level at which the PCE is executing instructions is called the *current priority level.*

The priority levels can be enabled or disabled for requests for program execution. When a priority level is enabled, programs associated with that priority level can be given control in response to a request. When a priority level is disabled, all requests for programs at that level remain pending until the level is enabled.

Whether priority levels are enabled or disabled is indicated and controlled by mask bits in the *master mask* and the *common mask.* The two masks provide a hierarchy of control. The 1-bit master mask controls priority levels 1-7 as a group. The 8-bit common mask provides individual control for each of priority levels 0-7. The master mask takes precedence over the common mask.

A third mask, the 1-bit channel mask, determines whether channel input/output operations are enabled or disabled.

A program is given an execution priority by means of its PSV/ACV pair. Associated with each priority level are principal and adjunct register locations that are permanently assigned to hold PSV/ACV information. The register locations that hold a PSV and its paired ACV are uniquely associated with one priority level. Thus, the priority level at which a program executes is determined by the register locations in which its PSV/ACV pair is held.

When the optional floating-point feature is installed, one FSV is assigned to each priority level. The FSV provides information for the proper execution at that priority level of a program which uses the floating-point instructions.

Register locations for two PSV/ACV pairs are associated with each priority level; one is designated the *primary PSV/ACV pair* and the other is the *secondary PSV/ACV pair.* This dual PSV/ACV facility allows a supervisory program and an application program to execute at the same priority level. The primary PSV/ACV pair should normally be used for a supervisory program, and the secondary PSV/ACV pair can be used for any program.

PSV information is held in an even/odd numbered pair of consecutive registers in a principal register set. Corresponding to the register locations of a PSV is an even/odd numbered pair of register locations in an adjunct register set. This adjunct register pair is permanently assigned to hold the corresponding ACV and EBI. The PSV and ACV/EBI register locations and their association with the priority levels are illustrated in Figure 1-32. Information in the PSV and ACV/EBI register locations may be inspected or modified using register-indirect operations (see "Register Organization" in this chapter).

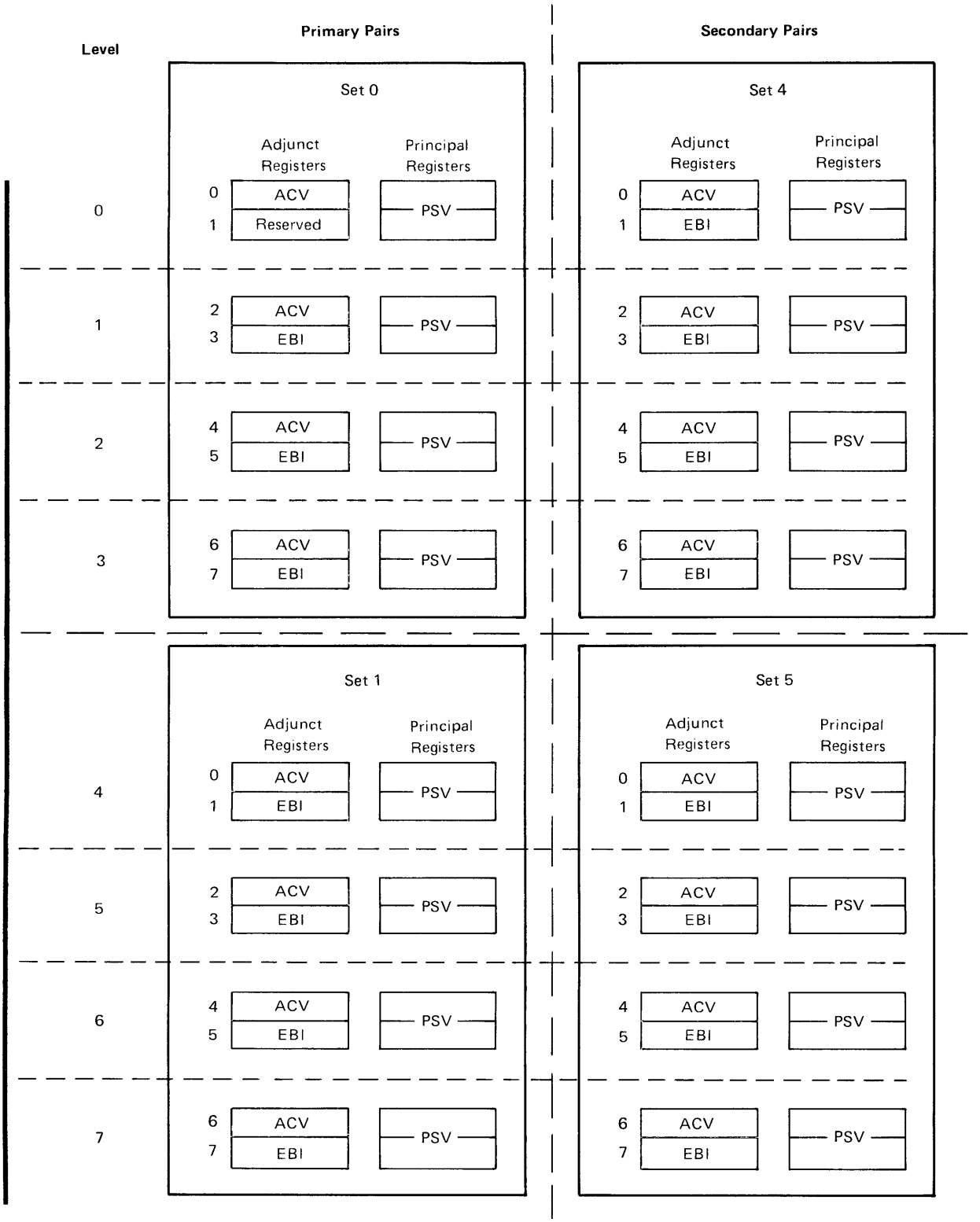


Figure 1-32. PSV and ACV/EBI Register Locations

Interrupt Requests

The PCE's dispatching mechanism operates in response to requests generated by a program, I/O devices, or PCE-detected errors. These requests for program execution can result in an interruption of the current program and the introduction of a new PSV/ACV pair. For this reason, the request for program execution is called an *interrupt request.* An interrupt request is always associated with a priority level and represents a request for execution of a program at a specific priority level.

Interrupt requests from I/O devices are held in an I/O interrupt request vector (IOIRV). Requests created by the currently executing program are held in a programmed interrupt request vector (PIRV). In both the IOIRV and the PIRV, one bit position is defined for each of the eight priority levels. A request for program execution at a specific priority level is indicated by the bit associated with that level. For the purpose of indicating interrupt requests, an I/O device can be assigned to a priority level.

System checks result in a request for program execution at priority level 0 and are indicated in an error interrupt request vector (EIRV). The group of system checks includes machine checks, I/O checks, channel exceptions, and those program exceptions detected while a primary PSV is active (usually indicating an error in a supervisory program).

The three types of interrupt requests are generated independently of each other and may be present at any time in any combination. Two or more priority levels can, and often do, have simultaneous interrupt requests present. Depending upon the configuration of the system, a given priority level may be associated with only a single interrupt-request source, or the priority level may be shared for the processing of requests from more than one source. When a priority level is shared, the applicable sources must be examined to determine which one generated the interrupt request.

Priority Level Dispatching

The procedure performed by the PCE to select a priority level for program execution is called *priority level dispatching.* Conceptually, the procedure to select a priority level for program execution is repeated after every unit of operation. The entire execution of a noninterruptible instruction is a single *unit of operation.* For interruptible instructions, a unit of operation may consist of only partial execution of the instruction. Interrupted instructions are normally resumed automatically with the next unit of operation when the interrupted program is next given control.

The PCE determines which priority levels are eligible for selection by combining all requests and excluding those associated with disabled priority levels. Normally an interrupt request for a given level must be present in order for program execution to take place at that level. A PSV/ACV pair associated with the highest enabled priority level having an interrupt request present is given control.

If the selected level is the same as the current level, program execution continues on the current level with the next unit of operation. In two cases, however, a new program is given control at the current priority level. The first of these is execution of the instruction CALL PSV, which introduces the opposite (dual) PSV/ACV pair for the same priority level (switches from the primary PSV/ACV pair to the secondary pair or from the secondary pair to the primary pair). The

second case occurs when a program exception is encountered while a secondary PSV is active. In this case, the primary PSV/ACV pair for the current priority level is introduced. In both cases, the current level does not change even though a new PSV/ACV pair is introduced. An 8-bit control vector, the program activation vector (PAV), is updated to indicate which PSV/ACV pair is to be introduced when control is returned to the current level following an interruption.

If the selected level and current level are different, program execution at the current level is interrupted and the program at the selected level is given control. A new priority level is selected when (1) there is a request for program execution at a level higher in priority than the current level, and the higher priority level is enabled, or (2) the request for program execution at the current priority level is removed or the current level is disabled, and another request is present that is associated with an enabled priority level.

The PCE normally continues program execution at an enabled priority level only while an interrupt request is present for that level. The master mask can also be used to continue program execution at the current level even when no interrupt request is present for that level. When the request sustaining program execution on the current priority level is removed, program execution on that level is considered to be concluded at the completion of the current instruction.

The wait state is entered at the completion of the current instruction when program execution on the current priority level is concluded and no other interrupt request is present for an enabled priority level. When the wait state is ended because of an interrupt request, priority level dispatching is resumed as if the interrupt request were present at the completion of the last instruction.

The numbers of the current priority level (CPL) and the last priority level (LPL) are automatically maintained by the PCE and can be read by the program. The CPL number indicates the currently active priority level. The LPL number designates the priority level that was active before the current PSV/ACV pair was introduced.

Interruption Action

An interruption is defined as the action performed by the PCE when control is taken from one PSV/ACV pair and given to another PSV/ACV pair. The program associated with the PSV/ACV pair from which control is taken is called the interrupted program. Interruptions occur when the PCE's dispatching mechanism determines that a new PSV/ACV pair is to be introduced (a new program given control), whether at the current priority level or at a higher level. The interruption action is performed automatically by the PCE.

The interruption action includes storing the current PSV, updating certain control information, and introducing a new PSV and ACV. Information from the current PSV is stored in the register locations from which the PSV was loaded. The current ACV is not stored because its contents cannot be changed during program execution. When the floating-point feature is installed, the interruption action also includes making a new FSV active when a new priority level is dispatched. Processing resumes as specified by the new PSV/ACV pair. This interruption action is performed automatically by the PCE; no action by the program is necessary to store PSV information or introduce a new PSV/ACV pair.

The stored PSV holds all necessary PCE-status information relative to the program being executed at the time of the interruption. When program execution

is interrupted because of requests not associated with errors, the stored PSV contains the address of the instruction to be executed next. This permits automatic resumption of the interrupted program.

When an interruption occurs as a result of a program exception or system check, information is stored in the PSV that permits identification of the instruction being executed when the interruption occurred. When appropriate, this information can also be used for resumption of the interrupted program. The cause is identified by additional information made available to the program. The specific information and its location depends on the interruption type.

Instructions for PCE Control

A class of instructions, the PCE-control instructions, is provided to read and modify active PCE-control information, as well as to perform other operations necessary for PCE control. Instructions that can modify active PCE-control information can be used only by a program that has the proper authorization indicated in the program-mode field of its PSV. Operations that read active PCE-control information are valid in all program modes.

Input/Output Operations

The transfer of information between an I/O device and main storage, or between an I/O device and a register in the PCE, is referred to as an input/output operation. Two methods may be used to transfer data to or from an I/O device. They are called programmed I/O (PIO) and channel I/O (CHIO).

- PIO refers to the transfer of a single unit of data between the I/O device and the PCE. Specifically, the transfer occurs between the I/O device and a general register designated in an I/O instruction. PIO is used to selectively reset I/O devices, and to read and modify device-status information. It is also used when distinct I/O device operations are to be directly controlled by the program. Three I/O instructions are provided. Two of the instructions transfer a single byte of data to or from the I/O device; the third instruction transfers a single halfword. I/O instructions can be executed only by a program that has the proper authorization indicated in the program-mode field of its PSV.

- CHIO refers to the transfer of multiple units of data between the I/O device and main storage. CHIO is used principally for transferring information at a high data-rate. After the program initiates a CHIO operation, the program is free to perform other work; that is, the channel transfers data asynchronously with respect to program execution. The internal facilities of the PCE may be shared by the channel for controlling CHIO operations. This sharing is accomplished automatically and the program is not affected except for an increase in execution time.

Programmed I/O Operations

Each of the three I/O instructions specifies an 8-bit device address, an 8-bit command, and a general-register operand location from or into which data is transferred. Execution of an I/O instruction consists of the logical selection (connection) of the addressed device, the transfer of the command to the device, and the transfer of one unit of data to or from the device. Execution of the I/O instruction is completed after the data unit is transferred. The condition

indicators in the current PSV are set to reflect the outcome of the PIO operation. When the channel detects an error during execution of an I/O instruction, a system-check interruption is generated. PIO is illustrated in Figure 1-33.

INPUT/OUTPUT (byte) and INPUT/OUTPUT (byte, immediate) instructions designate a byte operand and are normally used with devices that transfer a single byte of data during each operation (these instructions may also be used with devices that transfer halfwords). INPUT/OUTPUT (halfword) specifies a halfword data operand and is provided for use only with devices that transfer a single halfword of data during each operation.

The device address is specified with the contents of a general-register byte operand designated by the I/O instruction; it provides for 256 unique addresses. The PIO command code specifies to the I/O device the operation to be performed. The low-order bit of the command code identifies the direction of data transfer. Except for four commands discussed in the following paragraphs, the significance of the high-order seven bit positions of the command code depends on the type of I/O device.

Each I/O device provides a basic status register (BSTAT). The information contained in the BSTAT identifies certain device status. Basic-status information is made available to the program by means of the PIO command Read BSTAT. Ordinarily, the handling of I/O interruptions by the program includes the reading of the BSTAT. Additional status information that may be provided by the device is made available by device-specific PIO commands.

The BSTAT is also used by the device as a control register. Certain bits are used to control functions such as the generation of I/O interrupt requests or the disabling of operations at the device. The PIO commands Set BSTAT Under Mask and Reset BSTAT Under Mask allow the program to modify the BSTAT and, thus, control these functions. The command Reset Device is used to selectively reset I/O devices.

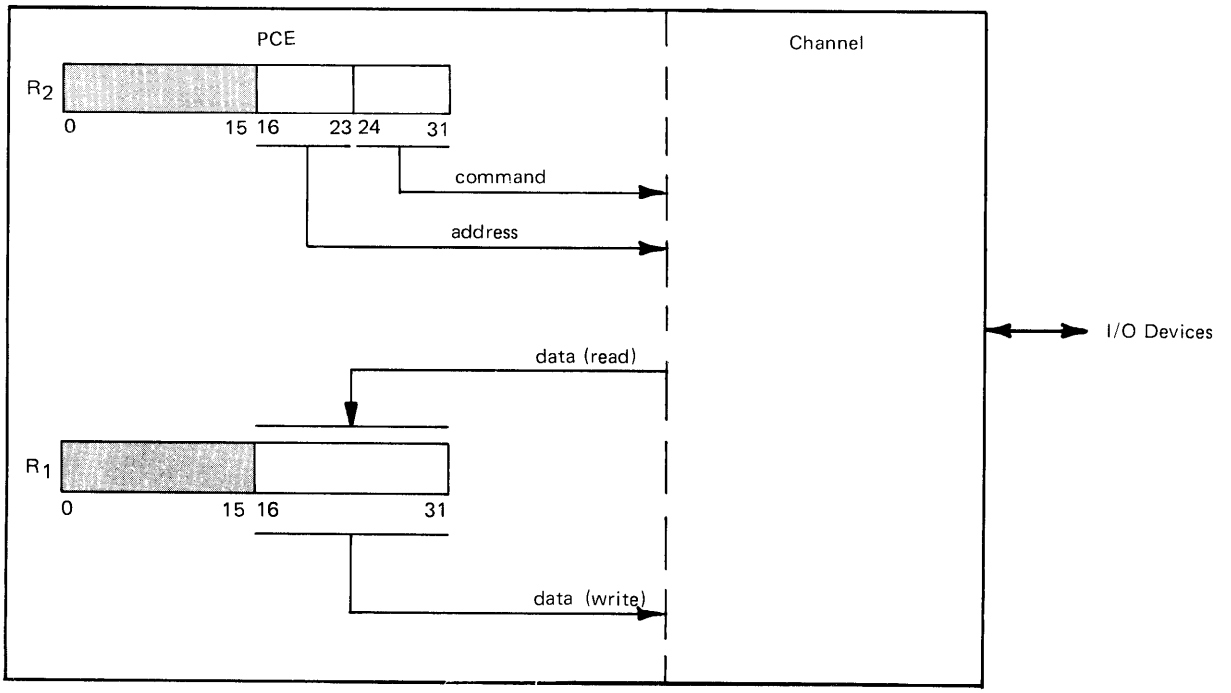


Figure 1-33. PIO Operation (halfword)

Channel I/O Operations

Channel I/O operations are used to transfer multiple units of data between main storage and an I/O device. The channel synchronizes the transfer of data to or from main storage.

The facilities in the channel may be shared by a number of concurrently operating I/O devices. Each concurrently operating device obtains a time interval during which one or more units of data are transferred. The length of the interval depends on the operating characteristics of the device, such as the number of bytes in a burst and the device transfer rate. During such an interval, only one device is logically connected to the channel.

The program initiates a CHIO operation with an I/O instruction. The PIO command code designated by the I/O instruction is interpreted by the device as the initiation of a CHIO operation. The command code that initiates a CHIO operation depends on the specific device.

Three elements of information are normally required to control a CHIO operation: the command, the beginning logical address in main storage, and the data count. Control information may be supplied to the device by means of one or more I/O instructions executed before initiating the CHIO operation. Alternatively, control information may be written to the device as data during a CHIO operation.

Operations such as reading or writing data and reading or writing address information can be specified by the CHIO command. The beginning main-storage address is obtained by specifying the number of a channel pointer that contains the logical address.

A channel pointer is 32 bits long and is used by the channel during CHIO operations to address main storage. The IBM 8100 system provides 64 channel pointers for CHIO operations. Each channel pointer is permanently assigned to one of the principal registers from sets 8-15. During a CHIO operation, the channel maintains and updates the logical address in the channel pointer.

The initial address in the channel pointer designates the location in main storage from or into which the channel transfers the first byte of data. The address can be supplied by a supervisory program or read from the I/O device during a CHIO operation. Storage locations are used in ascending order of addresses. As information is transferred to or from main storage, the address in the channel pointer is incremented by the channel. When the operation is concluded, the channel has increased the address in the channel pointer by an amount equal to the number of bytes transferred to or from main storage.

The channel pointer is associated with an ACV. All main storage addresses used by the channel are treated as logical addresses. The ACV defines the logical address space used by the channel during data transfer to or from main storage. During each storage reference, the logical address is relocated by means of the dynamic address relocation facility. When address translation is designated in the ACV, the relocated address is then translated to a real address by means of the dynamic address translation facility. Otherwise, the relocated address is used as the real address (see "Dynamic Address Transformations" in this chapter). Storage-addressing information for channel I/O operations is illustrated in Figure 1-34.

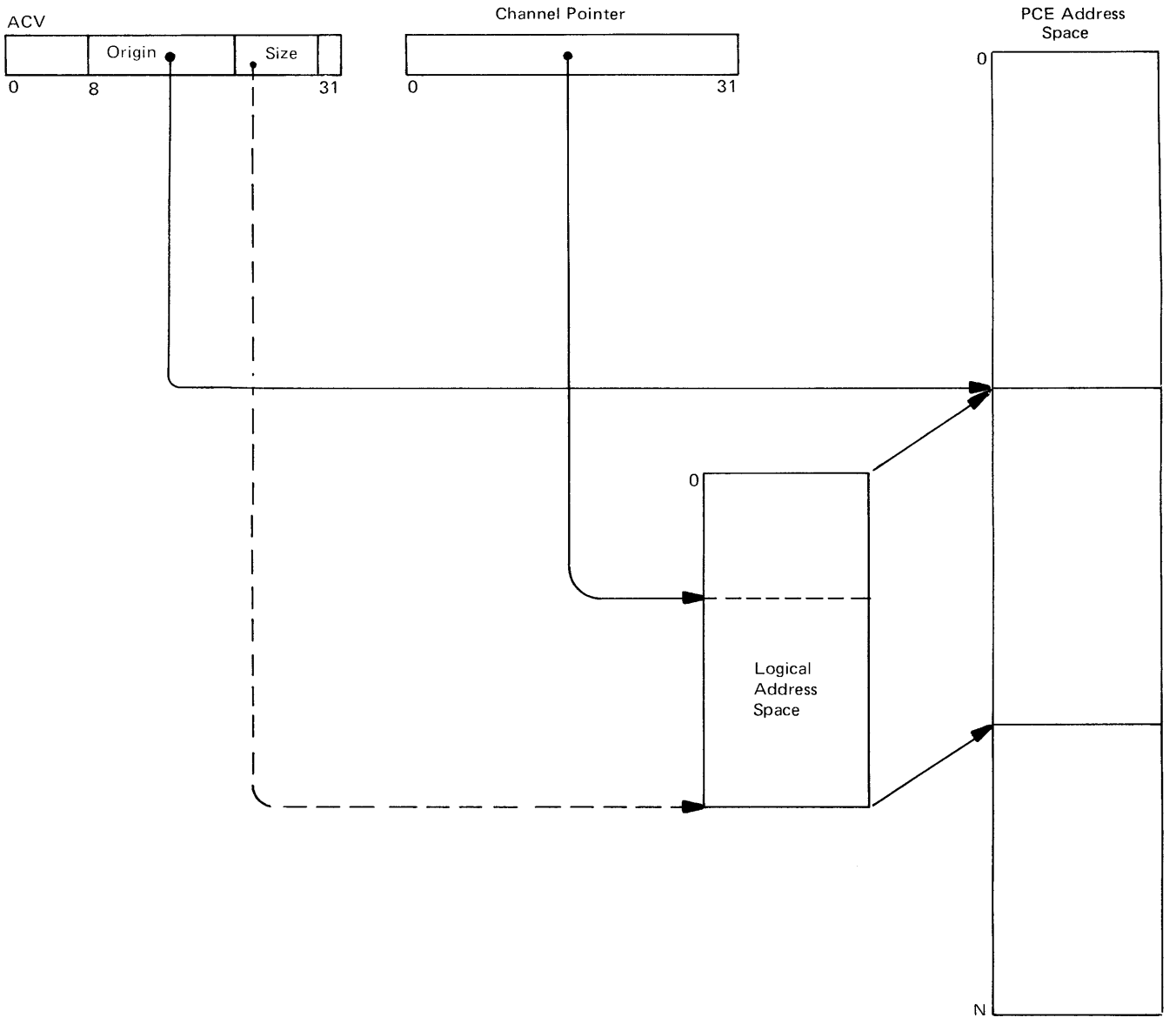


Figure 1-34. Channel I/O Storage-Addressing Information

The dynamic address translation facility also provides storage access protection for CHIO operation by means of the access-control field contained in the translation-table entries. Depending on processor model, it also provides separation protection by means of protection keys and translation locks.

During a CHIO operation, the device maintains a count of the data units transferred. Data recorded by an I/O device may be divided into physical blocks. The length of a block depends on the device. When a physical block length is defined, data-count information need not be supplied by the program to the device before the initiation of an I/O operation. One or more blocks may be transferred in one CHIO operation. The capability to transfer multiple blocks in one CHIO operation, and the manner in which it is accomplished, depends on the particular device.

For some devices or operations, blocks are not defined and the amount of information transferred is specified by the program. The data count, in this case,

is provided to the device as control information before the data transfer. Functions peculiar to each device, such as rewinding a magnetic tape or positioning the access mechanism on a disk drive, are specified by means of device-specific protocols. That is, the format and meaning of control information specifying such functions, and the method used to supply the device with the information, depend on the particular device and operation. Device-dependent control information may appear in a PIO command code, or may be transferred to the device as data during a CHIO or PIO operation.

Normally, a CHIO operation lasts until the final unit of information is transferred to or from the device. However, when the channel recognizes an exception or detects equipment malfunctioning, it terminates the data transfer immediately, logically disconnects the device, and generates a system-check interruption. When a channel I/O operation is ended, the conclusion may be signaled by the device with an I/O interruption. For devices that do not generate an I/O interruption, the conclusion may be determined by programmed interrogation.

At the conclusion of an operation, the device generates status information that indicates conditions pertaining to the execution of the CHIO operation. This status information is stored in the device's basic status register. Additional status information, if any, is stored in device-specific status registers. Status information may be obtained by the program by execution of one or more PIO commands that read the status information.

PART II. INFORMATION PROCESSING FACILITIES

Chapter 2. Storage and Registers

This chapter describes the information units used in the 8100 system. It also describes the logical organization of main storage and general registers as observed by an executing program. The assignment of registers and storage for use by a program is normally controlled by a supervisory program. This control is discussed separately in Chapter 6, "Register Organization," and Chapter 7, "Dynamic Address Relocation and Translation."

Information Units

The IBM 8100 Information System transmits or operates on information in units of 8 bits, or a multiple of 8 bits, at a time. Each 8-bit unit of information is called a *byte* — the basic building block of all units.

The bits in a byte are numbered consecutively, left to right, 0 through 7. Within any fixed-length information unit of multiple bytes, the bits making up the unit are consecutively numbered from left to right, starting with the number 0. Leftmost bits are sometimes referred to as the "high-order" bits, and rightmost bits are referred to as the "low-order" bits.

For purposes of error detection, one or more check bits may be transmitted with each byte or with a group of bytes. Check bits are generated automatically by the system and cannot be directly controlled by the program. References in this manual to the size of information units and registers exclude mention of any associated check bits. All storage capacities are expressed in number of bytes provided, without regard to storage width.

Bytes may be handled separately or grouped together in *fields.* A *halfword* is a group of 2 consecutive bytes and is the basic building block of instructions. A *word* is a group of 4 consecutive bytes; a *doubleword* is a group of 8 consecutive bytes. The location of any field or group of bytes is identified by the address of its leftmost byte.

The length of a field is either implied by the operation to be performed or stated explicitly as a parameter of the operation. When the length is implied, the information is said to have a fixed length, which can be either 1, 2, 4, or 8 bytes. When the length of a field is stated explicitly, the information is said to have variable field length. Variable-length fields are variable by increments of 1 byte or 1 halfword.

When information is placed in main storage, the contents of only those byte locations included in the designated field are replaced, even though the physical path may be wider than the field being stored. When information is fetched from main storage, the contents of those byte locations that are not included in the designated field are ignored, even though the physical path may be wider than the field being fetched.

Note: *Information units used as operands during instruction execution, or transferred during I/O operations, are usually referred to as data units.*

Main Storage

Main storage provides the system with directly addressable fast-access storage of data. Both data and programs must be loaded into main storage (usually from input devices) before they can be processed. Main storage is shared by both PCEs in dual-PCE processors.

Addressing

Byte locations in storage are consecutively numbered, left to right, starting with 0; each number is considered the address of the corresponding byte. A group of bytes in storage is addressed by the leftmost byte of the group. The number of bytes in the group is either implied or explicitly stated for the operation. The addressing arrangement uses a 32-bit binary address to provide an addressing capability of 4,294,967,296 byte addresses. This addressing *capability* should not be confused with the maximum amount of physical main storage installed.

Storage addresses are linear from address 0 to the maximum byte address. Storage addressing does not wrap from the maximum byte address to address 0. If the maximum-address boundary is crossed when main storage is referred to, a program exception is indicated.

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

Real addresses are the lowest level of program-recognizable addresses, and in this publication they are considered to be the addresses of physical storage locations. A physical storage location is not associated with more than one real address.

An address used by the program or in a channel I/O operation is referred to as a *logical address.* A logical address is always transformed into a relocated address by the dynamic-address-relocation facility before main storage is accessed. When dynamic address translation is invoked, the relocated address is then translated to a real address before main storage is accessed. When dynamic address translation is not invoked, the relocated address is used as the real address. These address transformations are described in Chapter 7, "Dynamic Address Relocation and Translation."

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

Integral Boundaries

Certain units of information must be located in main storage on an *integral boundary* (see Figure 2-1). A boundary is called integral for a unit of information when its storage address is a multiple of the length of the unit in bytes. For example, a halfword (2 bytes) is on an integral boundary when it is located in storage so that its address is a multiple of the number 2, and a word (4 bytes) is on an integral boundary when it has an address that is a multiple of the number 4.

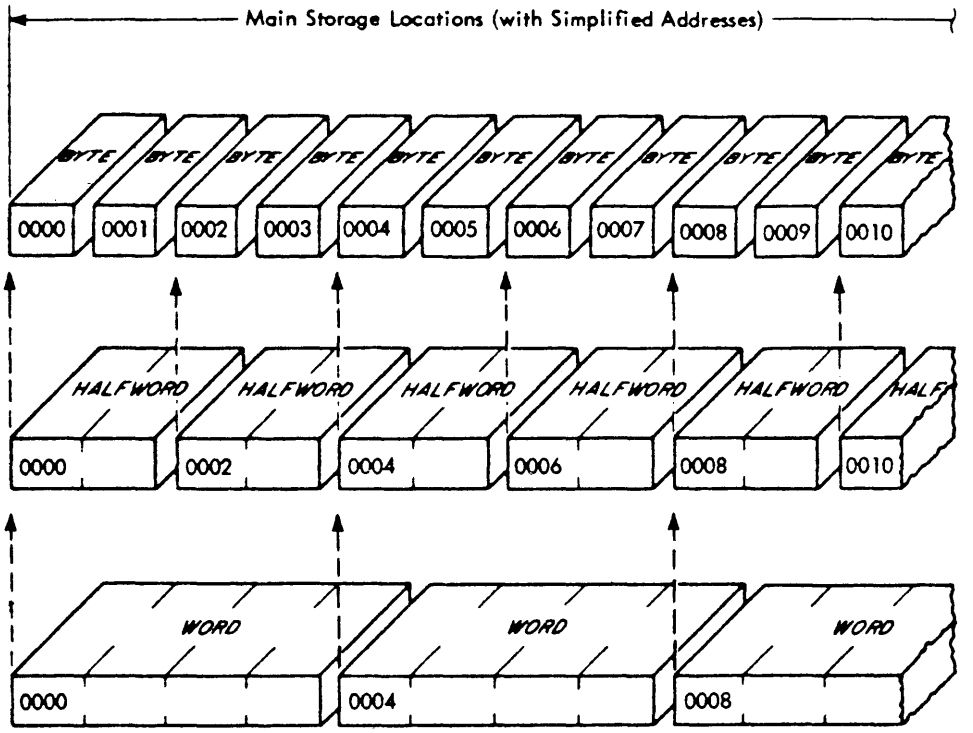


Figure 2-1. Integral Boundaries for Halfwords and Words

Instructions must appear in storage on halfword integral boundaries. Halfword and word storage operands must also appear on integral boundaries. Both short and long floating-point storage operands must appear on word integral boundaries. The binary representation of storage addresses designating halfwords and words on integral boundaries contains 1 or 2 low-order 0 bits, respectively.

Addresses used to designate information that must appear on an integral boundary are not checked for the correct number of low-order 0 bits. That is, the 1 or 2 low-order bits of addresses that designate instructions or operands that are required to be located on integral boundaries are ignored and assumed to be 0.

Programming Note: An address designating a unit of information that must appear on an integral boundary is assumed to contain the appropriate (1 or 2) low-order 0 bits. A word storage operand at location 2000, for example, can be referred to with any of the four addresses 2000, 2001, 2002, or 2003. However, the program should use addresses that correspond to the integral-boundary locations for halfword and word units of information.

General Registers

The PCE can address information stored in general registers. Each general register contains 32 bits. These registers may be used as base registers in main storage addressing and as accumulators in arithmetic and logical operations. A program may directly refer to 16 general registers.

General registers are organized in *register sets.* Each set consists of eight general registers numbered 0-7. Forty-eight register sets are provided for assignment to programs. A program has two general register sets assigned, a *primary register set* and a *secondary register set,* for a total of 16 general registers. Two fields in the program status vector designate the numbers of the primary and secondary sets

assigned to the program. When a program is executing, the general register sets designated in its program status vector contain the *active general registers* available to the program.

Note: *This manual normally refers to the active general registers simply as the general registers. Explicit references to active general registers are made only when it is necessary to distinguish them from any of the register sets that may be assigned as general registers. For a complete description of all register sets and the privileged register-indirect instructions used to access a register in any register set, refer to Chapter 6.*

Operands in general registers may be 1-byte, 2-byte (halfword) or 4-byte (word) data units. For operations using word operands, all 32 bits of the register participate in the operation. If an operation produces a word result in a general register, all 32 bits in the register are replaced by that result.

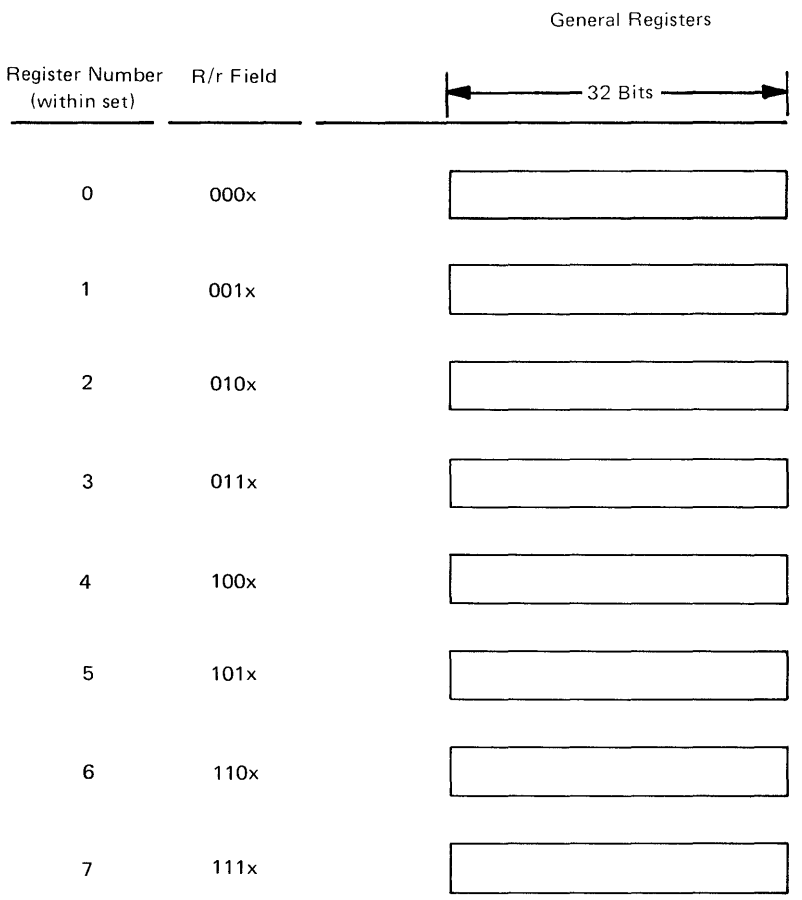
Operations on halfword operands in general registers use either bits 16-31 (called the lower half) or bits 0-15 (called the upper half) of the register. A halfword result placed in a general register replaces the contents of either the lower or upper half of the register. The contents of the other half of the register remain unchanged.

Operations on byte operands can refer to either of the 2 low-order bytes in a general register: bits 16-23 (called the upper byte) or bits 24-31 (called the lower byte). A byte result placed in a general register replaces either the upper byte or the lower byte; the contents of the remaining bit positions in the register are not changed. Except for data movement by means of the privileged register-indirect instructions, bits 0-15 of a general register *cannot* be used for either operands or results in operations on byte data units.

A general register may hold multiple data units, each of which can be processed independently. The three possible allocations of data units to a general register are either a 32-bit word (bit positions 0–31), two 16-bit halfwords (bit positions 0–15 and 16–31), or one halfword (bit positions 0–15) and two 8-bit bytes (bit positions 16–23 and 24–31). Refer to Figure 1-5 for data unit allocation for a general register.

General-register operands are designated by a 4-bit field in an instruction. The leftmost 3 bits of the field specify the binary number (0-7) of the general register within one of the two register sets assigned to the program. Figure 2-2 illustrates the numbering of general registers within a register set. The register set containing the general register is designated either by the rightmost bit of the 4-bit field or as part of the operation code.

The 4-bit field designating general-register word and halfword operands is identified in this publication by the uppercase symbol R, and is called the R field. For general-register byte operands, the field is identified by the lowercase symbol r and is referred to as the r field. For word and halfword operands, the rightmost bit of the R field designates which of the two register sets (primary or secondary) contains the general register. For halfword operands, the operation code indicates whether the operand is located in the upper or lower half of the general register. Word and halfword operands in general registers and the associated R-field values are shown in Figure 2-3.



Explanation:

For word and halfword operands, the rightmost bit (x) of the instruction R field is used to designate the register set in which the general register is located:

x = 0 designates the primary register set.
x = 1 designates the secondary register set.

For byte operands, the rightmost bit (x) of the instruction r field designates which of the two low-order bytes in the general register contains the operand:

x = 0 designates the upper byte (bits 16-23) of the register.
x = 1 designates the lower byte (bits 24-31) of the register.

For byte operands, the primary/secondary register set is either implied or specified by the operation code.

Figure 2-2. General Registers Within a Register Set

For byte operands, the rightmost bit of the r field designates which of the 2 low-order bytes within the general register contains the operand. The operation code indicates which of the two register sets (primary or secondary) contains the general register. The r-field values for primary-register-set byte operands are the same as the values for the secondary-register-set byte operands, as shown in Figure 2-4.

The general registers may be used as base-address registers in address generation. When general registers are used as base-address registers, they are designated by a 4-bit B field in an instruction. The bits of this field have the same organization as the bits of an R field.

For some operations, two adjacent general registers are treated as a pair. In these operations, the leftmost 3 bits of the R field in the instruction designate an even-numbered register. The next higher numbered register is implied to be the second register in the designated even/odd pair.

A number of general-register halfword operands may be addressed as a unit, called a quadrant. A *quadrant* consists of eight consecutive halfword operands. Within a register set there are two quadrants: one consisting of all of the halfword operands in the lower halves of the eight registers, and one consisting of all halfword operands in the upper halves of the registers. Thus, within the two register sets assigned to a program as general registers, there are four quadrants that the program can address.

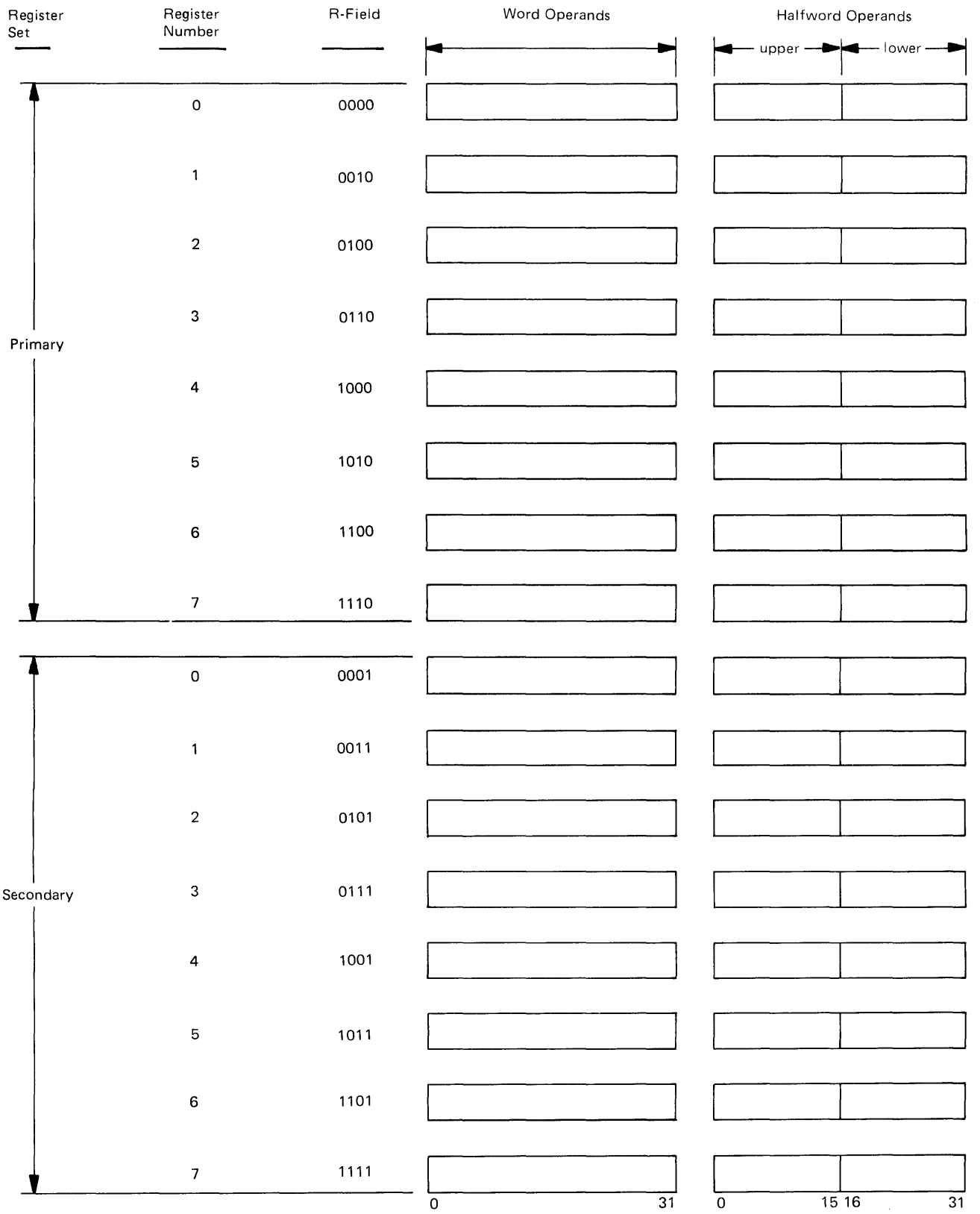


Figure 2-3. General-Register Word and Halfword Operands

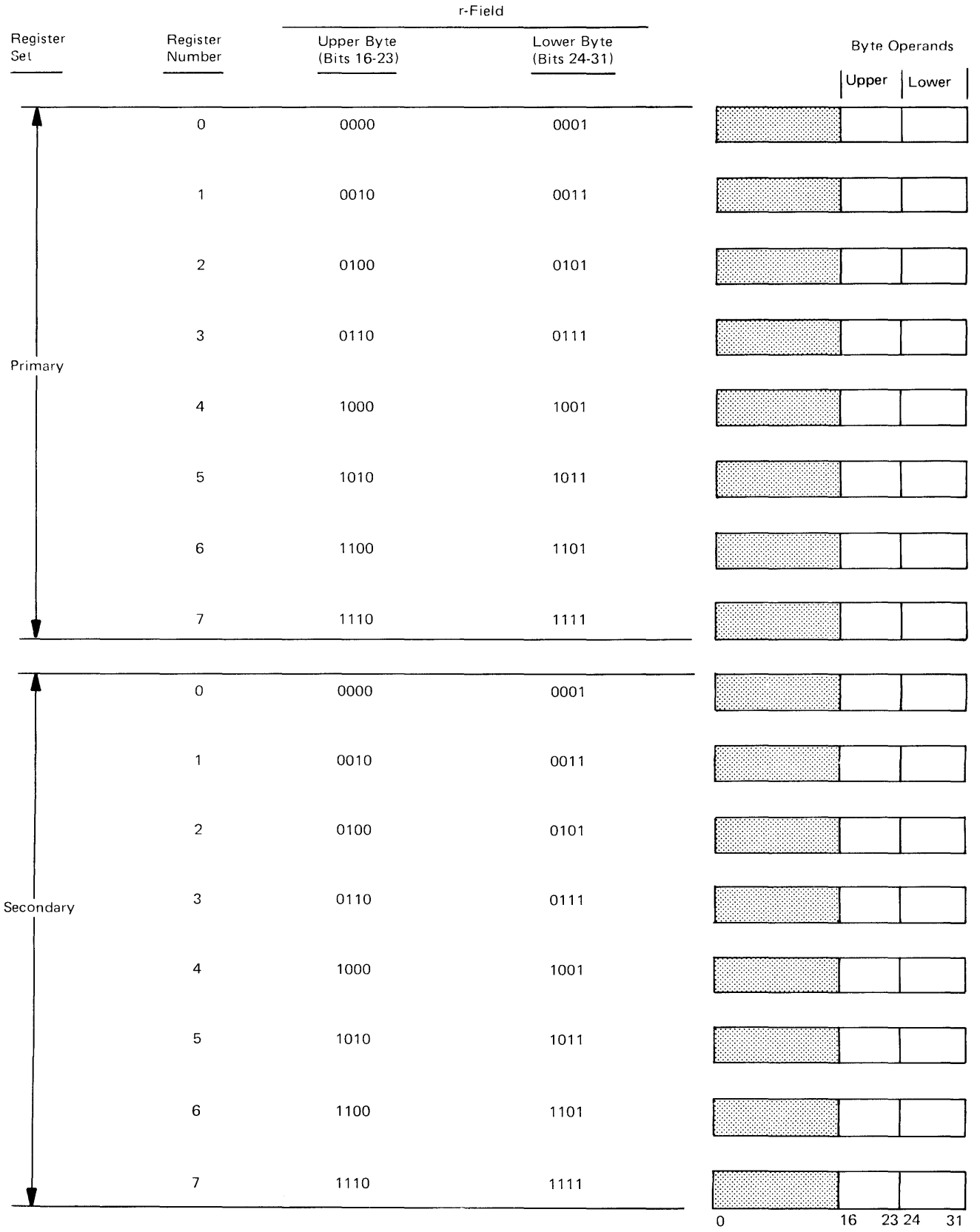


Figure 2-4. General-Register Byte Operands

Floating-Point Registers

When the floating-point feature is installed, floating-point operations are provided on data residing in floating-point registers. Each floating-point register contains 64 bits (doubleword). These registers can be used as accumulators in arithmetic operations on floating-point data.

Floating-point registers are organized in register sets. Each set consists of four floating-point registers numbered 0-3. Eight floating-point register sets are provided for assignment to programs when the floating-point feature is installed. Each program can have one floating-point register set assigned to it. A field in the floating-point status vector designates the number of the set currently assigned to the program. When a program is executing, the floating-point register set designated in the program's floating-point status vector contains the *active floating-point registers.*

Operands in floating-point registers can be either short-format (32-bit) or long-format (64-bit) floating-point data. A short operand occupies the high-order bit positions (0-31) of a floating-point register. The low-order portion (bit positions 32-63) is ignored and remains unchanged in arithmetic operations calling for short operands and a short result.

Floating-point registers are addressed by a 2-bit F field in instructions. These bits specify the floating-point register number within the register set assigned to the program. Figure 2-5 illustrates the numbering of floating-point registers within a register set.

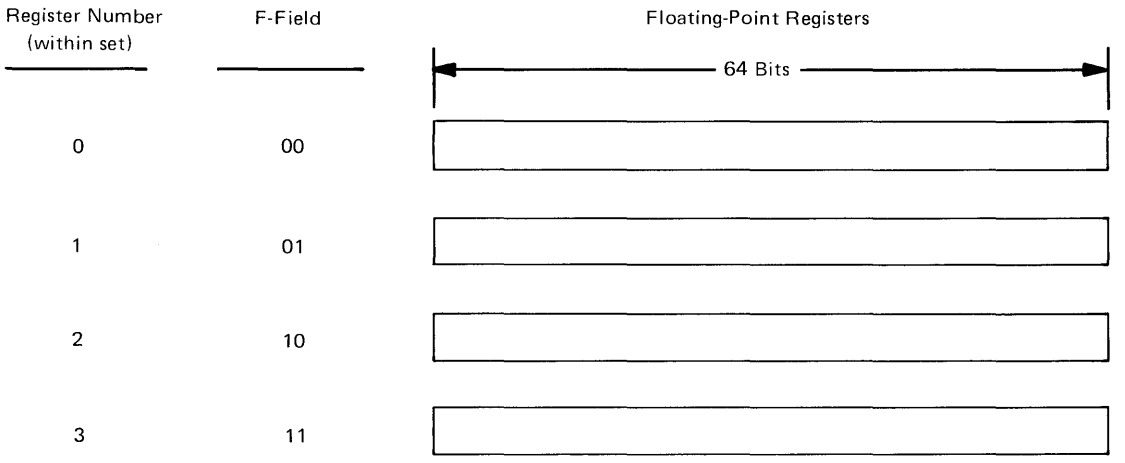


Figure 2-5. Registers in a Floating-Point Register Set

Chapter 3. Program Execution

This chapter describes the role of instructions during program execution, and discusses briefly program status information used by the PCE to control execution of a program. Included is a detailed description of program exceptions other than those related to floating-point operations. Chapter 5, "Floating-Point Instructions," describes floating-point in detail. The action taken by the PCE when an interruption occurs due to a program exception is described under "Interruptions" in Chapter 9, "PCE Control."

Instructions

Each instruction consists of two major parts: (1) operation code fields, which specify the operation to be performed, and (2) fields which designate the operands that participate.

Operands

Operands participating in an operation can be grouped in three classes: operands located in registers, immediate operands, and operands in main storage. Operands may be either explicitly or implicitly designated. For purposes of description, operations may be grouped according to the locations of their operands: register-to-register, register-and-storage, register-and-immediate, storage-to-storage, and sequencing.

Register operands can be located in general or floating-point registers, with the type of register identified by the operation code fields. For all instructions except floating-point, the register containing the operand is specified by identifying the register in a 4-bit field, called the R field, in the instruction. (In the individual instruction descriptions, the 4-bit fields that designate general-register byte operands are denoted with a lowercase r in place of the uppercase R.) For floating point instructions, the floating-point register containing the operand is designated by a 2-bit F field. For some instructions, an operand is located in an implicitly designated register, in which case the register is implied by the operation code fields.

Immediate operands are contained within instructions, and the field containing the immediate operand is called the I field.

Operands in main storage may either have an implied length or a length specified by the contents of a general register. The addresses of operands in main storage may be specified by a format that uses the contents of a general register as all or part of the address. This makes it possible to:

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

The address used to refer to main storage either is contained in a register designated by the R field in the instruction or is calculated from a base address and displacement, designated by the B and D fields, respectively, in the instruction.

In describing instruction execution, operands are designated as first and second operands and, in some cases, third operands. In general, two operands participate in an instruction execution, and the result replaces the first operand. An exception is an instruction with STORE in the name, in which the result replaces the second operand. Except for storing the final result, and for instructions that include address modification as part of the operation, the contents of all registers and storage locations participating in the addressing or execution part of an operation remain unchanged.

Instruction Formats

An instruction is 1 or 2 halfwords long and must be located in main storage on an integral halfword boundary. For purposes of describing operand designations, each instruction is treated as having one of eight general formats: RR, RR-Long, RS, RS-Long, RI, J, FF, and FS. The format names express, in general terms, the form used to specify the operands that participate in the operation, *not* the class of operation. For example, the RR format is used for register-to-register operations, and for certain register-and-storage operations in which the address of the storage operand is contained in a register. In both types of operation, the two operand specification fields in the instruction designate general registers and thus the format name is RR. The eight general formats are illustrated in Figure 3-1.

The RR-Long format is used for register-to-register operations, and for storage-to-storage operations in which the addresses of both operands are contained in registers. Both RS formats are used only for register-and-storage operations that require specification of a register operand and the base and displacement components of a main storage address. RI formats are normally used for operations that require specification of a register operand and immediate data. The J format is used for sequencing operations in which the instruction address is always used as a base address. The FF and FS formats are used only for floating-point operations: FF is used for register-to-register operations and FS is used for register-and-storage operations.

The first four bits of all instructions contain an operation code field (Op Code). For most instructions, additional fields are also used for operation code information. Some instructions contain fields that vary somewhat from the general format, and some instructions do not follow the general rules for operand specification stated in this section. All such exceptions are explicitly identified in the individual instruction descriptions.

Register Operand Specification

In the RR, RS, and RS-Long instruction formats, the R_1 field designates the general register containing the first operand. In the RR-Long format, the R_1 field designates the general register containing the first operand for register-to-register operations. In the RI format, the r_1 field designates the first operand which, in most cases, is one byte contained in a general register. For FF and FS formats, the floating-point register designated by the F_1 field contains the first operand.

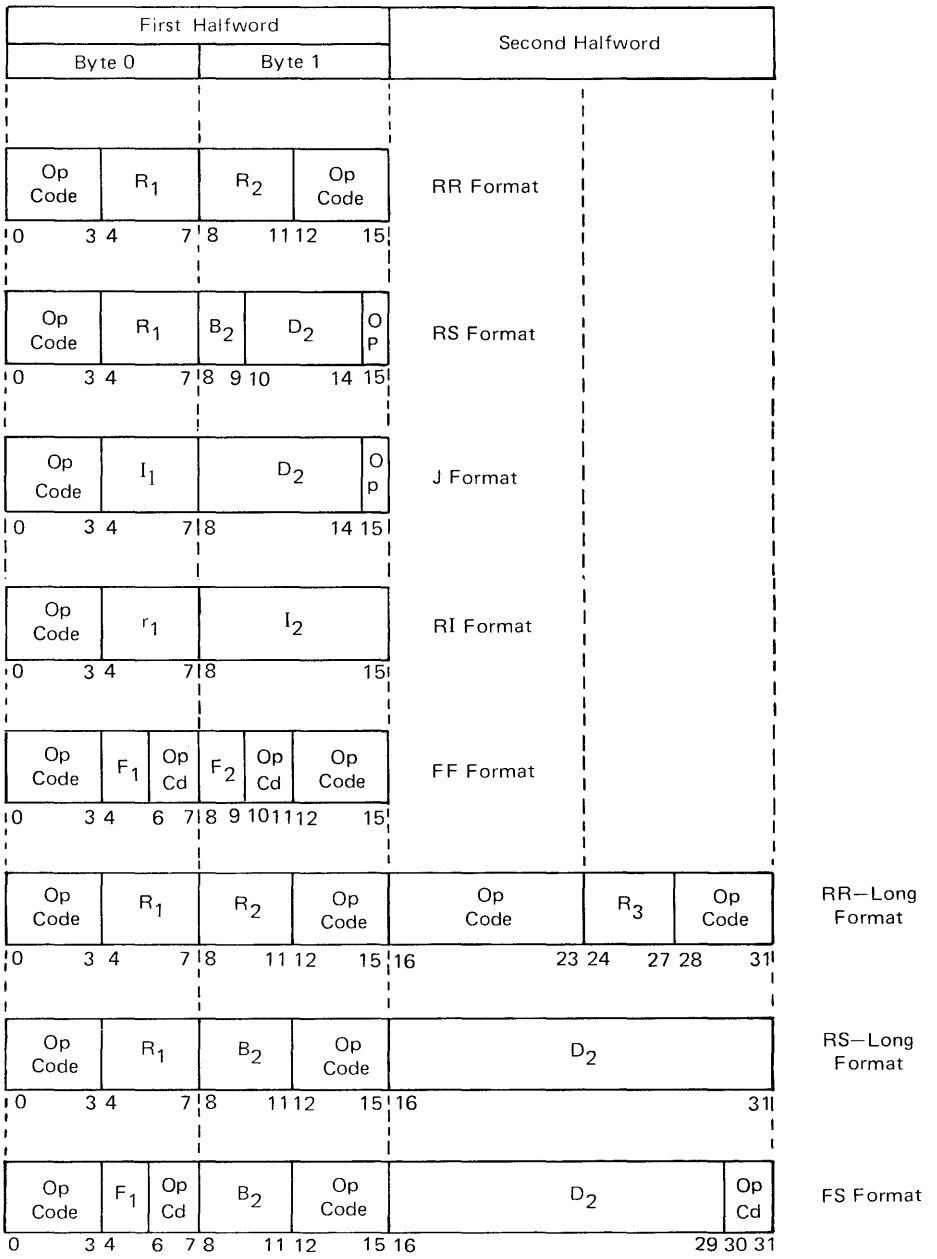


Figure 3-1. General Formats of Instructions

In the RR and RR-Long formats used for register-to-register operations, the R_2 field designates the general register containing the second operand. The R_3 field of the RR-Long format is not defined for register-to-register operations. In the FF format, the F_2 field designates the floating-point register containing the second operand. In all formats for register-to-register operations, the same register may be designated for the first and second operands. The content of register 0 may not be used in certain operations. When register 0 is specified for these operations, the operand is either an implied 0 or the updated content of the instruction address.

Operands in general registers may have lengths of 1 byte (8 bits), 1 halfword (16 bits), or 1 word (32 bits); the second operand is the same length as the first.

Operands in floating-point registers may be short-format (32-bit) or long-format (64-bit) floating-point data. The second operand has the same format, and thus the same length, as the first.

Immediate Operand Specification

In the RI format used for register-and-immediate operations, the contents of the 8-bit immediate-data field, the I_2 field of the instruction, are used directly as the second operand. In certain operations specified in the RR format, a 4-bit immediate-data field, I_2, is defined in place of one of the R fields, and is used directly as the second operand.

For programmed input/output or PCE-control operations using the RI format, the I_2 field contains the command code for an I/O device or an extension to the operation code, respectively.

Storage Operand Specification

In the RR format for register-and-storage operations, the contents of the general register designated by the R_2 field are used as the second-operand address. In the RS, RS-Long, and FS 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 RR-Long format for storage-to-storage operations, the contents of the general register designated by the R_1 field are used as the first-operand address. The contents of the general register designated by the R_2 field are used as the second-operand address. The contents of bit positions 24-31 of the general register designated by the R_3 field are used as the length of the operands in storage-to-storage operations. The second operand has the same length as the first operand. The length of the operands is 1 to 256 data units (bytes or halfwords). The maximum length is obtained by specifying a value of 0 in bit positions 24-31 of the general register specified by R_3. Results replace the first operand and are never stored outside the field specified by the first-operand address and the length.

Address Generation

The address used to refer to main storage either is contained in a register designated by an R field in the instruction, or is calculated from the following two binary numbers: base address and displacement.

Base Address

The base address is a 32-bit unsigned binary integer, all bits of which are used to represent an address value that is treated as positive. The base address is referred to in instructions either as the contents of a general register, or as the updated instruction address. Base addresses can be used as a means of independently addressing each program and data area. In array-type calculations, they can specify the location of an array, and in record-type processing, they can identify the record.

When the base address is the contents of a general register, the register is designated in the instruction's B field. Except in the RS instruction format, this field occupies four bit positions, which allows for the specification of any of the 16 general registers as a base register. In the RS format, the B field occupies two bit positions. This field is considered to be the two low-order bit positions of an instruction R field in which the two high-order bit positions contain 1's. The 4-bit

R field is described in detail under "General Registers" in Chapter 2. Only general registers 6 and 7 in both the primary and secondary register sets may contain base addresses for operands of RS-format instructions.

For instructions in the RS Long format when the B field contains all 0's, the contents of the general registers are *not* used as the base address. Instead, the updated instruction address from the current program status vector is used. During address generation, the updated instruction address and the displacement are algebraically added without causing a carry from the low-order bit (bit 31):

- For LOAD (L, LH, LW), STORE (ST, STH, STW), and LOAD ADDRESS (LA) instructions, bit 31 of the updated instruction address is ignored (assumed to be 0).

- For BRANCH AND LINK (BAL) and BRANCH on CONDITION (BC) instructions, bit 31 of the instruction address participates during address generation. However, the value of bit 31 is unchanged, since only even-valued displacements are used in the address computation.

For instructions in the J format, the updated instruction address is always used as a base address.

The following conditions involving bit 31 of the instruction address should be considered when evaluating address generation:

- Instructions must appear on halfword integral boundaries. Thus, during instruction fetch references, bit 31 of the instruction address is ignored (assumed to be 0).

- Bit 31 is loaded only by means of a BRANCH type instruction or when a new PSV is introduced. During updating for the next sequential instruction, the instruction address is incremented by an even number of bytes (2 or 4), thereby preserving the value of bit 31 (0 or 1) until a subsequent BRANCH type instruction is issued or when a new PSV is introduced.

- Bit 31 is saved by a BRANCH AND LINK (BAL or BALR) instruction, or when a new PSV is introduced.

- For BRANCH AND LINK (BAL) and BRANCH ON CONDITION (BC) instructions when the B field contains all 0's, and for JUMP ON CONDITION (JC) and JUMP ON BIT ZERO (JBZ) instructions, the instruction address is updated by adding an even-valued displacement, thereby preserving the value contained in bit 31.

Programming Note: For an RS-Long format instruction, the contents of primary general register 0 cannot be used as a base address.

Displacement

Displacement is a binary number contained in a field, called the D field, in the instruction. The displacement provides for addressing relative to 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.

Except for the RS instruction format, the displacement is designated as a signed binary integer. Positive displacements are represented in true binary notation with the sign bit set to 0. Negative displacements are represented in twos-complement binary notation with a 1 in the sign-bit position. In generating the address, a positive displacement is logically extended to 32 bits with high-order 0's; a negative displacement is logically extended with high-order 1's. A 0-value D field has no special significance.

In the RS instruction format, the displacement is designated as an unsigned binary integer; all bits are used to represent the value which is treated as positive. An unsigned displacement is logically extended to 32 bits with high-order 0's.

During address generation, all 32 bits of the base address are added to all 32 bits of the logically extended displacement. The sum, which represents the generated address, is then checked for validity. If the sum is valid, the operation proceeds, treating the generated address as a 32-bit unsigned integer. When an invalid sum is computed, and a storage reference is attempted using the sum as the generated address, an address exception is indicated.

The sum is considered invalid in the following cases:

- When the sum of the base address and an extended displacement is greater than the maximum logical address available to the program.

- When the sum of the base address and an extended displacement is less than 0. This occurs when the magnitude of a negative displacement is greater than the base address (conceptually yielding a negative sum).

An instruction can designate the same general register both as the base register and as the location of and operand. Address generation is completed before execution of the operation. The computed operand address designates an operand in main storage. For branching instructions, the second-operand address is used as the branch address.

In the RS format, the displacement is contained in a 5-bit D field that specifies the displacement as an integral number of halfwords. The displacement is treated as an unsigned 6-bit positive binary integer by appending a 0 in the low-order bit position. This low-order 0 bit has the effect of multiplying the displacement by 2 so that it is expressed in terms of an even number of bytes. The displacement provides for addressing of up to 62 bytes beyond the location specified by the base address.

In the J format the displacement is contained in a 7-bit D field that specifies the displacement as an integral number of halfwords. The displacement is treated as a signed 8-bit binary number by appending a 0 as the low-order bit. This low-order 0 bit has the effect of multiplying the displacement by 2 so that it is expressed in terms of an even number of bytes. The displacement is combined with the updated instruction address, and thus provides for addressing 128 bytes preceding the location designated by the updated instruction address, and 126 bytes beyond it.

In the RS-Long format, the displacement is contained in a 16-bit D field. For all instructions using the RS-Long format, except BRANCH ON CONDITION and BRANCH AND LINK, the D field contains a signed 16-bit binary integer that

specifies the displacement as an integral number of bytes. The D field provides for addressing 32,768 bytes preceding the location designated by the base address and 32,767 bytes beyond it.

The 16-bit D field in BRANCH ON CONDITION and BRANCH AND LINK instructions specifies the displacement as an integral number of halfwords. The contents of the D field are treated as a 17-bit signed binary integer. The low-order bit of the D field contains the sign bit. During address generation, the sign bit is moved to the high-order bit position in the PCE's representation of the displacement, and a low-order 0 bit is appended to the representation. This low-order 0 bit has the effect of multiplying the displacement by 2 so that it is expressed in terms of an even number of bytes. Figure 3-2 illustrates this representation. The D field provides for addressing 65,536 bytes preceding the location designated by the base address and 65,534 bytes beyond it.

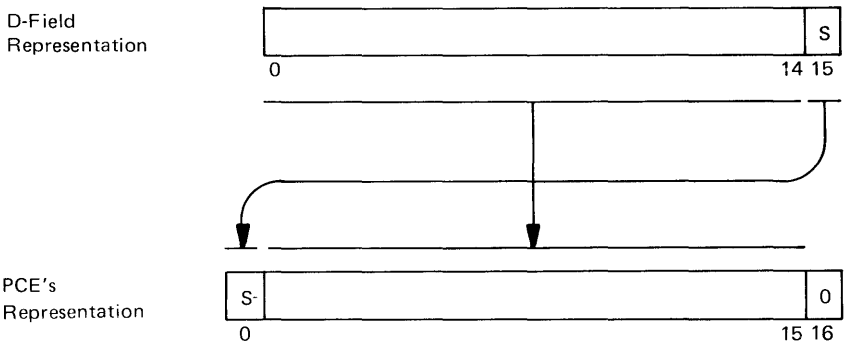


Figure 3-2. Displacement of RS-Long Format BRANCH Instructions

In the FS format, the displacement is contained in a 14-bit D field that specifies the displacement as an integral number of words. The contents of the D field are treated as a 16-bit signed binary integer by appending two low-order 0's. These two low-order 0 bits have the effect of multiplying the displacement by 4 so that it is expressed in terms of a number of bytes. The D field provides for addressing 32,768 bytes preceding the location designated by the base address and 32,764 bytes beyond it.

Certain operations include modification of an operand address, contained in a general register, as part of the operation. For all such operations, both the original address value and the modified address are treated as unsigned 32-bit positive binary integers. An address exception is indicated when an incremented address that exceeds the maximum logical address available to the program is used to refer to main storage.

Execution of a Program

Normally, operation of the PCE is controlled by instructions executed in sequence. This sequence is governed by the program status vector which contains the primary information required for proper program execution. A change in the sequential operation may be caused by branching or by introduction of a new program status vector.

Program Status Vector

The program status vector (PSV) is 64 bits long and contains the information required for proper program execution. The PSV includes the instruction address, condition indicators, register-set numbers, and other fields. The PSV format is described in Chapter 9, "PCE Control." In general, the PSV is used to control instruction sequencing and to hold and indicate the status of the system in relation to the program currently being executed. The active or controlling PSV is called the *current* PSV. By storing the current PSV, the status of the PCE can be preserved for subsequent inspection. By loading a new PSV or part of a PSV, the state of the PCE can be initialized or changed.

Floating-Point Status Vector

The floating-point status vector (FSV) is 24 bits long and contains additional information required for program execution when floating-point operations are used. The FSV includes the floating-point register-set number, masks, status indicators, and other fields. The FSV format is described in Chapter 9, "PCE Control." In general, the FSV is used to hold and indicate additional status for programs using floating-point operations. The active FSV is called the *current* FSV.

Instruction Execution

In program execution, an instruction is fetched from the location designated by the instruction address in the current PSV. The instruction address is then increased by the number of bytes in the fetched instruction in order to address the next instruction in sequence. (This new value of the instruction address is referred to as the *updated instruction address.)* The fetched instruction is then executed, and the same steps are repeated using the new value of the instruction address.

Instructions must appear on halfword integral boundaries. During the fetching of instructions, the low-order bit of the instruction address is ignored and assumed to be 0.

Branching

The normal sequential execution of instructions may be changed by use of branching operations in order to perform subroutine linkage, decision-making, and loop control.

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

Facilities for decision making are provided by the BRANCH ON CONDITION and JUMP ON CONDITION operations. These operations test any of five logical entities, called *result conditions,* that indicate the outcome of arithmetic, logical, and I/O operations. For example, such outcomes as 0 sum, first operand high, overflow, equal, mixed 0's and 1's, and carry may be indicated. BRANCH ON CONDITION and JUMP ON CONDITION instructions can specify combinations of the five result conditions as the criterion for branching. The 4-bit M1 field of these instructions is used as a mask specifying the result conditions to be tested.

The five result conditions are numbered 8, 4, 2, 1, and 0. These numbers correspond to the mask value represented in the M1 field as shown in the following table:

M1 Field	Mask Value	Result Condition Tested
1000	8	8
0100	4	4
0010	2	2
0001	1	1
0000	0	0

The specific meaning associated with any result condition depends on the particular instruction. For example, result condition 8 indicates a 0 sum for an addition operation, and indicates equal operands for a comparison operation.

After execution of an instruction for which result conditions are specified, each of the five conditions is placed in one of two possible states: *indicated* or *not-indicated.* Thus all five result conditions reflect only the outcome of that instruction. If a result condition is indicated, the outcome of the instruction execution is described by the meaning associated with the indicated condition. If a result condition is not-indicated, the meaning associated with that condition does not apply to the outcome. Any result condition that is not assigned a meaning for a particular instruction is also not-indicated. The states of the five result conditions remain unchanged after execution of any instruction for which result conditions are not specified.

Result conditions 8, 4, and 2 are mutually exclusive; that is, only one of these conditions can be indicated at any one time. For example, an addition cannot produce a result that is at the same time zero (8), less than zero (4), and greater than zero (2). One of the three, however, is always indicated. Result conditions 1 and 0 are indicated independently; that is, one or both of these conditions may be indicated concurrently with condition 8, 4, or 2. Thus, one, two, or three result conditions are indicated at any one time. The states of the five result conditions are derived from the settings of condition-indicator bits in the current PSV (see Chapter 9, "PCE Control").

A branch or jump may be made on more than one result condition, except condition 0, by specifying a value in the M1 field that is the sum of the individual result-condition numbers. For example, a mask value of 13 (binary 1101) specifies that a branch is to be made if any combination of result conditions 8, 4, and 1 is indicated. An unconditional branch or jump is made by specifying a mask value of 14 or 15.

Loop control can be performed by the use of the BRANCH ON CONDITION or JUMP ON CONDITION operations to test the outcome of address arithmetic or counting operations. For some particularly frequent combinations of counting and testing, the instruction BRANCH ON COUNT is provided.

Another facility for decision making is provided by the instruction JUMP ON BIT ZERO. The jump criterion is satisfied if the specified bit has the value 0.

An n-way branch is provided by the instruction BRANCH ON INDEX, which uses the specified index value to select the new address from a table.

Introduction of a New PSV

Sequential execution of instructions is also changed when a new PSV is introduced. This occurs as a result of an interruption or execution of CALL PSV or DISPATCH NEW LEVEL. One type of interruption, a program-exception interruption, occurs when the PCE recognizes a program exception. Program exceptions that cause an interruption are described in this chapter and in Chapter 5, "Floating-Point Instructions." Refer to Chapter 9 for a detailed discussion of interruptions and DISPATCH NEW LEVEL and to Chapter 4 for a description of CALL PSV.

Interruptible Instructions

The MOVE and COMPARE LOGICAL operations are referred to as interruptible instructions. That is, an interruption is permitted after partial execution of the instruction. Whenever discussion in this publication pertains to points of interruption that include those occurring within the execution of an interruptible instruction, the term *unit of operation* is used. This use of the term considers that the entire execution of a noninterruptible instruction consists, in effect, of one unit of operation.

The execution of an interruptible instruction is considered to consist of a number of units of operation, and an interruption is permitted between units of operation. Depending on processor model, more than one unit of operation could be executed between points in the operation at which an interruption is allowed. In this case, the number of units of operation executed without allowing interruptions is predetermined. After each predetermined number of units of operation, the operand addresses and the count value are updated to correspond to the amount of data processed. The specific predetermined number of units of operation is fixed, except for the first and last execution groups.

Sequence of Storage References

Conceptually, the PCE executes instructions one at a time, with the execution of one instruction preceding the execution of the following instruction. Also, the execution of the instruction specified by a successful branching operation follows the execution of the branch. The sequence of events implied by this manner of instruction execution is sometimes called the *conceptual* sequence or *conceptual* order.

Physical storage width and the overlap of instruction execution with storage accessing may cause actual processing to be different. As observed by the program itself, each operation is performed sequentially, with one instruction being fetched after the preceding operation is completed and before execution of the following operation (the just-fetched instruction) is begun. With certain exceptions discussed in the following section, "Instruction Fetch," and in Chapter 7, "Dynamic Address Relocation and Translation," the results generated are those that would have been obtained had the operation been performed in the conceptual sequence.

Storage references associated with instruction execution are of the two types: instruction fetches and storage-operand references. Synchronization of storage references between PCEs in processors having two PCEs is described in Chapter 10.

Instruction Fetch

Instruction fetching consists of fetching the 1 or 2 halfwords specified by the instruction address in the current PSV. The immediate-operand field of an instruction is accessed as part of an instruction fetch. If an instruction specifies a storage operand at the location occupied by the instruction itself, the location is accessed both as an instruction and as a storage operand. The instruction may be fetched multiple times for a single execution; for example, an interruptible instruction may be fetched more than once between units of operation.

Instructions are not necessarily fetched in the order in which they are conceptually executed. In particular, the fetching of an instruction may precede the storage-operand references for a conceptually earlier instruction. As a consequence, modification of an instruction in storage by a conceptually earlier instruction does not change the prefetched copy of the instruction. In all cases, however, the instruction fetch occurs before all storage-operand references for conceptually later instructions.

The number of instructions prefetched depends on the processor model. Therefore, programs that attempt to modify conceptually later instructions may not yield the same results on all processor models. Storing caused by channel I/O operations does not change the copy of prefetched instructions.

Conceptual sequential instruction execution within a PCE is assured when either (1) a new PSV is introduced, (2) an entry is stored in the translation table or the translation lock table when dynamic address translation is active, or (3) a BRANCH or JUMP operation is executed causing the instruction address to be replaced with the branch or jump address. The latter case includes the designation of the address of the next sequential instruction as the branch or jump address. Refer to Chapter 10 for a description of instruction execution in dual-PCE processors.

Programming Note: As observed by the program itself, instruction prefetching is not normally apparent; the only exception occurs when an instruction attempts to modify a conceptually later instruction that was prefetched by the PCE. If a program modifies the instruction stream, which could affect a prefetched instruction, and the change must take effect immediately, the instruction causing the change should be followed by a BRANCH or JUMP instruction, which causes the instruction address to be replaced.

Storage-Operand References

A storage-operand reference is the fetching or storing of the explicit operand or operands in the main storage locations specified by the instruction. Storage-operand references are of three types: fetches, stores, and updates.

Storage-Operand Fetch References

When the bytes of a storage operand participate in the instruction execution only as a source, the reference to the location is called a storage-operand fetch reference.

All bits within a single byte of a fetch reference are accessed concurrently. When an operand data unit consists of more than 1 byte, the bytes may be fetched

piecemeal from main storage. The bytes within a halfword, word, or doubleword data unit are not necessarily fetched in any particular order. The bytes, however, appear in the expected order in the destination.

Storage-Operand Store References

When the bytes of a storage operand participate in the instruction execution only to the extent of being replaced by the result, the reference to the location is called a storage-operand store reference.

All bits within a single byte of a store reference are accessed concurrently. When an operand data unit consists of more than 1 byte, the bytes may be stored piecemeal into main storage. Unless otherwise specified, the bytes within a halfword, word, or doubleword data unit are not necessarily stored in any particular order. The bytes in storage, however, appear in the expected order.

The results of one instruction are placed in main storage after the results of all preceding instructions are placed in main storage and before any results of the succeeding instructions are stored. For any one instruction that operates on multiple data units, the data units are stored in the order specified for that instruction.

An operand data unit is not fetched from a main-storage location until all information destined for that physical main-storage location, due either to the execution of a preceding instruction or the current instruction, are placed in main storage. In a MOVE operation in which the operand fields overlap, for example, the store reference to a given main-storage location is completed before any conceptually later fetch reference to that location is made. As noted earlier, a prefetched copy of an instruction is not modified even when the instruction in storage is changed before the prefetched instruction is executed.

Storage-Operand Update References

In the instruction TEST AND SET, the storage-operand location participates both as a source and as a destination. In this case, the reference to the location consists first of a fetch and then of a store. The combination of the two accesses is referred to as an update reference. In TEST AND SET, the update reference is interlocked against accesses to storage during instruction execution.

Programming Note: Two programs may attempt to update information at a common main-storage location using a sequence of operations. Because of the priority-level dispatching mechanism, it is possible for both programs to fetch the same information, modify it, and then store the updated information. The change made by the higher-priority program in such a case is lost when the lower-priority program completes the update. The instruction TEST AND SET enables updating of such a common storage location. In order not to lose the change by either program, both programs should use this single instruction in an established program convention that provides an interlocked update.

Program Exceptions

Exceptions resulting from execution of the program, including the improper specification or use of instructions and data, cause a program-exception interruption. A program-exception interruption causes the current PSV to be stored and a new PSV to be introduced. The information in the stored PSV

permits identification of the program exception and the last instruction executed. Chapter 9, "PCE Control," describes in detail the storing and introduction of PSV information.

When a program-exception interruption occurs, the instruction-address field and instruction-address-modifier field in the stored PSV identify the location of the instruction associated with the program exception. The bit in the instruction-address-modifier field indicates whether the instruction address designates the location of the first byte of the instruction or two bytes beyond it.

Types of Ending

When a program exception is recognized, instruction execution ends in one of four ways: completion, suppression, suspension, or termination.

- When instruction execution is *completed,* results are provided as called for in the definition of the instruction.

- When instruction execution is *suppressed,* the instruction is effectively not executed. That is, the contents of any result fields defined for this instruction are not changed. This includes the designated operand location, the condition indicators in the current PSV, and any address or count value due to be changed by the instruction.

- When instruction execution is *suspended,* the contents of any fields due to be changed by the instruction may be partially updated. The operation may have replaced all, part, or none of the contents of the designated operand locations, and it may have changed the condition indicators, an address value, or a count value, if such change was called for by the instruction. The instruction may be retried without software adjustment of register values, assuming the cause of suspension is removed.

- When instruction execution is *terminated,* the contents of any fields due to be changed by the instruction are unpredictable. The operation may have replaced all, part, or none of the contents of the designated operand locations, and it may have changed the condition indicators, an address value, or a count value, if such change was called for by the instruction.

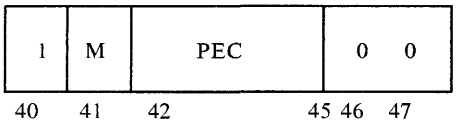
When a program-exception interruption occurs after completion or suspension of a unit of operation of an interruptible instruction, all prior units of operation are completed. The address and count values are adjusted to correspond with the last completed unit of operation.

When a program-exception interruption occurs after the termination of a unit of operation of an interruptible instruction, all prior units of operation are completed. The address and count values are adjusted to correspond to the last completed unit of operation. Depending on processor model and the point in the operation at which the exception is detected, one or more units of operation can be terminated before the interruption.

Programming Note: Program-exception interruptions usually cause instruction execution to be suppressed, suspended, or completed. In some cases, however, the instruction may be terminated. The specific cases are noted under "Program Exception Conditions" in this chapter and in the individual instruction descriptions.

Exception Information

The cause of a program exception is identified by the program information code in bit positions 40–47 of the stored PSV. The format of the program information code in the PSV is shown in the following figure:



The fields in the program information code are allocated as follows after a program-exception interruption occurs:

Program Exception: Bit position 40 of the stored PSV is set to 1 to identify the information as program-exception information.

Intruction Address Modifier (M): Bit position 41 contains the instruction address modifier. A 0 in bit position 41 indicates that the instruction address field (bit positions 0-31) in the stored PSV contains a value 2 greater than the address used to fetch the first halfword of the instruction associated with the program exception. A 1 in bit position 41 indicates that the instruction address contains the address used to fetch the first halfword of the instruction.

Program Exception Code (PEC): Bit positions 42-45 of the stored PSV contain the program exception code identifying the cause of the program exception, as shown in the following table. Only one exception can be indicated at a time.

Bits 42-45	Code	Program Exception
0000	0	Specification Exception
0001	1	Access Exception
0010	2	Operation Exception
0011	3	Separation Exception
0100	4	Address Exception
0101	5	Register-Indirect Exception
0110	6	Fixed-Point-Overflow Exception
0111	7	Floating-Point Exception
0000-1110	8-14	Reserved, are not indicated
1111	15	Available for programming use

Reserved Bits: Bit positions 46 and 47 are reserved and are set to 0's.

Programming Note: Code 15 is not indicated by the PCE; it is intended to use by supervisory program. Bit positions 46 and 47 of the stored PSV are reserved. Therefore, a program should not depend on 0's being placed in these bit positions.

Program Exception Conditions

The conditions that cause a program-exception interruption are detailed in the following sections. In addition to these conditions, a program-exception interruption may be caused by executing the instruction PROGRAM EXCEPTION, as described in Chapter 4.

Specification Exception (code 0)

A specification exception is indicated for the following conditions:

- ***PSV/ACV Format:***

  - A PSV is introduced that contains a 1 in a reserved bit position (bit positions 32-35).

  - An ACV is introduced that contains a 1 in a reserved bit position (bit positions 0-7).

  - An ACV is introduced that contains an invalid value in the size field. The invalid value may exceed the maximum size provided by the PCE, or the value may not be defined.

  - An ACV is introduced that contains an origin-address value that exceeds the maximum address in the PCE address space.

    The introduction of the new PSV and ACV is completed, but a program-exception interruption occurs thereafter. Refer to Chapter 9, "PCE Control," for a discussion of when the exceptions associated with the PSV and ACV are indicated.

    A program exception other than specification may be indicated when an invalid PSV or ACV, as defined by the preceding four points, is introduced. For more information, refer to "Handling of Multiple Program Exceptions" later in this chapter.

- ***Operand:***

  - The first operand of STORE TO ADDRESS TRANSLATION TABLE contains either (1) a 1 in a bit position corresponding to a reserved bit position in the translation-table entry (bit positions 3, 5-10), or (2) a block address that designates a location that exceeds a specific value. This value depends on processor model, and is equal to the size of the PCE address space when dynamic address translation is inactive.

  - The first operand of STORE TO ADDRESS TRANSLATION LOCK TABLE contains one or more 1's in bit positions 16-23.

  - The translation-table index specified in LOAD FROM ADDRESS TRANSLATION TABLE or STORE TO ADDRESS TRANSLATION TABLE exceeds the number of entries in the translation table provided by the PCE or, depending on processor model, bit positions 0-15 of the second operand for these instructions contain one or more 1's.

  - The translation-lock-table index specified in LOAD FROM ADDRESS TRANSLATION LOCK TABLE or STORE TO ADDRESS TRANSLATION LOCK TABLE exceeds the number of entries in the translation lock table provided by the PCE, or bit positions 0-10 of the second operand for these instructions contain one or more 1's.

  - Depending on processor model, specifying nonzero values for the contents of r_1 corresponding to bit 7 of a WRITE EIRV, bits 0 and 1 of a WRITE

PRIMARY REGISTER SET NUMBER and WRITE SECONDARY REGISTER SET NUMBER, and bit 4 of a DISPATCH NEW LEVEL instruction may cause a specification exception.

- The WRITE PROGRAM ACTIVATION VECTOR instruction attempts to change the state of the PAV bit corresponding to the active PSV/ACV pair. Depending on processor model, the PCE may indicate a specification exception and not allow the program to change the state of the PAV bit associated with the current priority level.

- ***Real Address:***

  The PCE causes a reference to a physical main-storage location that is not installed. A real address designating an uninstalled storage location is referred to as invalid.

  When dynamic address translation is active, a specification exception is indicated when the real address, after translation, is invalid. When translation is inactive, the exception is indicated when the real address, after dynamic address relocation, is invalid.

  When part of an operand location is designated in installed main storage and part is not, storing may be performed in the part associated with the installed storage. A specification exception for an operand that is partially designated in uninstalled storage is not indicated when the operation can be completed without use of the inaccessible part of the operand.

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

  For BRANCH or JUMP operations, the target address is tested for validity as part of the execution of the BRANCH or JUMP operation. If a specification exception is detected for the first halfword associated with the target address, the IA in the stored PSV indicates the BRANCH or JUMP as the failing instruction.

A specification exception is indicated only when the PCE attempts to execute the instruction with which the exception is associated. In particular, the exception is *not* indicated when the PCE attempts to prefetch an instruction from an inaccessible location or otherwise detects the specification exception, but a branching operation or an interruption changes the instruction sequence such that the instruction is not executed.

A specification exception is recognized when the real address designated by a translation-table entry does not correspond to an installed physical location. Depending on the particular processor model and on the block-address value, either (1) a specification exception (for operand) is recognized when an attempt is made to store the entry, or (2) a specification exception (for real address) is recognized when the PCE attempts to refer to the uninstalled location.

The PCE may indicate a specification exception instead of an address exception (code 4) when it encounters an address-limit condition. More details are provided under "Address Exception" later in this chapter.

Execution of the instruction identified by the stored PSV is suppressed except for the following special cases: A MOVE, COMPARE LOGICAL, or LOAD QUADRANT operation is terminated and a STORE QUADRANT or floating-point LOAD or floating-point STORE operation is suspended when a specification exception for an invalid real address is detected.

Access Exception (code 1)

An access exception can be detected only when dynamic address translation is active (bit position 31 in the ACV is 1). An access exception is indicated when the program causes a reference to a main-storage location that is protected against that type of reference, as specified by the access-control field in the translation-table entry corresponding to the logical address.

Depending on processor model, when an access exception is indicated, the block index of the PCE address in error is placed in the EBI register associated with the active ACV.

An access exception is indicated for the following conditions:

- ***Block Invalid:*** A reference of any type is made to a storage location when the block-invalid bit for the location is 1.

- ***Store Protection:*** A store reference is made to a storage location, the program is being executed in application mode, I/O mode, or supervisor mode, and the store-protection bit for the location is 1.

- ***Execution Protection:*** An instruction-fetch reference is made to a storage location, the program is being executed in application mode, I/O mode, or supervisor mode, and the execution-protection bit for the location is 1.

An access exception for the block-invalid condition is indicated regardless of the mode in which the program is being executed. Conversely, an access exception for the store- or execution-protection condition is not indicated when the program is being executed in master mode (bit positions 38 and 39 of the current PSV contain the value 00).

When fetching of protected information is attempted, the information is not loaded into an addressable register or moved to another storage location. When part of an operand location is protected against storing and part is not, storing may be performed in the unprotected part. The contents of a protected location remain unchanged. An access exception for a partially inaccessible operand is not indicated when the operation can be completed without use of the inaccessible part of the operand.

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

For BRANCH or JUMP operations, the target address is tested for validity as part of the execution of the BRANCH or JUMP operation. If an access exception is detected for the first halfword associated with the target address, the IA in the stored PSV indicates the BRANCH or JUMP as the failing instruction.

An access exception is indicated only when the PCE attempts to execute the instruction with which the exception is associated. In particular, the exception is

not indicated when the PCE attempts to prefetch an instruction from an inaccessible location or otherwise detects the access exception, but a branching operation or an interruption changes the instruction sequence such that the instruction is not executed.

The PCE may indicate an access exception instead of an address exception (code 4) when it encounters an address-limit condition. More details are provided under "Address Exception" later in this chapter.

Execution of the instruction identified by the stored PSV is suppressed except for the following special cases: A MOVE, COMPARE LOGICAL, or LOAD QUADRANT operation is terminated and a STORE QUADRANT or floating-point LOAD or floating-point STORE is suspended when an access exception is detected.

Operation Exception (code 2)

An operation exception is indicated for the following conditions:

- ***Invalid Operation:***
  - An instruction is encountered with an invalid value in an operation-code field. The operation code may not be defined or the instruction with that operation code may be defined for a feature not installed. If the floating-point feature is installed, and the PCE encounters an FF or FS format instruction with an undefined operation code, a floating-point exception (code 7) is indicated, not an operation exception.
  - A 1 is encountered in any instruction bit position that is reserved and required to contain 0.
  - If R_3 designates the same register as R_1 or R_2 for a MOVE or COMPARE LOGICAL operation, the result is unpredictable. Depending on processor model, an operation exception may occur.

  Programming Note: Whenever the PCE encounters all 1's in the first halfword of an instruction, an operation exception is indicated. The PROGRAM EXCEPTION instruction is provided to cause an exception at a known location.

- ***Privileged Operation:*** A privileged instruction, other than an FS format instruction, is encountered that is not allowed by the program-mode field of the current PSV. Specifically, an operation exception for the privileged-operation condition is indicated in the following cases:
  - The PCE encounters a supervisor-privileged instruction when the program-mode field specifies I/O or application mode.
  - The PCE encounters an I/O-privileged instruction when the program-mode field specifies application mode.

The PCE may indicate an operation exception instead of an address exception (code 4) when it encounters an address-limit condition. Refer to "Address Exception" later in this chapter for more information.

In all cases, the instruction is suppressed.

Operation-exception conditions related to FF and FS format instructions are summarized later in this section under "Floating-Point Exception." A detailed description is in Chapter 5.

Separation Exception (code 3)

On processor models that implement separation protection, a separation exception is indicated only with dynamic address translation active. When the program causes a reference to main storage, the protection key associated with the active ACV is compared with the translation-lock-table entry of the corresponding block in the PCE address space. A separation exception is indicated when the protection key and translation-lock-table values are not identical and neither value is zero.

When a separation exception is indicated, the block index of the PCE address in error is placed in the EBI register associated with the active ACV.

For MOVE, COMPARE LOGICAL, LOAD QUADRANT, STORE QUADRANT, or floating-point LOAD or STORE instructions, operation is suspended when a separation exception is detected. For all other instructions, execution of the instruction identified by the stored PSV is suppressed.

Address Exception (code 4)

An address exception is indicated when the program attempts to refer to a main-storage location using a logical address that is not available to the program. A logical address that is not available to the program is referred to as invalid. The storage location associated with an invalid address is considered unavailable.

An address exception is indicated for the following conditions:

- ***Address Limit:*** The logical address is greater than the maximum logical address available to the program. The maximum address available to the program is 1 less than the value designated by the size field in the current ACV.

- ***Address Underflow:*** The magnitude of a negative displacement is greater than the base address or updated instruction address to which it is added during address generation.

When fetching of information from an unavailable location is attempted, the information is not loaded into an addressable register or moved to another storage location. Similarly, when storing to an unavailable location is attempted, the information is not stored, and the contents of the unavailable location remain unchanged. When part of an operand location is available to the program and part is not, fetching or storing may be performed using the available part. An address exception for a partially unavailable operand is not indicated when the operation can be completed without referring to the unavailable part of the operand.

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

For BRANCH or JUMP operations, the target address is tested for validity as part of the execution of the BRANCH or JUMP operation. If an address exception is detected for the first halfword associated with the target address, the IA in the stored PSV indicates the BRANCH or JUMP as the failing instruction.

An address exception is indicated only when the PCE attempts to execute the instruction with which the exception is associated. In particular, the exception is *not* indicated in the following cases:

- An invalid operand address is generated for an instruction that includes as part of its operation incrementing the operand address, but the instruction can be completed without referring to the unavailable location.

- The PCE attempts to prefetch an instruction from an unavailable location or otherwise detects the address exception, but a branching operation or an interruption changes the instruction sequence such that the instruction is not executed.

Depending on processor model:

- A specification exception may be indicated when an invalid operand address is obtained for a BRANCH or JUMP operation but the instruction can be completed without referring to the unavailable location.

- An address exception may *not* be detected when an address-limit condition exists. This condition can occur when a logical address is generated that exceeds a specific value; the value is specific for the particular processor model and is greater than the maximum address in the PCE address space. If a program exception is detected, it is indicated as either a specification exception (code 0), an access exception (code 1), or an address exception (code 4). If this condition is not detected, the operation is completed with unpredictable results.

- An address exception may *not* be indicated when concurrent address-limit and invalid operation conditions exist during the fetching of an instruction. If the instruction fetch is not the result of a BRANCH or JUMP taken operation, an operation exception (code 2) may instead be indicated.

Operation is suppressed except for the following special cases: A MOVE, COMPARE LOGICAL, or LOAD QUADRANT operation is terminated and a STORE QUADRANT or floating-point LOAD or floating-point STORE operation is suspended when an address exception is indicated.

Register-Indirect Exception (code 5)

A register-indirect exception is indicated during register-indirect operations for the following causes:

- The register-indirect addressing vector contains a 1 in a reserved bit position (bit positions 2-5).

- The program attempts to refer to a byte or halfword of an adjunct register that is not available. In particular, the exception is indicated when the register-indirect addressing vector designates the high-order half of the 32-bit adjunct register (bit positions 0 and 1 of the register-indirect addressing vector contain the value 11), and either:

- Only the two low-order byte locations of the adjunct registers are available.

- Only the three low-order byte locations of the adjunct registers are available and LOAD (halfword, register-indirect) or STORE (halfword, register-indirect) is encountered.

- Only the three low-order byte locations of the adjunct registers are available and LOAD (byte, register-indirect) or STORE (byte, register-indirect) is encountered designating the high-order byte of the register (bit position 15 of the addressing vector contains a 0).

In all cases, the operation is suppressed.

Fixed-Point-Overflow Exception (code 6)

A fixed-point overflow is indicated in fixed-point division when the divisor is 0 or the quotient exceeds 16 bits.

The operation is suppressed.

Floating-Point Exception (code 7)

A floating-point exception is indicated in floating-point operations when the floating-point feature is installed and the exception relates to any of the following conditions:

Floating-point operation
Floating-point privileged operation
Floating-point specification
Floating-point divide
Significance
Exponent overflow
Exponent underflow

When the floating-point feature is not installed, and the PCE encounters an FF or FS format instruction, an operation exception (code 2) is indicated. See "Program Exceptions" in Chapter 5 for a detailed discussion of exceptions associated with floating-point operations.

Handling of Multiple Program Exceptions

When a program-exception interruption occurs, the PSV indicates only one exception type. However, more than one program exception can result from an instruction sequence, as one exception can cause another one to occur. Except for the case described earlier in this chapter under "Address Exception" where a processor model may indicate an operation exception instead of an address exception, the program exception code identifies the exception that has the highest priority. Thus, the program exception code may not indicate the valid (antecedent) exception. In particular, a specification exception due to a PSV or ACV format error may not be indicated when a new PSV and ACV are introduced if the instruction fetch that uses them detects an exception that has a higher priority, such as an address, access, or operation exception.

The following lists program exceptions in priority order, with 1 being the highest. The letters indicate exceptions that cannot occur together. For example, a floating-point exception (5C) or fixed-point-overflow exception (5B) cannot occur when a register-indirect exception (5A) is occurring.

1. Address Exception (code 4)
2. Separation Exception (code 3)
3. Access Exception (code 1)
4. Operation Exception (code 2)

5A. Register-Indirect Exception (code 5)
5B. Fixed-Point-Overflow Exception (code 6)
5C. Floating-Point Exception (code 7)

6. Specification Exception (code 0)

Chapter 4. General Instructions

This chapter describes all of the unprivileged instructions that have general application. Within this class are instructions for fixed-point arithmetic, logical operations, and instruction sequencing control. Preceding the instruction descriptions in this chapter are descriptions of data formats, fixed-point numbers, and extended fixed-point numbers.

Data Formats

A general instruction treats data as being one of three types: signed fixed-point numbers, unsigned fixed-point numbers, or unstructured logical quantities.

Data resides in general registers or in storage, or is introduced from the instruction stream.

In storage-to-storage MOVE and COMPARE LOGICAL operations, 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 the compare instructions, overlapping does not affect execution of the operation. For the move instructions, 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 unit of data at a time, where the data unit is a byte or a halfword. All overlapping fields are considered valid to the PCE.

Fixed-Point Numbers

Fixed-point numbers are treated as signed or unsigned integers.

In an unsigned fixed-point number, all bits are used to express the absolute value of the number. When an unsigned fixed-point number participates in an operation such as addition or subtraction with a longer number, it is considered to be extended with high-order 0's to the length of the longer number.

For signed fixed-point numbers, the leftmost bit represents the sign, which is followed by the integer field. Positive numbers are represented in true binary notation with the sign bit set to 0. Negative numbers are represented in twos-complement binary notation with a 1 in the sign-bit position.

Specifically, a negative number is represented by the twos complement of the positive number. The twos complement of a number is obtained by inverting each bit of the number and adding a 1 in the low-order bit position of the inverted number.

This type of number representation can be considered the low-order portion of an infinitely long representation of the number. When the number is positive, all bits to the left of the most significant bit of the number are 0's. When the number is negative, all these bits are 1's. Therefore, when an operand must be extended with high-order bits, the expansion is achieved by setting the bits equal to the high-order (sign) bit of the operand.

The notation for signed fixed-point numbers does not include a negative 0. It has a number range in which the set of negative numbers is 1 larger than the set of

positive numbers. The maximum positive number consists of an all-1 integer field with a sign bit of 0, whereas the maximum negative number (the number with the greatest absolute value) consists of an all-0 integer field with a sign bit of 1. A value of 0 consists of all 0's including the sign bit.

The complement of the maximum negative number cannot be represented in the same number of bits. When an operation, such as a subtraction of the maximum negative number from 0, attempts to produce the complement of the maximum negative number, an overflow occurs. An overflow does not result, however, when the maximum negative number is complemented and the final result is within the representable range. An example of this case is subtraction of the maximum negative number from minus 1.

In discussions of signed fixed-point numbers in this publication, the expression "8-bit signed integer" denotes a 7-bit integer with a sign bit, and the expression "16-bit signed integer" denotes a 15-bit integer with a sign bit. In general, the expression "n-bit signed integer" denotes an integer of n-1 bits with a sign bit.

The range of integer values (I) covered by an n-bit signed fixed-point number is:

In the 8-bit format:
$-128 \leq I \leq 127$

In the 16-bit format:
$-32768 \leq I \leq 32767$

In the general n-bit format:
$-2^{(n-1)} \leq I \leq 2^{(n-1)} - 1$

In addition or subtraction of signed fixed-point numbers, a carry out of the integer field is retained as the sign bit of the result; a carry out of the sign-bit position is retained as bit 56 of the current PSV. (Bit 56 is the condition-indicator bit that represents result condition 0.) If the carry out of the integer field agrees with the carry out of the sign bit, no overflow occurs and the result is properly indicated; otherwise, an overflow condition is indicated. In either case, however, the setting of result condition 8, 4, or 2 (which indicates whether the result is 0, less than 0, or greater than 0) is determined by the rules of algebra from the magnitude and sign of the operands, except that condition 8 is always indicated when the stored result is all 0's, even if overflow occurred.

In some operations, such as comparison or subtraction, the result is achieved by the use of the ones complement of the number. The ones complement of a number is obtained by inverting each bit of the number.

Programming Notes:

- A signed fixed-point number may be considered to represent the sum of the integer part of the number, taken as a positive value, and the sign, representing a value of either 0 when the sign is 0, or the maximum negative number when it is 1.

- A 0 result with overflow occurs when two maximum negative numbers are added. For this particular situation, the rules of algebra (which call for a less-than-0 indication) are not used to determine the indicated result condition. Instead, a 0 sum (as well as overflow and carry) is indicated. The

0-sum indication is required when the operands represent intermediate portions of two extended fixed-point numbers (described in the following section) that are of opposite sign and equal magnitude. When two such numbers are added, it is necessary to indicate the intermediate sum as 0 so that the final sum can be properly indicated as 0.

- In arithmetic operations involving signed fixed-point operands, the operands may be considered by the program as unsigned positive integers. When doing so, the program should interpret the result conditions as follows:

  Result conditions after addition of unsigned integers:

  8 Sum is 0 if condition 0 is not indicated.
  4 Sum is not 0.
  2 Sum is not 0.
  1 No meaning is associated with condition 1.
  0 Carry (sum overflowed).

  Result conditions after comparison of unsigned integers:

  8 Operands are equal.
  4 Operands are not equal.
  2 Operands are not equal.
  1 No meaning is associated with condition 1.
  0 First operand is high if condition 8 is not indicated; otherwise, operands are equal.

  Result conditions after subtraction of unsigned integers:

  8 Difference is 0.
  4 Difference is not 0.
  2 Difference is not 0.
  1 No meaning is associated with condition 1.
  0 Carry (difference is properly represented).

Extended Fixed-Point Numbers

The IBM 8100 system provides the capability to process extended fixed-point numbers. The format of an extended fixed-point number is the same as a basic fixed-point number (see "Fixed-Point Numbers" earlier in this chapter), except that the length of an extended fixed-point number consists of n bits, where n is a multiple of 8.

For unsigned numbers, all n bits are used to express the absolute value of the number. For signed numbers, the leftmost bit represents the sign, which is followed by n-1 integer bits. For example, a "24-bit signed integer" denotes an extended fixed-point number with a high-order sign bit followed by a 23-bit integer field.

Addition and subtraction of extended fixed-point numbers is accomplished by processing the operands right to left in increments of 8 bits or 16 bits, whichever is most convenient. Where applicable, both increment sizes may be used. For example, addition of two 24-bit numbers can be achieved by first adding the rightmost 16 bits of the numbers, and then adding the remaining 8 bits of the numbers.

ADD WITH CARRY and SUBTRACT WITH CARRY are the operations used, along with ADD and SUBTRACT, to process extended fixed-point numbers. The rightmost 8 or 16 bits of the two numbers must be processed first, using ADD or SUBTRACT. The remaining portion of the two numbers must be processed right to left using the corresponding "with carry" instructions.

A carry out of the integer field at any intermediate step is retained as the sign bit of the extended fixed-point result. A carry out of the sign-bit position is retained in bit 56 of the current PSV (the condition-indicator bit that represents result condition 0), provided the program does not execute a condition-setting instruction between the addition or subtraction of adjacent portions of the two numbers. The carry retained in PSV-bit 56 participates in ADD WITH CARRY and SUBTRACT WITH CARRY as a low-order 0 or 1 (refer to the individual instruction descriptions). If the carry out of the integer field of the extended fixed-point result agrees with the carry out of the sign bit, no overflow occurs and the extended result is satisfactory; otherwise, an overflow condition is indicated.

At the completion of each intermediate step employing a "with carry" operation, a result of 0 is indicated as such only if the results of all previous intermediate steps were 0. That is, result condition 8 can be indicated (reflecting a result of 0) only if it was indicated at the beginning of the "with carry" operation. Thus, a non-0 extended fixed-point result is properly indicated, even if one or more steps of the extended fixed-point calculation, including the last, yielded intermediate results of 0.

When addition or subtraction of two extended fixed-point numbers is complete, the extended fixed-point result is in its proper form and the resulting conditions reflect the outcome as if the extended numbers were processed with a single arithmetic operation. In particular, the setting of result condition 8, 4, or 2 which indicates whether the extended result is 0, less than 0, or greater than 0) is determined by the rules of algebra from the sign and magnitude of the extended fixed-point operands, except that condition 8 is indicated when the stored extended result is all 0's, even if overflow occurred.

Processing extended fixed-point numbers of unequal length is accomplished in a similar manner by specifying the shorter number as the second operand. A provision of certain ADD WITH CARRY and SUBTRACT WITH CARRY instructions allows for the specification of an implied 0 value for the second operand. When an implied 0 is specified for the "with carry" instructions, the shorter number is effectively extended with high-order 0's. The shorter number, therefore, is always treated as a positive extended fixed-point number.

Two such extended fixed-point numbers are processed by first adding or subtracting the short number and the corresponding right-hand portion of the long number, and then completing the arithmetic using an implied 0 as the second operand of the "with carry" instructions. For example, addition of a positive 8-bit immediate value to a 16-bit signed integer can be accomplished with ADD (byte, register-immediate), followed by ADD WITH CARRY (byte, register) with an implied 0 specified as the second operand. Subtraction of a positive 16-bit integer from a 32-bit signed integer may be achieved using SUBTRACT (halfword, register), followed by SUBTRACT WITH CARRY (halfword, register, extended) with 0 as the implied second operand.

Besides addition and subtraction, COMPARE WITH CARRY is provided for the comparison of two extended fixed-point numbers. Its operation is similar to SUBTRACT WITH CARRY except that the operands remain unchanged. The COMPARE WITH CARRY instruction is normally used for the comparison of the high-order 16 bits of two 32-bit fixed-point numbers.

Programming Notes:

- During the addition or subtraction of two extended fixed-point numbers whose sum or difference is 0, the result of every intermediate operation is 0.
- Addition of two fixed-point numbers of unequal length, in which the shorter number is negative, can be accomplished using a SUBTRACT WITH CARRY instruction for each intermediate operation that specifies an implied 0 as the second operand. Similarly, subtraction of two such numbers is achieved using ADD WITH CARRY for each operation in which an implied 0 is used as the second operand.
- Result conditions are indicated at the completion of each intermediate step of an extended-fixed-point calculation. However, because the indicated conditions reflect only that portion of the extended result that was processed, the indicated conditions reflect the full extended result only after the final step of the calculation is complete.

Instruction Descriptions

A detailed description of each general instruction is given in this chapter under "Instructions." Instructions for floating-point operations are described in Chapter 5. Instructions that indirectly access the principal and adjunct register groups are described in Chapter 6. Instructions for accessing the translation table are described in Chapter 7. Input/output instructions are described in Chapter 8. Instructions that read or modify system status information are described in Chapter 9.

Note: *Appendix A contains three lists that summarize the instruction descriptions. These lists arrange the instructions by name, mnemonic, and by instruction type.*

The same general organization is used for the detailed descriptions of all instructions. The description of each instruction includes:

- The instruction name
- An assembler language statement for the instruction
- A diagram of the machine instruction format
- A symbolic expression of the operation
- A detailed description of the operation
- Result conditions that may be indicated
- The program exceptions that cause an interruption

Instruction Name

The instruction name consists of the name of the operation and, where applicable, certain qualifiers of the operation. The operation name is in all uppercase. The operation qualifiers, if any, follow in parentheses; they may include a designation of the data-unit size, an indication of the type of operation (such as register-to-register or storage-to-storage), or other information needed to differentiate each instruction.

Assembler Language Statement

The assembler language statement contains the instruction mnemonic and symbolic operand specification defined for the IBM 8100 DPPX Assembler licensed program. One mnemonic is defined for each instruction name and, in general, symbolically represents the machine instruction's operation code. For operations involving byte data units, two or more machine instruction operation codes may be associated with one mnemonic. For these instructions, the operand specification includes the distinguishing operation-code information.

The operand specification is represented as symbols that denote the type of specification. For example, in the operand specification for LOAD (byte), "rpb,db16s(ra)", the symbol "rpb" denotes the specification of a byte-operand location in a primary general register, the symbol "db16s" denotes the specification of a displacement value that is expressed in the machine instruction as a 16-bit signed value, and the symbol "ra" denotes the specification of a general register containing a base address.

Operands are specified in the assembler statement in left-to-right order, beginning with the first-operand specification. Each specification is delimited from an adjacent specification with a comma. In LOAD (byte), for example, the first-operand specification is "rpb" and the second is "db16s(ra)".

Additional information pertaining to the operand specification that is defined for the IBM 8100 assembler language is in Appendix B. A detailed definition of the 8100 assembler language is in *IBM 8100 DPPX Assembler Programming: Language Reference and Guide,* SC27-0412.

Machine Instruction Format

In the format illustration for each individual instruction description, the op-code fields show the operation code in hexadecimal representation. The hexadecimal representation uses one graphic for a 4-bit code, and therefore two graphics for an 8-bit code. The graphics 0 through 9 are used for the codes 0000-1001; the graphics A through F are used for codes 1010-1111. Fields with less than 4 bits are shown in binary representation.

The remaining fields in the format illustration pertain to the designation of operands. Operand fields are designated with symbols consisting of two characters: a letter and a subscript number. The letter indicates the type of information in the operand field. For example, "I" denotes immediate data, "R" indicates a general-register designation, and "D" identifies a displacement value (see "Instructions" in Chapter 3. The subscript number denotes the operand to which the field applies. Subscript 1 denotes the first operand of an instruction, and subscript 2 denotes the second. In several instructions, a third operand is designated, which is denoted with subscript 3.

For most instructions, the operands in the machine instruction format are designated in left-to-right order beginning with the first operand, which corresponds to their order in the assembler language statement. However, a few machine instructions do not adhere to this general rule. For these instructions, the designation of the second operand precedes that of the first, thus reversing the order from that of the corresponding assembler language statement. These exceptions are identified in the programming notes for the applicable instructions.

Note: *In certain cases, the operand specification for the 8100 assembler language is not directly represented in the machine instruction's operand field. General registers, for example, are specified with the numbers 0, 2, 4, ..., 30 for the assembler language. The operand field in the machine instruction designates the corresponding general register as 0 through 7, in either the primary or the secondary register set. The correlation between the 8100 assembler language operand specification and the machine instruction's operand designation is provided in Appendix B.*

Operation

The operation of each instruction is depicted with a symbolic expression. The symbolic expression is intended only as a quick-reference reminder of an instruction's operation. The description of the instruction following the symbolic expression provides the detailed definition of the operation.

The symbolic expression is presented as a sequence of statements designating the operands used and the results produced. The syntax of the symbolic statements is similar to that of high-level programming languages. The order of the statements corresponds to the conceptual order in which the PCE executes the steps of the operation. Conditional execution of steps is indicated by statements of the form If...Then...Else...; unconditional changes in the sequence are indicated by statements using Go To....

Figure 4-1 lists the symbols used in the instruction format illustration and in the symbolic expression of the operation. In the individual instruction descriptions, a subscript number following a symbol denotes the operand to which the symbol applies. Some abbreviations used in the symbolic expressions of operations may not be defined in Figure 4-1. Such abbreviations are defined in the Glossary.

Description

The prose description provides the detailed definition of the instruction. The specific operation of the instruction and any restrictions that may apply to the instruction's format and operand designation are described.

The individual instruction descriptions do not provide a specification of every function related to the execution of the instruction. For example, information pertaining to the format of instruction fields (such as the R field or D field), the format of data (such as fixed-point or floating-point numbers), or the addressing of data units on integral storage boundaries, is found in this manual in the section dealing with the specific function, and is not normally included with each instruction description.

Symbol	Meaning
(n)	Contents of general register designated by n
<—	"is replaced by"
<==	"is determined by"
+	Addition
-	Subtraction
x	Multiplication
/	Division
MOD	Modulo division
||	Concatenation
•	Boolean AND
v	Boolean OR
¬	Boolean inverse (1's complement)
≠	Boolean exclusive OR
:	Logical comparison
=	Equal to
{n+a}	General register designated by the sum of n and a
≠	Not equal to
Bn	Instruction field designating a base register (operand number n)
C	Condition indicator for "carry" in current PSV (bit 56)
Dn	Displacement field of instruction (operand number n)
Fn	Instruction field designating a floating-point register (operand number n)
In	Immediate field of instruction (operand number n)
IA	Updated instruction address in current PSV
IOD[@]	I/O device designated by PIO address @
Mn	Mask field of instruction (operand number n)
MS[@]	Contents of main-storage location addressed by @
n<a>	Bit a of quantity identified by n
n<a..b>	Bits a through b of quantity identified by n
NSI	Next sequential instruction
PGRn	Implied primary general register number n
Qn	Instruction field designating register quadrant (operand number n)
rn	Field of instruction designating a general-register byte operand (operand n)
Rn	Field of instruction designating a general register (operand number n)
RG[@]	Contents of register-group location addressed by @
RQ<q>	Contents of general-register quadrant designated by q
TEMP	Temporary working register within PCE
TL[@]	Contents of translation-lock-table entry addressed by @
TT[@]	Contents of translation-table entry addressed by @

Figure 4-1. Symbols Used in Instruction Descriptions

Result Conditions

For each instruction, the possible result conditions are listed. If no result conditions are listed, the condition indicators in the PSV remain unchanged.

For some operations, only certain result conditions, such as 8, 4, and 2, may be indicated. In this case, the notation "--" is used for result conditions 1 and 0 to denote that no meaning is assigned for these conditions. This notation means that at the completion of the operation, the condition is not indicated. Note that a test of such a condition with a branching operation results in no branch.

Program Exceptions

The possible program-exceptions that can be detected as part of the execution of the instruction are listed. These include exceptions in format, operand designation, or results. Not listed are exceptions, such as specification due to an invalid PSV/ACV format, or operation due to invalid operation code, as they are not directly related to the operation of any defined instruction.

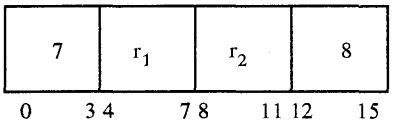
Each exception listed contains the program-exception name corresponding to a distinct program-exception code. When the listed exception applies to an operand or the designation of an operand, the particular operand (first or second) is indicated. When the exception identified by the program-exception code can be caused by several conditions, the applicable conditions are indicated.

Instructions

The general instructions and their mnemonics, formats, and operation codes follow.

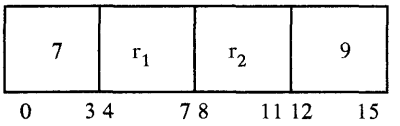
ADD (byte, register)

AR rpb,rpb



RR Format
(primary-register-set operands)

AR rsb,rsb



RR Format
(secondary-register-set operands)

Operation

If $r_2 \neq 0000$
  Then $(r_1) \leftarrow (r_1) + (r_2)$
  Else $(r_1) \leftarrow (r_1) + 00000000$

Description

The second-operand byte is added to the first-operand byte, and the sum is placed in the first-operand location. Addition is performed by adding all 8 bits of both operands. The two operands are considered to be signed fixed-point numbers.

An implied second-operand byte of all 0's is used in place of the register contents when the r_2 field of the instruction is all 0's.

The operands are located in the same register set, designated with bit positions 12-15 of the instruction: hexadecimal "8" designates the primary set, and hexadecimal "9" designates the secondary set.

Result Conditions

8 Sum is 0.
4 Sum is less than 0.
2 Sum is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions

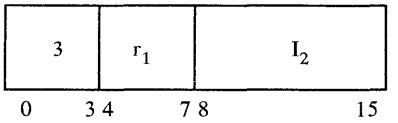
None

Programming Note

The byte in bit positions 16-23 of register 0 in the primary and secondary register sets can be designated only as the first operand; it cannot be designated as the the second operand.

ADD (byte, register-immediate)

ARI rpb,i8s



RI Format

Operation

$(r_1) \leftarrow (r_1) + I_2$

Description

The byte of immediate data, I_2, is added to the first-operand byte, and the sum is placed in the first-operand location. Addition is performed by adding all 8 bits of both operands. The two operands are considered to be signed fixed-point numbers.

The first operand is located in the primary register set.

Result Conditions

8 Sum is 0.
4 Sum is less than 0.
2 Sum is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions

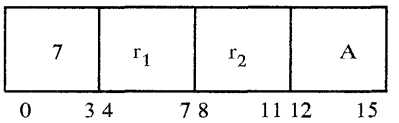
None

Programming Note

The immediate operand can range in value from -128 to 127. Addition of a negative immediate value is equivalent to subtraction of the corresponding positive immediate value of the same magnitude.

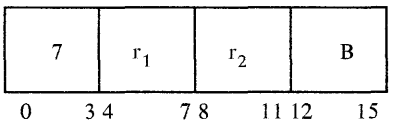
ADD WITH CARRY (byte, register)

AYR rpb,rpb



RR Format
(primary-register-set operands)

AR rsb,rsb



RR Format
(secondary-register-set operands)

Operation

If $r_2 \neq 0000$
  Then $(r_1) \leftarrow (r_1) + (r_2) + C$
  Else $(r_1) \leftarrow (r_1) + 00000000 + C$

Description

The second-operand byte and the value of bit 56 (the carry-condition indicator) in the current PSV are added to the first-operand byte, and the sum is placed in

the first-operand location. Addition is performed by adding all 8 bits of the second operand and a low-order 0 or 1, taken from bit 56 (C) in the current PSV, to all 8 bits of the first operand. The two operands are considered to be signed fixed-point numbers.

An implied second-operand byte of all 0's is used in place of the register contents when the r_2 field of the instruction is all 0's.

The operands are located in the same register set, designated with bit positions 12-15 of the instruction: hexadecimal "A" designates the primary set, and hexadecimal "B" designates the secondary set.

Result Conditions
8 Extended sum is 0.
4 Extended sum is less than 0.
2 Extended sum is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions
None

Programming Notes
The ADD WITH CARRY instructions are provided for addition of extended fixed-point numbers. A carry from any ADD or ADD WITH CARRY instruction is accounted for by executing a subsequent ADD WITH CARRY instruction without executing an intervening instruction that changes the indicated result conditions.

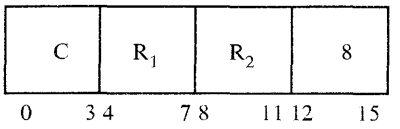
ADD WITH CARRY (byte, register) can be used to propagate only the carry, if any, from the previous ADD by specifying an implied second-operand byte of all 0's.

Result condition 8 can be indicated (reflecting a result of 0) only if it was indicated at the beginning of the operation.

The byte in bit positions 16-23 of register 0 in the primary and secondary register sets can be designated only as the first operand; it cannot be designated as the second operand.

ADD (halfword, register)

AHR rh,rh



RR Format

Operation
$(R_1<16..31>) \leftarrow (R_1<16..31>) + (R_2<16..31>)$

Description
The second-operand halfword is added to the first-operand halfword, and the sum is placed in the first-operand location. Addition is performed by adding all 16 bits of both operands. The two operands are considered to be signed fixed-point numbers.

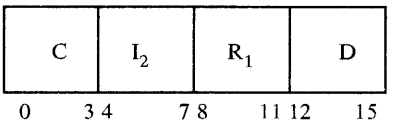
The operands occupy the low-order 16 bits of the registers designated by the R_1 and R_2 fields.

Result Conditions
8 Sum is 0.
4 Sum is less than 0.
2 Sum is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions
None

ADD (halfword, register-immediate)

AHRI rh,i4



RR Format

Operation
$(R_1<16..31>)$ <— $(R_1<16..31>) + 000000000000 \mid\mid I_2$

Description
The 4 bits of immediate data, I_2, are added to the first-operand halfword and the sum is placed in the first-operand location. The immediate operand is treated as an unsigned 4-bit positive binary integer. Addition is considered to be performed by first expanding the immediate operand to 16 bits with 12 high-order 0's. Then, all 16 bits of the expanded immediate operand are added to all 16 bits of the first operand. The first operand is considered to be a signed fixed-point number.

The first operand occupies the low-order 16 bits of the register designated by the R_1 field.

Result Conditions
8 Sum is 0.
4 Sum is less than 0.
2 Sum is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

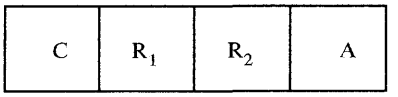
Program Exceptions
None

Programming Notes
The second operand can range in value from 0 to 15, inclusive.

Bit positions 4-7 and 8-11 of the instruction format contain the I_2 and R_1 fields, respectively. This is reversed from the normal left-to-right order for the RR instruction format.

ADD WITH CARRY (halfword, register)

AYHR rh,rh



RR Format

Operation

$(R_1<16..31>) \leftarrow (R_1<16..31>) + (R_2<16..31>) + C$

Description

The second-operand halfword and the value of bit 56 (the carry-condition indicator) in the current PSV are added to the first-operand halfword, and the sum is placed in the first-operand location. Addition is performed by adding all 16 bits of the second operand and a low-order 0 or 1, taken from bit 56 (C) in the current PSV, to all 16 bits of the first operand. The two operands are considered to be signed fixed-point numbers.

The operands occupy the low-order 16 bits of the registers designated by the R_1 and R_2 fields.

Result Conditions

8 Extended sum is 0.
4 Extended sum is less than 0.
2 Extended sum is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions

None

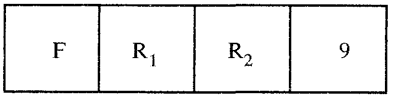
Programming Notes

The ADD WITH CARRY instructions are provided for addition of extended fixed-point numbers. A carry from any ADD or ADD WITH CARRY instruction is accounted for by executing a subsequent ADD WITH CARRY instruction without executing an intervening instruction that changes the indicated result conditions.

Result condition 8 can be indicated (reflecting a result of 0) only if it was indicated at the beginning of the operation.

ADD WITH CARRY (halfword, register, extended)

AYHRE ruh,ruh



RR Format

Operation

If $R_2 \neq 0000$
  Then $(R_1<0..15>) \leftarrow (R_1<0..15>) + (R_2<0..15>) + C$
  Else $(R_1<0..15>) \leftarrow (R_1<0..15>) + 0000000000000000 + C$

Description
The second-operand halfword and the value of bit 56 (the carry-condition indicator) in the current PSV are added to the first-operand halfword, and the sum is placed in the first-operand location. Addition is performed by adding all 16 bits of the second operand and a low-order 0 or 1, taken from bit 56 (C) in the current PSV, to all 16 bits of the first operand. The two operands are considered to be signed fixed-point numbers.

An implied second-operand halfword of all 0's is used in place of the register contents when the R_2 field of the instruction is all 0's.

The operands occupy the high-order 16 bits of the registers designated by the R_1 and R_2 fields.

Result Conditions
8 Extended sum is 0.
4 Extended sum is less than 0.
2 Extended sum is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions
None

Programming Notes
The ADD WITH CARRY instructions are provided for addition of extended fixed-point numbers. A carry from any ADD or ADD WITH CARRY instruction is accounted for by executing a subsequent ADD WITH CARRY instruction without executing an intervening instruction that changes the indicated result conditions.

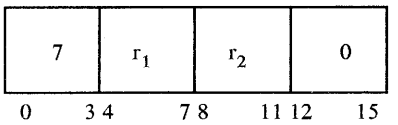
ADD WITH CARRY (halfword, register, extended) can be used following an ADD (halfword, register) instruction to perform word register-to-register addition. This instruction can also be used to propagate only the carry, if any, from the previous ADD by specifying an implied second-operand halfword of all 0's.

Result condition 8 can be indicated (reflecting a result of 0) only if it was indicated at the beginning of the operation.

The halfword in bit positions 0-15 of register 0 in the primary register set can be designated only as the first operand; it cannot be designated as the second operand.

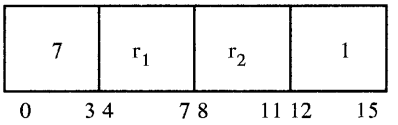
AND (byte, register)

NR rpb,rpb



RR Format
(primary-register-set operands)

NR rsb,rsb



RR Format
(secondary-register-set operands)

Operation

If $r_2 \neq 0000$
  Then $(r_1) \leftarrow (r_1) \bullet (r_2)$
  Else $(r_1) \leftarrow (r_1) \bullet 00000000$

Description

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

The operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit. A bit position in the result is set to 1 if the corresponding bit positions in both operands contain a 1; otherwise, the result bit is set to 0.

An implied second-operand byte of all 0's is used in place of the register contents when the r_2 field of the instruction is all 0's.

The operands are located in the same register set, designated with bit positions 12-15 of the instruction: hexadecimal "0" designates the primary set, and hexadecimal "1" designates the secondary set.

Result Conditions

8 Result is all 0's.
4 Result is all 1's.
2 Result is mixed 0's and 1's.
1 --
0 --

Program Exceptions

None

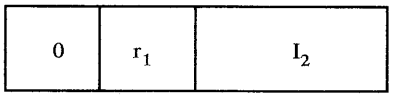
Programming Notes

This instruction may be used to set a bit to 0.

The byte in bit positions 16-23 of register 0 in the primary and secondary register sets can be designated only as the first operand; it cannot be designated as the second operand.

AND (byte, register-immediate)

NRI rpb,i8s



RI Format

Operation
(r_1) <— $(r_1) \bullet I_2$

Description
The AND of the first-operand byte and the byte of immediate data is placed in the first-operand location.

The operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit. A bit position in the result is set to 1 if the corresponding bit positions in both operands contain a 1; otherwise, the result bit is set to 0.

The first operand is located in the primary register set.

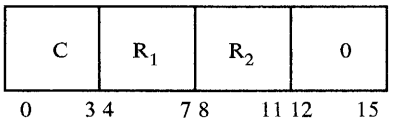
Result Conditions
8 Result is all 0's.
4 Result is all 1's.
2 Result is mixed 0's and 1's.
1 --
0 --

Program Exceptions
None

Programming Note
This instruction may be used to set a bit to 0.

AND (halfword, register)

NHR rh,rh



RR Format

Operation
$(R_1<16..31>)$ <— $(R_1<16..31>) \bullet (R_2<16..31>)$

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

The operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit. A bit position in the result is set to 1 if the corresponding bit positions in both operands contain a 1; otherwise, the result bit is set to 0.

The operands occupy the low-order 16 bits of the registers specified by the R_1 and R_2 fields.

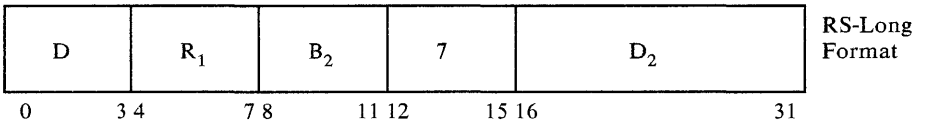
Result Conditions
8 Result is all 0's.
4 Result is all 1's.
2 Result is mixed 0's and 1's.
1 --
0 --

Program Exceptions
None

Programming Note
This instruction may be used to set a bit to 0.

BRANCH AND LINK

BAL ra,dh16s(ra)



Operation
TEMP1 <— D_2
TEMP1 <— TEMP1 rotated right 1 position
If $B_2 \neq 0000$
  Then TEMP2 <— (B_2) + TEMP1 x 2
  Else TEMP2 <— IA + TEMP1 x 2
(R_1) <— IA
IA <— TEMP2

Description
The updated instruction address in the current PSV is loaded as a link address in the general register designated by the R_1 field. Subsequently, the instruction address is replaced by the branch address. The second-operand address is used as the branch address.

When the B_2 field contains all 0's, the branch address is computed using the updated instruction address in place of the contents of primary general register 0.

The branch address is computed before the link address is loaded.

When a branch occurs, the branch address is tested for validity as part of the instruction execution. If a specification, access, separation, or address exception is detected for the first halfword associated with the branch address, the IA in the stored PSV indicates the current instruction as the failing operation.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, execution protection)
Separation (operand 2)
Address (operand 2: all)

Programming Notes
A jump-type address (an offset from the updated instruction address) is designated by specifying B_2 as primary register 0.

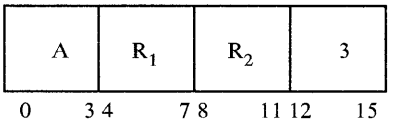
Specifying the same general register with R_1 and B+$sub2. does not alter the operation of the instruction.

Relative to the base address, the branch range, in bytes, covered by the displacement (D_2) is -65536 ≤ D_2 x 2 ≤ 65534. Note that the displacement can be specified only in terms of an even number of bytes.

The sign bit for the D_2 field is located in bit position 31 of this instruction. The formation of the branch address is described under "Address Generation" in Chapter 3, "Program Execution."

BRANCH AND LINK (register)

BALR ra,ra



RR Format

Operation
TEMP <— (R_2)
(R_1) <— IA
If $R_2 \neq 0000$
  Then IA <— TEMP
  Else NSI

Description
The updated instruction address in the current PSV is loaded as a link address in the general register designated by the R_1 field. Subsequently, the instruction address is replaced by the branch address.

The contents of the general register designated by the R_2 field are used as the branch address. The branch address is temporarily saved before the link address is loaded. However, when the R_2 field contains all 0's, the operation is performed without branching.

When a branch occurs, the branch address is tested for validity as part of the instruction execution. If a specification, access, separation, or address exception is detected for the first halfword associated with the branch address, the IA in the stored PSV indicates the current instruction as the failing operation.

Result Conditions
The conditions remain unchanged.

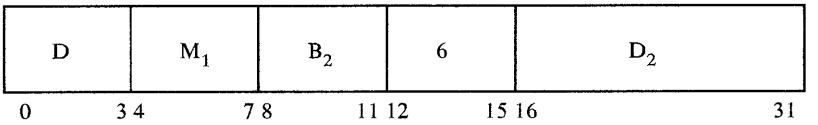
Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, execution protection)
Separation (operand 2)
Address (operand 2: address limit)

Programming Notes
Specifying the same general register with R_1 and R_2 does not alter the operation of the instruction.

When R_2 is specified as primary register 0, the link address is loaded without change. Thus, the instruction can be used to establish the updated instruction address as a base address.

BRANCH ON CONDITION

BC m4,dh16s(ra)



Operation
If M_1 specifies an indicated result condition

Then TEMP <— D_2
  TEMP <— TEMP rotated right 1 position
  If $B_2 \neq 0000$
    Then IA <— (B_2) + TEMP x 2
    Else IA <— IA + TEMP x 2
Else NSI

Description
The updated instruction address in the current PSV is replaced by the branch address if any result condition designated by the M_1 field is indicated in the current PSV. Otherwise, normal instruction sequencing proceeds with the updated instruction address. The second-operand address is used as the branch address.

When the B_2 field contains all 0's, then branch address is computed using the updated instruction address in place of the contents of primary general register 0.

The M_1 field is used as a 4-bit mask. The 4 bits of the mask correspond, left to right, with the four result conditions 8, 4, 2, and 1. A 1-bit in the mask specifies that the corresponding condition is to be tested. A mask containing two or more 1's tests all corresponding conditions. A mask of all 0's specifies a test for result condition 0.

M_1 Field	Mask Value	Result Condition Tested
1000	8	8
0100	4	4
0010	2	2
0001	1	1
0000	0	0

The branch is successful whenever any tested result condition is indicated in the current PSV. The branch is not taken if none of the tested conditions are indicated.

When a branch occurs, the branch address is tested for validity as part of the instruction execution. If a specification, access, separation, or address exception is detected for the first halfword associated with the branch address, the IA in the stored PSV indicates the current instruction as the failing operation.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, execution protection)
Separation (operand 2)
Address (operand 2: all)

Programming Notes
A jump-type address (an offset from the updated instruction address) is designated by specifying B_2 as primary register 0.

A branch can be made on more than one result condition, except condition 0, by specifying the pertinent conditions in the mask as the sum of their corresponding mask values. A mask value of 12 (binary 1100), for example, specifies that a branch is to be made if either result condition 8 or 4 is indicated.

An unconditional branch is made by specifying a mask of 14 or 15.

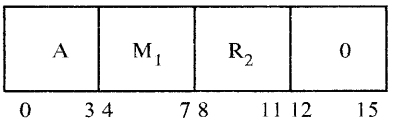
Result condition 8, 4, or 2 can be tested for its absence by specifying the other two conditions in the mask. A mask of 12, for example, will cause a branch to be executed if result condition 2 is not indicated. Result conditions 1 and 0 are each indicated independently, and the mask only provides for testing whether the corresponding condition is indicated.

Relative to the base address, the branch range, in bytes, covered by the displacement (D_2) is -65536 * D_2 x 2 * 65534. Note that the displacement can be specified only in terms of an even number of bytes.

The sign bit for the D_2 field is located in bit position 31 of this instruction. The formation of the branch address is described under "Address Generation" in Chapter 3, "Program Execution."

BRANCH ON CONDITION (register)

BCR m4,ra



RR Format

Operation
If M_1 specifies an indicated result condition

Then IA <— (R_2)
Else NSI

Description
The updated instruction address in the current PSV is replaced by the branch address if any result condition designated by the M_1 field is indicated in the current PSV. Otherwise, normal instruction sequencing proceeds with the updated instruction address. The contents of the general register designated by the R_2 field are used as the branch address.

The M_1 field is used as a 4-bit mask. The 4 bits of the mask correspond, left to right, with the four result conditions 8, 4, 2, and 1. A 1-bit in the mask specifies that the corresponding condition is to be tested. A mask containing two or more 1's tests all corresponding conditions. A mask of all 0's specifies a test for result condition 0.

M_1 Field	Mask Value	Result Condition Tested
1000	8	8
0100	4	4
0010	2	2
0001	1	1
0000	0	0

The branch is successful whenever any tested result condition is indicated in the current PSV. The branch is not taken if none of the tested conditions are indicated.

When a branch occurs, the branch address is tested for validity as part of the instruction execution. If a specification, access, separation, or address exception is detected for the first halfword associated with the branch address, the IA in the stored PSV indicates the current instruction as the failing operation.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, execution protection)
Separation (operand 2)
Address (operand 2: address limit)

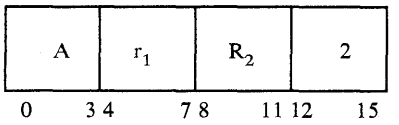
Programming Notes
A branch can be made on more than one result condition, except condition 0, by specifying the pertinent conditions in the mask as the sum of their corresponding mask values. A mask value of 12 (binary 1100), for example, specifies that a branch is to be made if either result condition 8 or 4 is indicated.

An unconditional branch is made by specifying a mask of 14 or 15.

Result condition 8, 4, or 2 can be tested for its absence by specifying the other two conditions in the mask. A mask of 12, for example, will cause a branch to be executed if result condition 2 is not indicated. Result conditions 1 and 0 are each indicated independently, and the mask only provides for testing whether the corresponding condition is indicated.

BRANCH ON COUNT (byte, register)

BCTR rpb,ra



RR Format

Operation

TEMP <— (R_2)
(r_1) <— (r_1) - 1
If (r_1) ≠ 00000000
  Then IA <— TEMP
  Else NSI

Description

The first-operand byte is a count that is algebraically reduced by 1. When the resulting count is 0, normal instruction sequencing proceeds with the updated instruction address. When the resulting count is not 0, the instruction address in the current PSV is replaced by the branch address.

The contents of the general register designated by the R_2 field are used as the branch address. The branch address is temporarily saved before the first operand is reduced.

The counting operation is performed by treating the first operand as an unsigned 8-bit positive binary integer, from which a low-order 1 is subtracted. The subtraction of 1 from an initial count of 0 yields a count of 255.

The first operand is located in the primary register set.

When a branch occurs, the branch address is tested for validity as part of the instruction execution. If a specification, access, separation, or address exception is detected for the first halfword associated with the branch address, the IA in the stored PSV indicates the current instruction as the failing operation.

Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)

Specification (operand 2: real address)
Access (operand 2: block invalid, execution protection)
Separation (operand 2)
Address (operand 2: address limit)

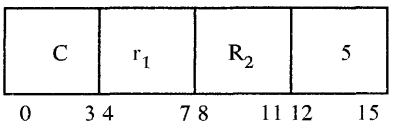
Programming Notes

An initial count of 1 results in 0, and no branching takes place; an initial count of 0 results in 255 and branching occurs; an initial count of 255 results in 254 and branching occurs; and so on. In a loop, branching takes place each time the instruction is executed until the result is again 0. The maximum loop count of 256 is attained by using an initial count of 0.

Designating the first operand in the same general register as that specified by R_2 does not alter the operation of the instruction.

BRANCH ON INDEX (byte)

BNX rpb,ra



RR Format

Operation

$IA<16..31> \leftarrow MS[(R_2) + (r_1) \times 2]$

Description

The low-order 16 bits of the instruction address in the current PSV are replaced by the halfword from the main storage location designated by the branch-table address and the index. The high-order 16 bits of the instruction address remain unchanged.

The contents of the general register designated by the R_2 field are used as the branch table address. The byte operand designated by the r_1 field is used as the index and is treated as an unsigned 8-bit positive binary integer.

The main storage address is considered to be formed by multiplying the index by two, expanding the result to 32 bits with high-order 0's, and adding the expanded index to the branch-table address.

Normal instruction-address updating is suppressed. That is, the high-order 16 bits of the instruction address used to refer to this instruction remain unchanged at the completion of execution.

The first operand (the index) is located in the primary register set.

When a branch occurs, the branch address is tested for validity as part of the instruction execution. If a specification, access, separation, or address exception is detected for the first halfword associated with the branch address, the IA in the stored PSV indicates the current instruction as the failing operation.

Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)

Specification (operand 2: real address; branch-table entry: real address)
Access (operand 2: block invalid; branch-table entry: block invalid, execution protection)
Separation (operand 2; branch-table entry)
Address (operand 2: address limit; branch-table entry: address limit)

Programming Notes

BRANCH ON INDEX provides an unconditional n-way branch, where "n" represents the index.

The branch table designated by the branch-table address must be aligned on a halfword boundary. A maximum of 256 halfword entries may be contained in the branch table.

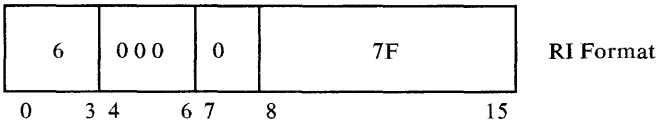
The branch-table entries are limited to 16 bits in order to conserve the amount of main storage required for the branch table. Each entry, therefore, contains only the low-order 16 bits of a storage address. Because only the low-order 16 bits of the instruction address are replaced with the table entry, and the high-order 16 bits remain unchanged, the range of branch addresses (A) that may be designated with branch-table entries is limited to:

$$M \times 2^{16} \leq A \leq (M+1) \times 2^{16}-1$$

where M denotes the value in the high-order 16 bits of the instruction address. That is, all branch addresses, of which only the low-order 16 bits are represented in the table entries, should designate locations within the set of 65,536 consecutive byte locations that (1) includes the location of the BRANCH ON INDEX instruction, and (2) begins at an address that is an integral multiple of 65536 (including 0).

CALL PSV

KI 0,127



Operation

```
Current-PSV<40..47>    <—    00000000
LPL    <—    CPL
If current-PSV = Primary
  Then Store primary PSV
    PAV<CPL>    <—  1
    Load secondary PSV and ACV
  Else Store secondary PSV
    PAV<CPL>    <—  0
    Load primary PSV and ACV
```

Description

The current PSV is stored into the register locations from which it was originally loaded, and the new PSV and ACV are loaded from the corresponding (dual) register locations for the current priority level.

The bit position, in the program activation vector (PAV), associated with the current priority level is set to correspond to the new PSV. The PAV bit is set to 1 when the secondary PSV is loaded; it is set to 0 when the primary PSV is loaded.

The 8-bit program-information-code field (bits 40-47) in the stored PSV is set to all 0's as part of the operation. The number of the last priority level is set equal to the number of the current priority level.

Bit positions 4-6 of the instruction are reserved and should contain all 0's. Otherwise, the 8-bit program-information-code field in the stored PSV is unpredictable.

Bit position 7 of the instruction is used as an extension to the operation code; the bit distinguishes this instruction from INPUT/OUTPUT (byte, immediate).

Result Conditions
The condition indicators in the stored PSV remain unchanged.

Program Exceptions
None

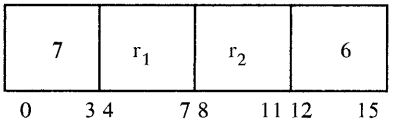
Programming Notes
After the new PSV is loaded, the result conditions are indicated as specified by the condition indicators in the new PSV.

CALL PSV is valid in all program modes. It can be used by a supervisory program to dispatch an application program, and it can be used by an application program to call the supervisor.

The high-order bit of the program information code (bit 40) in the stored PSV distinguishes whether the PSV was stored due to CALL PSV (bit 40 is 0) or a program-exception interruption (bit 40 is 1).

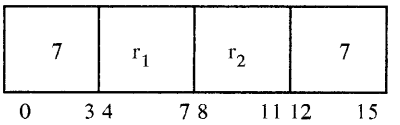
COMPARE (byte, register)

CR rpb,rpb



RR Format
(primary-register-set operands)

AR rsb,rsb



RR Format
(secondary-register-set operands)

Operation
If $r_2 \neq 0000$
  Then Result-Conditions <== $(r_1) + \neg(r_2) + 1$
  Else Result-Conditions <== $(r_1) + \neg 00000000 + 1$

Description
The first operand is compared with the second operand; the comparison determines the indicated result conditions. The operands remain unchanged.

Comparison is algebraic, treating both operands as 8-bit signed integers. It is performed by adding the ones complement of all 8 bits of the second operand and a low-order 1 to all 8 bits of the first operand, as in SUBTRACT (byte, register).

An implied second-operand byte of all 0's is used in place of the register contents when the r_2 field of the instruction is all 0's.

The operands are located in the same register set, designated with bit positions 12-15 of the instruction: hexadecimal "6" designates the primary set, and hexadecimal "7" designates the secondary set.

Result Conditions
8 Operands are equal.
4 First operand is low.
2 First operand is high.
1 Overflow.
0 Carry out of sign-bit position.

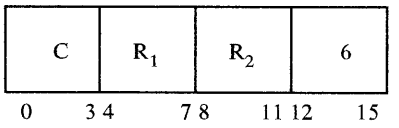
Program Exceptions
None

Programming Notes
Although the comparison is algebraic, result condition 0 can be tested after the instruction is executed to determine the logical relation of the two operands when the operands are considered to be binary unsigned quantities. When a carry is indicated, the first operand is logically higher than or equal to the second operand. When a carry is not indicated, the first operand is logically lower. An indication of overflow (result condition 1) is not significant.

The byte in bit positions 16-23 of register 0 in the primary and secondary register sets can be designated only as the first operand; it cannot be designated as the second operand.

COMPARE (halfword, register)

CHR rh,rh



RR Format

Operation
Result-Conditions <== (R_1<16..31>) + ¬(R_2<16..31>) + 1

Description
The first operand is compared with the second operand; the comparison determines the indicated result conditions. The operands remain unchanged.

Comparison is algebraic, treating both operands as 16-bit signed integers. It is performed by adding the ones complement of all 16 bits of the second operand and a low-order 1 to all 16 bits of the first operand, as in SUBTRACT (halfword, register).

The operands occupy the low-order halfwords of the registers designated by the R_1 and R_2 fields.

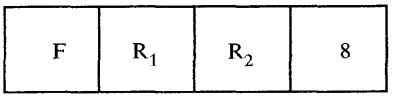
Result Conditions
8 Operands are equal.
4 First operand is low.
2 First operand is high.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions
None

Programming Note
Although the comparison is algebraic, result condition 0 can be tested after the instruction is executed to determine the logical relation of the two operands when the operands are considered to be binary unsigned quantities. When a carry is indicated, the first operand is logically higher than or equal to the second operand. When a carry is not indicated, the first operand is logically lower. An indication of overflow (result condition 1) is not significant.

COMPARE WITH CARRY (halfword, register, extended)

CYHRE ruh,ruh



RR Format

Operation

If $R_2 \neq 0000$
  Then Result-Conditions <== (R_1<0..15>) + ¬(R_2<0..15>) + C
  Else Result-Conditions <== (R_1<0..15>) + ¬0000000000000000 + C

Description

The first operand is compared with the second operand; the comparison determines the indicated result conditions. Bit 56 (the carry-condition indicator) in the current PSV participates in the operation. The operands remain unchanged.

Comparison is algebraic, treating both operands as 16-bit signed integers. It is performed by adding the ones complement of all 16 bits of the second operand and a low-order 0 or 1, taken from bit 56 (C) in the current PSV, to all 16 bits of the first operand, as in SUBTRACT WITH CARRY (halfword, register, extended). Algebraically, a borrow from the first operand occurs (due to the previous compare operation) when PSV-bit 56 is 0; no borrow occurs when bit 56 is 1.

An implied second-operand halfword of all 0's is used in place of the register contents when the R_2 field of the instruction is all 0's.

The operands occupy the high-order halfwords of the registers designated by the R_1 and R_2 fields.

Result Conditions

8 Extended operands are equal.
4 First extended operand is low.
2 First extended operand is high.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions

None

Programming Notes

COMPARE WITH CARRY (halfword, register, extended) is provided for the comparison of extended fixed-point numbers. A carry from any COMPARE instruction or the COMPARE WITH CARRY (halfword, register, extended) instruction is accounted for by executing a subsequent COMPARE WITH CARRY (halfword, register, extended) instruction without executing an intervening instruction that changes the indicated result conditions.

COMPARE WITH CARRY (halfword, register, extended) can be used following a COMPARE (halfword, register) instruction to perform word register-to-register comparison. This instruction can also be used to account for only the borrow, if any, due to the previous COMPARE by specifying an implied second-operand halfword of all 0's.

Result condition 8 can be indicated (reflecting an equal comparison) only if it was indicated at the beginning of the operation.

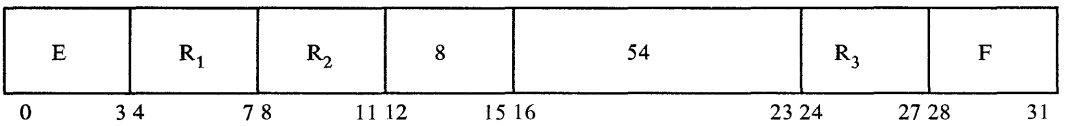
Although the comparison is algebraic, result condition 0 can be tested after the instruction is executed to determine the logical relation of the two extended fixed-point operands when the operands are considered to be binary unsigned quantities. When a carry is indicated, the first operand is logically higher than or equal to the second operand. When a carry is not indicated, the first operand is logically lower.

The halfword in bit positions 0-15 of register 0 in the primary register set can be designated only as the first operand; it cannot be designated as the second operand.

For compare operations, an indication of overflow (result condition 1) is not significant.

COMPARE LOGICAL (bytes, storage)

CLS ra,ra,rh



RR-Long Format

Operation

LOOP Result-Conditions <== MS[(R_1)]:MS[(R_2)]
TEMP1 <— (R_1)
TEMP2 <— (R_2)
TEMP3 <— (R_3<24..31>)
(R_1) <— TEMP1 + 1
(R_2) <— TEMP2 + 1
(R_3<24..31>) <— TEMP3 - 1
If Result-Condition=Equal and (R_3<24..31>)≠00000000
Then Go To LOOP
Else NSI

Description

The first operand is compared with the second operand; the comparison determines the indicated result conditions. The operands remain unchanged.

The locations of the leftmost bytes of the first- and second-operand fields are designated by the contents of the general registers specified by R_1 and R_2, respectively. The length of the first and second operands is designated by the contents of bit positions 24-31 of the general register specified by R_3. The contents of bit positions 0-23 of register R_3 are ignored and remain unchanged. The length is specified in terms of bytes, and comparison is performed on a byte-by-byte basis.

The comparison is logical, treating the first and second operands as binary unsigned quantities, with all binary values valid. The operation starts at the leftmost end of both fields and proceeds to the right in units of bytes. The operation ends when an inequality is detected or the end of the fields is reached.

Execution of the instruction is interruptible between units of operation. Conceptually, after each unit of operation, including the last, the operand addresses in registers R_1 and R_2 are both increased by 1 and the count in bit positions 24-31 of register R_2 is decremented increased by 1. When, before the

last unit of operation, an I/O interruption occurs, or a system-check interruption occurs due to a channel I/O check, the operand addresses and the count are updated so that the instruction, when reexecuted, resumes at the point of interruption.

Depending on processor model, more than one unit of operation may be executed between points in the operation at which an interruption is allowed. In this case, the number of units of operation executed without allowing an interruption is predetermined. After each predetermined number of units of operation, the operand addresses and count value are adjusted to correspond to the amount of data compared. The specific predetermined number of units of operation is fixed, except for the first and last execution groups.

The first and second operands are of equal length. They may be from 1 to 256 bytes. A count of 256 is designated with an initial value of all 0's in bit positions 24-31 of register R_3. Depending on processor model, if R_3 designates the same register as R_1 or R_2, an operation exception may not be indicated and the result is unpredictable.

When part of an operand is designated in an inaccessible location, but the operation ends because of an inequality by referring only to the available part of the operand, a program exception is not indicated. Otherwise, the exception is indicated.

The program exception is detected at the time the inaccessible location is referred to, and execution of the instruction is terminated. That is, the contents of registers R_1, R_2, and R_3 may not be adjusted to correspond with the amount of data compared.

Depending on processor model, if R_3 designates the same register as R_1 or R_2, an operation exception may not be indicated and the result is unpredictable.

Result Conditions

8 Operands are equal.
4 First operand is low.
2 First operand is high.
1 --
0 Carry out of high-order bit position of last byte compared.

Program Exceptions (Termination)

Specification (operand 1 or 2: real address)
Access (operand 1 or 2: block invalid)
Operation (depending on processor model, $R_3 = R_1$ or R_2)
Separation (operand 1 or 2)
Address (operand 1 or 2: address limit)

Programming Notes

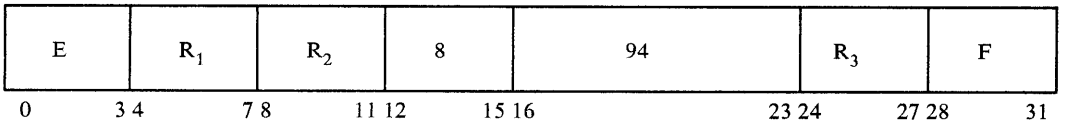
Since execution of the COMPARE LOGICAL (bytes, storage) is interruptible, the instruction cannot be used for situations in which interruptions are enabled and the program must rely on uninterrupted execution of the instruction.

When R_1 and R_2 specify the same register, data is compared with itself.

For compare-logical operations, an indication of carry (result condition 0) is not significant.

COMPARE LOGICAL (halfwords, storage)

CLHS ra,ra,rh



RR-Long Format

Operation

```
LOOP   Result-Conditions <== MS[(R1)]:MS[(R2)]
       TEMP1   <—   (R1)
       TEMP2   <—   (R2)
       TEMP3   <—   (R3<24..31>)
       (R1)   <—   TEMP1 + 2
       (R2)   <—   TEMP2 + 2
       (R3<24..31>)   <—   TEMP3 - 1
       If Result-Condition=Equal and (R3<24..31>)≠00000000
            Then Go To LOOP
            Else NSI
```

Description

The first operand is compared with the second operand; the comparison determines the indicated result conditions. The operands remain unchanged.

The locations of the leftmost halfwords of the first- and second-operand fields are designated by the contents of the general registers specified by R_1 and R_2, respectively. The length of the first and second operands is designated by the contents of bit positions 24-31 of the general register specified by R_3. The contents of bit positions 0-23 of register R_3 are ignored and remain unchanged. The length is specified in terms of halfwords, and comparison is performed on a halfword-by-halfword basis.

The comparison is logical, treating the first and second operands as binary unsigned quantities, with all binary values valid. The operation starts at the leftmost end of both fields and proceeds to the right in units of halfwords. The operation ends when an inequality is detected or the end of the fields is reached.

Execution of the instruction is interruptible between units of operation. Conceptually, after each unit of operation, including the last, the operand addresses in registers R_1 and R_2 are both increased by 2 and the count in bit positions 24-31 of register R_3 is decremented by 1. When, before the last unit of operation, an I/O interruption occurs, or a system-check interruption occurs due to a channel I/O check, the operand addresses and the count are updated so that the instruction, when re-executed, resumes at the point of interruption.

Depending on processor model, more than one unit of operation may be executed between points in the operation at which an interruption is allowed. In this case, the number of units of operation executed without allowing an interruption is predetermined. After each predetermined number of units of operation, the operand addresses and count value are adjusted to correspond to the amount of data compared. The specific predetermined number of units of operation is fixed, except for the first and last executions of the groups.

The first and second operands are of equal length. They may be from 1 to 256 halfwords. A count of 256 is designated with an initial value of all 0's in bit positions 24-31 of register R_3.

When part of an operand is designated in an inaccessible location, but the operation ends because of an inequality by referring only to the available part of the operand, a program exception is not indicated. Otherwise, the exception is indicated.

The program exception is detected at the time the inaccessible location is referred to, and execution of the instruction is terminated. That is, the contents of registers R_1, R_2, and R_3 may not be adjusted to correspond with the amount of data compared.

Depending on processor model, if R_3 designates the same register as R_1 or R_2, an operation exception may not be indicated and the result is unpredictable.

Result Conditions

8 Operands are equal.
4 First operand is low.
2 First operand is high.
1 --
0 Carry out of high-order bit position of last halfword compared.

Program Exceptions (Termination)

Specification (operand 1 or 2: real address)
Access (operand 1 or 2: block invalid)
Operation (depending on processor model, $R_3 = R_1$ or R_2)
Separation (operand 1 or 2)
Address (operand 1 or 2: address limit)

Programming Notes

Since execution of COMPARE LOGICAL (halfwords, storage) is interruptible, the instruction cannot be used for situations in which interruptions are enabled and the program must rely on uninterrupted execution of the instruction.

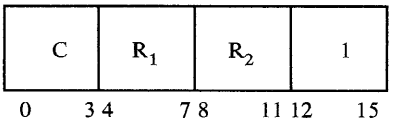
COMPARE LOGICAL (halfwords, storage) can be used to compare two byte strings when the strings begin on halfword boundaries and contain an even number of bytes. Note, however, that when the instruction ends because of an inequality, the result conditions indicate the relation of the last *halfwords* compared.

When R_1 and R_2 specify the same register, data is compared with itself.

For compare-logical operations, an indication of carry (result condition 0) is not significant.

COUNT LEADING ZEROS (halfword)

CTLZ rh,rh



RR Format

Operation

```
        TEMP    <—  0
LOOP    If (R2<TEMP + 16>) ≠ 0
                Then (R2<TEMP+16>)    <—  0
                        Go To END
                Else  TEMP    <—    TEMP + 1
                        If TEMP ≠ 16
                                Then Go To LOOP
                                Else  Continue
END     Result-Conditions <== (R2<16..31>)
        (R1<16..31>)  <—  TEMP
```

Description

A count of the number of leading (leftmost) 0 bits in the second-operand halfword is placed in the first-operand location, and the leftmost 1-bit in the second operand is made 0. The indicated result conditions are determined by the second-operand result.

The count placed in the first-operand location is an unsigned binary integer. The count value can range from 0 to 16, inclusive.

The operands occupy the low-order halfwords of the general registers specified by R_1 and R_2.

Result Conditions

8 Second-operand result is all 0's.
4 --
2 Second-operand result is mixed 0's and 1's.
1 --
0 --

Program Exceptions

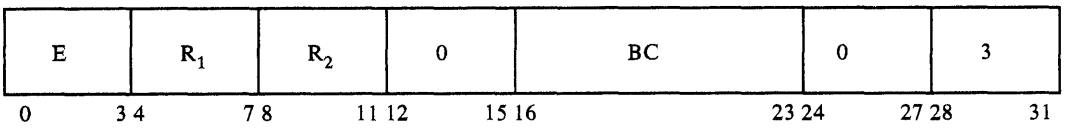
None

Programming Note

If R_1 and R_2 specify the same register, the second operand is replaced with the count.

DIVIDE (halfword, register)

DHR rh,rh



RR-Long Format

Operation

TEMP1 <— (R_1<16..31>) | | ({R_1+0010}<16..31>)
TEMP2 <— (R_2<16..31>)
(R_1<16..31>) <— TEMP1 MOD TEMP2
({R_1+0010}<16..31>) <— TEMP1 / TEMP2

Description

The dividend (first operand) is divided by the divisor (second operand), and the remainder and quotient are placed in the first-operand location.

The dividend is an unsigned 32-bit positive binary integer. The high-order half and low-order half of the dividend occupies the two low-order halfwords, respectively, of the even-odd pair of consecutive registers designated by the R_1 field. The divisor occupies the low-order halfword of the register designated by the R_2 field. The remainder and quotient replace the dividend in the even and odd registers, respectively. The high-order halfword of the general registers in which the dividend and divisor are located do not participate in the operation and remain unchanged. The remainder, quotient, and divisor are all treated as unsigned 16-bit positive binary integers.

When the relative magnitude of the dividend and divisor is such that the quotient cannot be expressed by an unsigned 16-bit integer, or when the divisor is 0, a fixed-point overflow exception is indicated. If the R_1 field contains xx1x (where x may be 0 or 1), specifying the odd register of an even-odd pair, the result is unpredictable, and a program exception due to the specification is not indicated.

Bit positions 24-27 of the instruction are reserved and must contain all 0's; otherwise, an operation exception is indicated.

Result Conditions

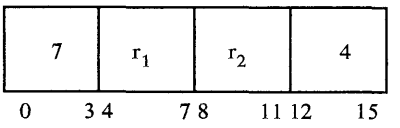
The conditions remain unchanged.

Program Exceptions (Suppression)

Operation (bits 24-27 of instruction not all 0's)
Fixed-Point Overflow

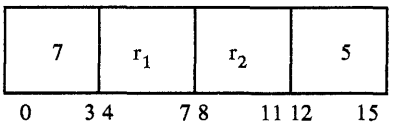
EXCLUSIVE OR (byte, register)

XR rpb,rpb



RR Format
(primary-register-set operands)

XR rsb,rsb



RR Format
(secondary-register-set operands)

Operation

If $r_2 \neq 0000$
  Then $(r_1) \leftarrow (r_1) \not\equiv (r_2)$
  Else $(r._1) \leftarrow (r_1) \not\equiv 00000000$

Description

The EXCLUSIVE OR of the first- and second-operand bytes is placed in the first-operand location.

The operands are treated as unstructured logical quantities, and the connective EXCLUSIVE OR is applied bit by bit. A bit position in the result is set to 1 if the bits in the corresponding positions of both operands are unlike; otherwise, the result bit is set to 0.

An implied second-operand byte of all 0's is used in place of the register contents when the r_2 field of the instruction is all 0's.

The operands are located in the same register set, designated with bit positions 12-15 of the instruction: hexadecimal "4" designates the primary set, and hexadecimal "5" designates the secondary set.

Result Conditions

8 Result is all 0's.
4 Result is all 1's.
2 Result is mixed 0's and 1's.
1 --
0 --

Program Exceptions

None

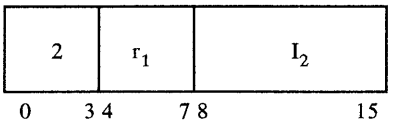
Programming Notes

This instruction may be used to invert a bit.

The byte in bit positions 16-23 of register 0 in the primary and secondary register sets can be designated only as the first operand; it cannot be designated as the second operand.

EXCLUSIVE OR (byte, register-immediate)

XRI rpb,i8



RI Format

Operation

$(r._1) \leftarrow (r_1) \not\equiv I_2$

Description

The EXCLUSIVE OR of the first-operand byte and the byte of immediate data is placed in the first-operand location.

The operands are treated as unstructured logical quantities, and the connective EXCLUSIVE OR is applied bit by bit. A bit position in the result is set to 1 if the bits in the corresponding positions of both operands are unlike; otherwise, the result bit is set to 0.

The first operand is located in the primary register set.

Result Conditions

8 Result is all 0's.
4 Result is all 1's.
2 Result is mixed 0's and 1's.
1 --
0 --

Program Exceptions

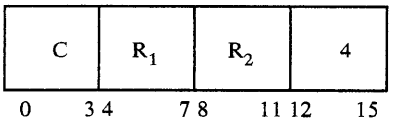
None

Programming Note

This instruction may be used to invert a bit.

EXCLUSIVE OR (halfword, register)

XHR rh,rh



RR Format

Operation

$(R_1<16..31>) \leftarrow (R_1<16..31>) \not\equiv (R_2<16..31>)$

Description

The EXCLUSIVE OR of the first- and second-operand halfwords is placed in the first-operand location.

The operands are treated as unstructured logical quantities, and the connective EXCLUSIVE OR is applied bit by bit. A bit position in the result is set to 1 if the bits in the corresponding positions of both operands are unlike; otherwise, the result bit is set to 0.

The operands occupy the low-order 16 bits of the registers specified by the R_1 and R_2 fields.

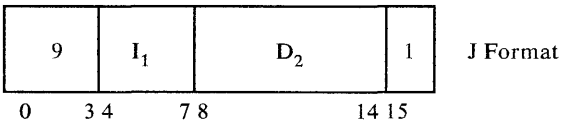
Result Conditions
8 Result is all 0's.
4 Result is all 1's.
2 Result is mixed 0's and 1's.
1 --
0 --

Program Exceptions
None

Programming Note
This instruction may be used to invert a bit.

JUMP ON BIT ZERO (halfword)

JBZ n4,dh7s



Operation
If (PGR1<I_1+16>) = 0
Then IA <— IA + D_2 x 2
Else NSI

Description
The updated instruction address in the current PSV is replaced by the jump address if the specified bit in the implied halfword operand is 0. Otherwise, normal instruction sequencing proceeds with the updated instruction address. The second-operand address is used as the jump address.

The implied operand occupies the low-order halfword of primary general register 1. The I_1 field contains an unsigned 4-bit binary integer that specifies the bit position in the implied-operand halfword. The value of I_1 can range from 0 to 15 (binary 0000 to 1111). A value of 0 tests bit position 16 of primary general register 1; a value of 1 tests bit position 17; and so on.

When a jump occurs, the jump address is tested for validity as part of the instruction execution. If a specification, access, separation, or address exception is detected for the first halfword associated with the jump address, the IA in the stored PSV indicates the current instruction as the failing operation.

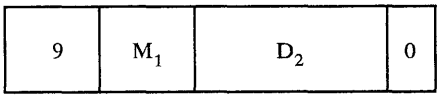
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, execution protection)
Separation (operand 2)
Address (operand 2: all)

Programming Note
Relative to the updated instruction address, the jump range, in bytes, covered by the displacement (D_2) is $-128 \leq D_2 \times 2 \leq 126$. Note that the displacement can be specified only in terms of an even number of bytes.

JUMP ON CONDITION

JC m4,dh7s



J Format

Operation
If M_1 specifies an indicated result condition
  Then IA <— IA + D_2 x 2
  Else NSI

Description
The updated instruction address in the current PSV is replaced by the jump address if any result condition designated by the M_1 field is indicated in the current PSV. Otherwise, normal instruction sequencing proceeds with the updated instruction address. The second-operand address is used as the jump address.

The M_1 field is used as a 4-bit mask. The 4 bits of the mask correspond, left to right, with the four result conditions 8, 4, 2, and 1. A 1-bit in the mask specifies that the corresponding condition is to be tested. A mask containing two or more 1's tests all corresponding conditions. A mask of all 0's specifies a test for result condition 0.

M_1 Field	Mask Value	Result Condition Tested
1000	8	8
0100	4	4
0010	2	2
0001	1	1
0000	0	0

The jump is successful whenever any tested result condition is indicated in the current PSV. The jump is not taken if none of the tested conditions are indicated.

When a jump occurs, the jump address is tested for validity as part of the instruction execution. If a specification, access, separation, or address exception is detected for the first halfword associated with the jump address, the IA in the stored PSV indicates the current instruction as the failing operation.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, execution protection)
Separation (operand 2)
Address (operand 2: all)

Programming Notes
A jump can be made on more than one result condition, except condition 0, by specifying the pertinent conditions in the mask as the sum of their corresponding mask values. A mask value of 12 (binary 1100), for example, specifies that a jump is to be made if either result condition 8 or 4 is indicated.

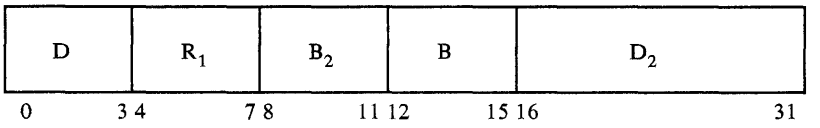
An unconditional jump is made by specifying a mask of 14 or 15. When the D_2 field contains all 0's, the jump instruction is equivalent to a no-operation.

Result condition 8, 4, or 2 can be tested for its absence by specifying the other two conditions in the mask. A mask of 12, for example, will cause a jump to be executed if result condition 2 is not indicated. Result conditions 1 and 0 are each indicated independently, and the mask only provides for testing whether the corresponding condition is indicated.

Relative to the updated instruction address, the jump range, in bytes, covered by the displacement (D_2) is $-128 \leq D_2 \times 2 \leq 126$. Note that the displacement can be specified only in terms of even number or bytes.

LOAD ADDRESS

LA ra,db16s(ra)



RR-Long Format

Operation
If $B_2 \neq 0000$
  Then $(R_1) \leftarrow (B_2) + D_2$
  Else $(R_1) \leftarrow IA + D_2$

Description
The sum of the base address and displacement, designated by the B_2 and D_2 fields, respectively, is placed in the general register designated by the R_1 field.

No storage references for operands take place and the computed address is not inspected for address exceptions. When the B_2 field contains all 0's, the address is computed using the updated instruction address in place of the contents of primary general register 0.

The address computation follows the rules for address arithmetic for RS-Long format instructions. Specifically, the D_2 field contains a 16-bit signed binary integer that is expanded to the left to 32 bits during address computation. The expansion is achieved by setting the 16 leftmost bits equal to the sign bit of the displacement. The computed address is then formed by adding all 32 bits of the expanded displacement to all 32 bits of the base address. The carry, if any, out of the high-order bit position is ignored.

Result Conditions
The conditions remain unchanged.

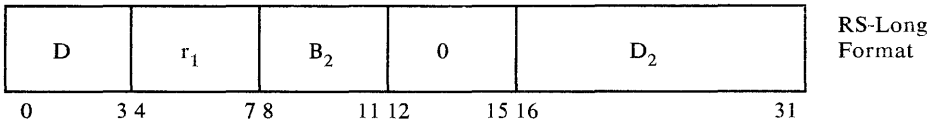
Program Exceptions
None

Programming Notes
The same general register may be specified by the R_1 and B_2 instruction fields, except that primary general register 0 can be specified only by the Re1 field. In this manner, it is possible to increment or decrement the contents of a general register, other then a primary register 0, by the contents of the D_2 field of the instruction.

The range covered by the displacement (D_2) is $-32768 \leq D_2 \leq 32767$.

LOAD (byte)

L rpb,db16s(ra)



Operation
If $B_2 \neq 0000$
  Then $(r_1) \leftarrow MS[(B_2) + D_2]$
  Else $(r_1) \leftarrow MS[IA + D_2]$

Description
The byte at the second-operand location is placed unchanged in the first-operand byte location.

When the B_2 field contains all 0's,the second-operand address is computed using the updated instruction address in place of the contents of primary general register 0.

The first operand is located in the primary register set.

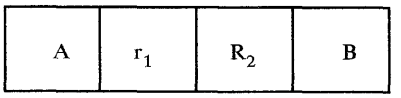
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Separation (operand 2)
Address (operand 2: all)

Programming Note
Relative to the base address for the second-operand location, the range, in bytes, covered by the displacement (D_2) is $-32768 \leq D_2 \leq 32767$.

LOAD (byte, with index)

LN rpb,ra



RR Format

Operation
(r_1) <— MS[(R_2)]

Description
The byte at the second-operand location is placed unchanged in the first-operand byte location.

The contents of the general register specified by the R_2 field are used as the second-operand address.

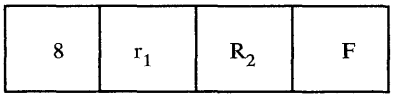
The first operand is located in the primary register set.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Separation (operand 2)
Address (operand 2: address limit)

LOAD (byte, with index decremented)

LND rpb,ra



RR Format

Operation
(R_2) <— (R_2) - 1
(r_1) <— MS[(R_2)]

Description
The contents of the general register designated by the R_2 field are decremented by 1, and the result is used as the second-operand address. The byte at the second-operand location is then placed unchanged in the first-operand byte location.

The contents of the general register specified by R_2 are reduced by 1 before the byte is placed in the first-operand location.

The first operand is located in the primary register set.

Decrementing the contents of register R_2 through 0 causes a wraparound to 4,294,967,295 (hex FFFF FFFF).

Program exceptions pertain to the *decremented* second-operand address and are indicated only when the decremented second-operand field is inaccessible. Detection of the program exception occurs when a reference to the inaccessible location is attempted. The execution of the instruction is *suppressed;* that is, the first-operand location and the second-operand address remain unchanged.

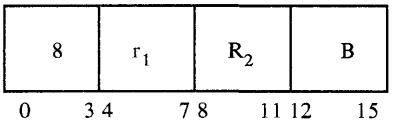
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Separation (operand 2)
Address (operand 2: address limit)

Programming Note
The updated contents of the general register specified by R_2 are partially overwritten with the byte fetched from main storage when the first operand is located in the general register specified by R_2.

LOAD (byte, with index incremented)

LNI rpb,ra



RR Format

Operation
TEMP <— MS[(R_2)]
(R_2) <— (R_2) + 1
(r_1) <— TEMP

Description
The byte at the second-operand location is placed unchanged in the first-operand byte location.

The initial contents of the general register designated by the R_2 field are used as the second-operand address. The contents of the register specified by R_2 are incremented by 1 after the byte is fetched from main storage and before it is placed in the first-operand location.

The first operand is located in the primary register set.

Program exceptions pertain to the *initial* second-operand address and are indicated only when the initial second-operand field is inaccessible. Detection of the program exception occurs when a reference to the inaccessible location is attempted. The execution of the instruction is *suppressed;* that is, the first-operand location and the second-operand address remain unchanged.

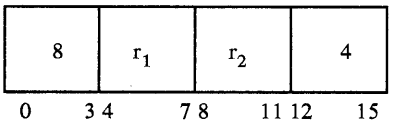
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Separation (operand 2)
Address (operand 2: address limit)

Programming Note
The updated contents of the general register specified by R_2 are partially overwritten with the byte fetched from main storage when the first operand is located in the general register specified by R_2.

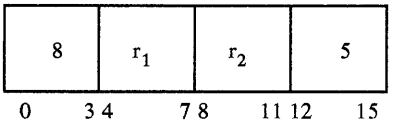
LOAD (byte, register)

LR rpb,rpb



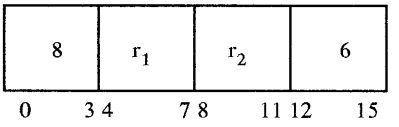
RR Format
(r_1: primary; r_2: primary)

LR rpb,rsb



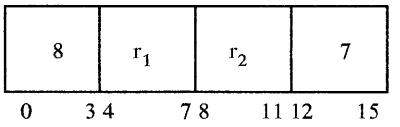
RR Format
(r_1: primary; r_2: secondary)

LR rsb,rpb



RR Format
(r_1: secondary; R_2: primary)

LR rsb,rsb



RR Format
(r_1: secondary; r_2: secondary)

Operation
(r_1) <— (r_2)

Description
The second-operand byte is placed unchanged in the first-operand byte location.

Bit positions 12-15 of the instruction indicate the register set or sets in which the first and second operands are located.

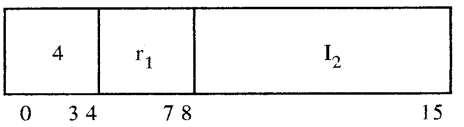
Instruction Bits 12-15	First Operand (r_1) Location	Second Operand (r_2) Location
0100 (hex 4)	primary	primary
0101 (hex 5)	primary	secondary
0110 (hex 6)	secondary	primary
0111 (hex 7)	secondary	secondary

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

LOAD (byte, register-immediate)

LRI rpb,i8



RI Format

Operation

(r_1) <— I_2

Description

The byte of immediate data is placed unchanged in the first-operand byte location.

The first operand is located in the primary register set.

Result Conditions

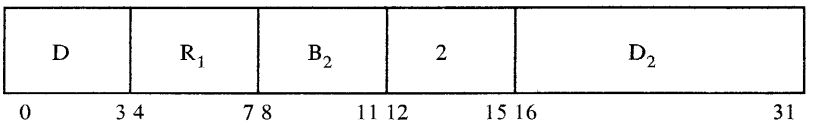
The conditions remain unchanged.

Program Exceptions

None

LOAD (halfword)

LH rh,db16s(ra)



RS-Long Format

Operation

If $B_2 \neq 0000$

Then $(R_1<16..31>)$ <— $MS[(B_2) + D_2]$

Else $(R_1<16..31>)$ <— $MS[IA + D_2]$

Description

The halfword at the second-operand location is placed unchanged in the first-operand location.

When the B_2 field contains all 0's, the second-operand address is computed using the updated instruction address in place of the contents of primary general register 0.

The first operand occupies the low-order halfword of the general register specified by R_1.

Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)

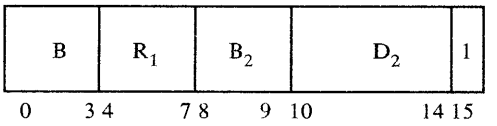
Specification (operand 2: real address)
Access (operand 2: block invalid)
Separation (operand 2)
Address (operand 2: all)

Programming Notes
Relative to the base address for the second-operand location, the range, in bytes, covered by the displacement (D_2) is $-32768 \leq D_2 \leq 32767$.

The contents of the general register specified by B_2 are partially overwritten with the halfword fetched from main storage when the first operand is located in the general register specified by B_2.

LOAD (halfword, short form)

LHS rh,dh5(ra)



RS Format

Operation
$(R_1<16..31>) \leftarrow MS[(B_2) + D_2 \times 2]$

Description
The halfword at the second-operand location is placed unchanged in the first-operand location.

The first operand occupies the low-order halfword of the general register specified by R_1.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Separation (operand 2)
Address (operand 2: address limit)

Programming Notes
The short form of the LOAD (halfword) instruction is provided to conserve program space. It can be used for base-plus-displacement addressing of data structures that comprise up to 32 contiguous halfwords.

The contents of the B_2 field represent the 2 low-order bits of a 4-bit R field in which the 2 high-order bits are both 1's. The specification of the base register is therefore limited to primary general register 6 or 7, or secondary general register 6 or 7, as shown in the following chart:

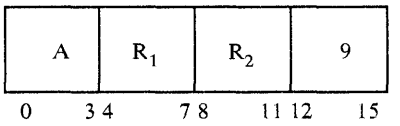
B_2 Field Bits 8, 9	Register Specified
0 0	Register 6, Primary Set
0 1	Register 6, Secondary Set
1 0	Register 7, Primary Set
1 1	Register 7, Secondary Set

Relative to the base address for the second-operand location, the range, in bytes, covered by the displacement (D_2) is $0 \leq D_2 \times 2 \leq 62$. Note that the displacement can be specified only in terms of an even number of bytes.

The contents of the general register specified by B_2 are partially overwritten with the halfword fetched from main storage when the first operand is located in the general register specified by B_2.

LOAD (halfword, with index)

LHN rh,ra



RR Format

Operation
(R_1<16..31>) <— MS[(R_2)]

Description
The halfword at the second-operand location is placed unchanged in the first-operand location.

The contents of the general register specified by the R_2 field are used as the second-operand address.

The first operand occupies the low-order halfword of the general register specified by R_1.

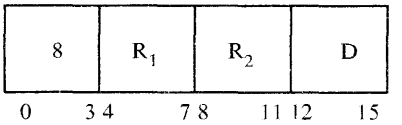
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Separation (operand 2)
Address (operand 2: address limit)

Programming Note
The contents of the general register specified by R_2 are partially overwritten with the halfword fetched from main storage when the first operand is located in the general register specified by R_2.

LOAD (halfword, with index decremented)

LHND rh,ra



RR Format

Operation
(R_2) <— (R_2) - 2
(R_1<16..31>) <— MS[(R_2)]

Description
The contents of the general register designated by the R_2 field are decremented by 2, and the result is used as the second-operand address. The halfword at the second-operand location is then placed unchanged in the first-operand halfword location.

The contents of the general register specified by R_2 are reduced by 2 before the halfword is placed in the first-operand location.

The first operand occupies the low-order halfword of the register specified by R_1.

Decrementing the contents of register R_2 through 0 causes a wraparound to 4,294,967,295 (hex FFFF FFFF).

Program exceptions pertain to the *decremented* second-operand address and are indicated only when the decremented second-operand field is inaccessible. Detection of the program exception occurs when a reference to the inaccessible location is attempted. The execution of the instruction is *suppressed;* that is, the first-operand location and the second-operand address remain unchanged.

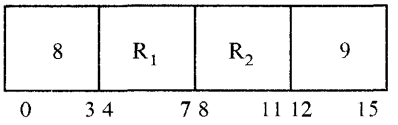
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Separation (operand 2)
Address (operand 2: address limit)

Programming Note
The updated contents of the general register specified by R_2 are partially overwritten with the halfword fetched from main storage when the first operand is located in the general register specified by R_2.

LOAD (halfword, with index incremented)

LHNI rh,ra



RR Format

Operation
TEMP <— MS[(R_2)]
(R_2) <— (R_2) + 2
(R_1<16..31>) <— TEMP

Description
The halfword at the second-operand location is placed unchanged in the first-operand halfword location.

The initial contents of the general register designated by the R_2 field are used as the second-operand address. The contents of the register specified by R_2 are incremented by 2 after the halfword is fetched from main storage and before it is placed in the first-operand location.

The first operand occupies the low-order halfword of the register specified by R_1.

Program exceptions pertain to the *initial* second-operand address and are indicated only when the initial second-operand field is inaccessible. Detection of the program exception occurs when a reference to the inaccessible location is attempted. The execution of the instruction is *suppressed;* that is, the first-operand location and the second-operand address remain unchanged.

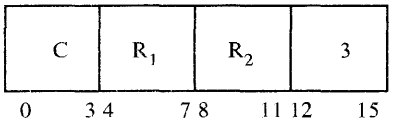
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Separation (operand 2)
Address (operand 2: address limit)

Programming Note
The updated contents of the general register specified by R_2 are partially overwrittern with the halfword fetched from main storage when the first operand is located in the general register specified by R_2.

LOAD (halfword, register)

LHR rh,rh



RR Format

Operation
$(R_1<16..31>) \leftarrow (R_2<16..31>)$

Description
The second operand is placed unchanged in the first-operand location.

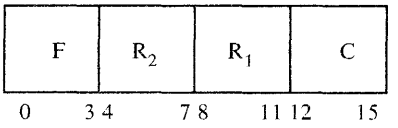
The first and second operands occupy the low-order halfwords of the general registers specified by R_1 and R_2, respectively.

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

LOAD (halfword, register, lower half from upper)

LHRLU rh,ruh



RR Format

Operation
$(R_1<16..31>) \leftarrow (R_2<0..15>)$

Description
The second operand is placed unchanged in the first-operand location.

The first operand occupies the low-order halfword of the general register specified by R_1. The second operand occupies the high-order halfword of the general register specified by R_2.

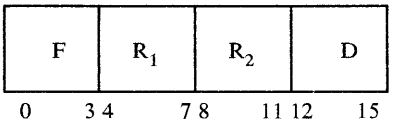
Result Conditions
The conditions remain unchanged.

Program Exceptions
None

Programming Note
Bit positions 4-7 and 8-11 of the instruction format contain the R_2 and R_1 fields, respectively. This is reversed from the normal left-to-right order for the RR instruction format.

LOAD (halfword, register, upper half)

LHRU ruh,ruh



RR Format

Operation
$(R_1<0..15>) \leftarrow (R_2<0..15>)$

Description
The second operand is placed unchanged in the first-operand location.

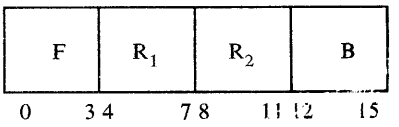
The first and second operands occupy the high-order halfwords of the general registers specified by R_1 and R_2, respectively.

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

LOAD (halfword, register, upper half from lower)

LHRUL ruh,rh



RR Format

Operation
$(R_1<0..15>) \leftarrow (R_2<16..31>)$

Description
The second operand is placed unchanged in the first-operand location.

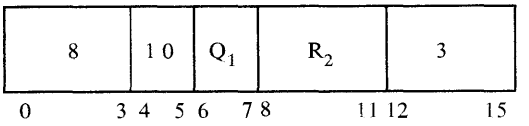
The first operand occupies the high-order halfword of the general register specified by R_1. The second operand occupies the low-order halfword of the general register specified by R_2.

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

LOAD (halfwords, quadrant)

LHQ q2,ra



RR Format

Operation
TEMP <— (R_2)
RQ<Q_1> <— MS[TEMP]
(R_2) <— TEMP + 16

Description
The eight consecutive general-register halfword fields specified by the first operand (Q_1) are loaded from the main storage locations designated by the second-operand address.

The contents of the register designated by R_2 are used as the second-operand address. At the completion of the operation, the second-operand address is increased by 16, and the updated address is placed back in the register specified by R_2.

The main storage area from which the halfwords are fetched starts at the location designated by the second-operand address and includes eight consecutive halfword locations. The general-register halfword fields are loaded in ascending order beginning with the first (lowest numbered) register of the set indicated by Q_1.

The register quadrant designated by the Q_1 field consists of the eight high-order or low-order halfword operands of the general registers that make up the primary or secondary register set, as shown in the following table:

	Register Quadrant	
Q_1 Operand	**Register Set**	**Halfword Fields**
00	primary	low-order <16..31>
01	secondary	low-order <16..31>
10	primary	high-order <0..15>
11	secondary	high-order <0..15>

When any part of the second operand is inaccessible, a specification, access, separation or address exception is detected at the time the inaccessible location is referred to, and execution is terminated. That is, the updated second-operand

address may not be stored back in register R_2; the original contents of the register fields that are loaded, if any, are lost. However, valid retry of the instruction can always be assured if R_2 is specified as register 7 of either the primary or secondary register set.

Bit position 4 of the instruction is used as an extension to the operation code; the bit distinguishes this instruction from ROTATE LEFT (byte). Bit position 5 of the instruction is reserved and must be 0; otherwise, an operation exception is indicated, and the operation is suppressed.

Result Conditions
The conditions remain unchanged.

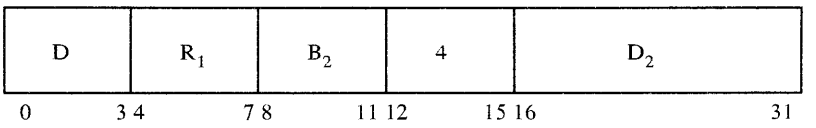
Program Exceptions (Termination/Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Operation (bit 5 of instruction is a 1)
Separation (operand 2)
Address (operand 2: address limit)

Programming Notes
When the register designated by R_2 is in the register set indicated by Q_1, the halfword field fetched from main storage is overwritten with the updated address.

When the second-operand location (eight consecutive halfwords) is completely unavailable, execution of this instruction is *suppressed* due to the specification, access, separation, or address exception. Therefore, termination of the operation can be avoided by locating the second operand starting at an address that is an integral multiple of 16.

LOAD (word)

LW rw,db16s(ra)



RS-Long Format

Operation
If $B_2 \neq 0000$
  Then (R_1) <— $MS[(B_2) + D_2]$
  Else (R_1) <— $MS[IA + D_2]$

Description
The word at the second-operand location is placed unchanged in the first-operand location.

When the B_2 field contains all 0's, the second-operand address is computed using the updated instruction address in place of the contents of primary general register 0.

Result Conditions
The conditions remain unchanged.

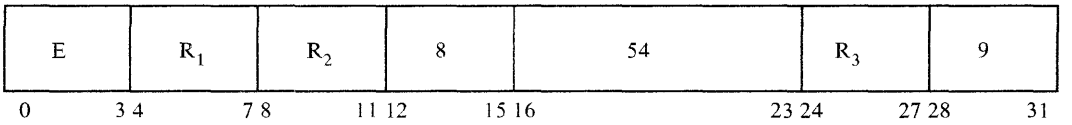
Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Separation (operand 2)
Address (operand 2: all)

Programming Notes
Relative to the base address for the second-operand location, the range, in bytes, covered by the displacement (D_2) is -32768 ≤ D_2 ≤ 32767.

The contents of the general register specified by B_2 are overwritten with the word fetched from main storage when the first operand is located in the general register specified by B_2.

MOVE (bytes, storage)

MVS ra,ra,rh



RR-Long Format

Operation

```
LOOP   MS[(R1)]   <—   MS[(R2)]
       TEMP1      <—   (R1)
       TEMP2      <—   (R2)
       TEMP3      <—   (R3<24..31>)
       (R1)       <—   TEMP1 + 1
       (R2)       <—   TEMP2 + 1
       (R3<24..31>)    <—   TEMP3 - 1
       If (R3<24..31>) ≠ 00000000
              Then Go To LOOP
              Else NSI
```

Description
The second operand is placed in the first-operand location.

The locations of the leftmost byte of the first- and second-operand fields are designated by the contents of the general registers specified by R_1 and R_2, respectively. The length of the first and second operands is designated by the contents of bit positions 24-31 of the general register specified by R_3. The contents of bit positions 0-23 of register R_3 are ignored and remain unchanged. The length is specified in terms of bytes, and the second-operand field is moved 1 byte at a time.

The operation starts with the leftmost byte of both fields and proceeds to the right. Each result byte is stored immediately after the necessary operand byte is fetched. The operation ends when the number of bytes specified by bit positions 24-31 of register R_3 is moved.

Execution of the instruction is interruptible between units of operation. Conceptually, after each unit of operation, including the last, the operand addresses in registers R_1 and R_2 are both increased by 1 and the count in bit positions 24-31 of register R_3 is decremented by 1. When, prior to the last unit of

operation, an I/O interruption occurs, or a system-check interruption occurs due to a channel I/O check, the operand addresses and the count are updated so that the instruction, when re-executed, resumes at the point of interruption.

Depending on processor model, more than one unit of operation may be executed between points in the operation at which an interruption is allowed. In this case, the number of units of operation executed without allowing an interruption is predetermined. After each predetermined number of units of operation, the operand addresses and count value are adjusted to correspond to the amount of data moved. The specific predetermined number of units of operation is fixed, except for the first execution group.

The first and second operands are of equal length.

They may be from 1 to 256 bytes. A count of 256 is designated with an initial value of all 0's in bit positions 24-31 of register R_3.

A program exception is indicated when any part of the first- or second-operand field is inaccessible.

The program exception is detected at the time the inaccessible location is referred to, and execution of the instruction is terminated. That is, the contents of registers R_1, R_2, and R_3 may not be adjusted to correspond with the amount of data moved.

Depending on processor model, if R_3 designates the same register as R_1 or R_2, an operation exception may not be indicated and the result is unpredictable.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Termination)
Specification (operand 1 or 2: real address)
Access (operand 1: block invalid, store protection; operand 2: block invalid)
Operation (depending on processor model, $R_3 = R_1$ or R_2)
Separation (operand 1 or 2)
Address (operand 1 or 2: address limit)

Programming Notes
Since execution of MOVE (bytes, storage) is interruptible, the instruction cannot be used for situations in which interruptions are enabled and the program must rely on uninterrupted execution of the instruction. Similarly, the program should normally not let the first operand of the MOVE instruction include the location of the instruction since the new contents of the location may be fetched for a resumption after an interruption.

When R_1 and R_2 specify the same register, data is replaced with itself.

MOVE (halfwords, storage)

MVHS ra,ra,rh

RR-Long Format

Operation

LOOP MS[(R_1)] <— MS[(R_2)]
TEMP1 <— (R_1)
TEMP2 <— (R_2)
TEMP3 <— (R_3<24..31>)
(R_1) <— TEMP1 + 2
(R_2) <— TEMP2 + 2
(R_3<24..31>) <— TEMP3 - 1
If (R_3<24..31>) ≠ 00000000
Then Go To LOOP
Else NSI

Description

The second operand is placed in the first-operand location.

The locations of the leftmost halfwords of the first- and second-operand fields are designated by the contents of the general registers specified by R_1 and R_2, respectively. The length of the first and second operands is designated by the contents of bit positions 24-31 of the general register specified by R_3. The contents of bit positions 0-23 of register R_3 are ignored and remain unchanged. The length is specified in terms of halfwords, and the second-operand field is moved one halfword at a time.

The operation starts with the leftmost halfword of both fields and proceeds to the right. Each result halfword is stored immediately after the necessary operand halfword is fetched. The operation ends when the number of halfwords specified by bit positions 24-31 of register R_3 have been moved.

Execution of the instruction is interruptible between units of operation. Conceptually, after each unit of operation, including the last, the operand addresses in registers R_1 and R_2 are both increased by 2 and the count in bit positions 24-31 of register R_3 is decremented by 1. When, prior to the last unit of operation, an I/O interruption occurs, or a system-check interruption occurs due to a channel I/O check, the operand addresses and the count are updated so that the instruction, when re-executed, resumes at the point of interruption.

Depending on processor model, more than one unit of operation may be executed between points in the operation at which an interruption is allowed. In this case, the number of units of operation executed without allowing an interruption is predetermined. After each predetermined number of units of operation, the operand addresses and count value are adjusted to correspond to the amount of data moved. The specific predetermined number of units of operation is fixed, except for the first execution group.

The first and second operands are of equal length. They may be from 1 to 256 halfwords. A count of 256 is designated with an initial value of all 0's in bit positions 24-31 of register R_3.

A program exception is indicated when any part of the first- or second-operand field is inaccessible.

The program exception is detected at the time the inaccessible location is referred to, and execution of the instruction is terminated. That is, the contents of registers R_1, R_2, and R_3 may not be adjusted to correspond with the amount of data moved.

Depending on processor model, if R_3 designates the same register as R_1 or R_2, an operation exception may not be indicated and the result is unpredictable.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Termination)
Specification (operand 1 or 2: real address)
Access (operand 1: block invalid, store protection;
Operand 2: block invalid)
Operation (depending on processor model, $R_3 = R_1$ or R_2)
Separation (operand 1 or 2)
Address (operand 1 or 2: address limit)

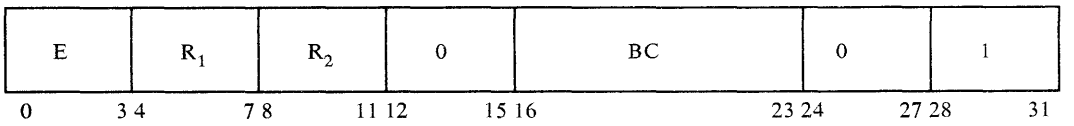
Programming Notes
Since execution of MOVE (halfwords, storage) is interruptible, the instruction cannot be used for situations in which interruptions are enabled and the program must rely on uninterrupted execution of the instruction. Similarly, the program should normally not let the first operand of the MOVE instruction include the location of the instruction since the new contents of the location may be fetched for a resumption after an interruption.

MOVE (halfwords, storage) can be used to move a byte string when the first and second operands begin on halfword boundaries and an even number of bytes is to be moved.

When R_1 and R_2 specify the same register, data is replaced with itself.

MULTIPLY (halfword, register)

MHR rh,rh



RR-Long Format

Operation
If R_1 = xx0x (where x can be 0 or 1)
Then (R_1<16..31>) | | ({R_1+0010}<16..31>) <— ({R_1+0010}<16..31>) x (R_2<16..31>)
Else (R_1<16..31>) <— (R_1<16..31>) x (R_2<16..31>)

Description
The product of the multiplier (second operand) and the multiplicand (first operand) replaces the multiplicand.

Both multiplier and multiplicand are unsigned 16-bit positive binary integers. The product is an unsigned 32-bit positive binary integer.

The high-order half and low-order half of the product occupies the two low-order halfwords, respectively, of the even-odd pair of consecutive registers designated by R_1. The multiplicand is taken from the low-order halfword of the odd register. The contents of the even register replaced by the product are ignored, unless the register contains the multiplier. The low-order halfword of the register specified by R_2 contains the multiplier. The high-order halfwords of the registers in which the operands are located do not participate in the operation and remain unchanged.

Because the multiplicand is replaced by the product, the R_1 field must designate an even register in order to retain the 32-bit product. If R_1 designates an odd register, however, only the low-order 16 bits of the product are retained. The high-order 16 bits of the product are lost and a program exception for overflow, if it occurs, is not indicated. When R_1 designates an even register, an overflow cannot occur.

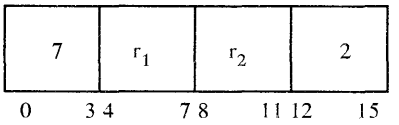
Bit positions 24-27 of the instruction are reserved and must contain all 0's; otherwise, an operation exception is indicated.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (bits 24-27 of instruction not all 0's)

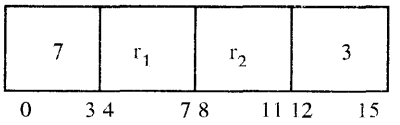
OR (byte, register)

OR rpb,rpb



RR Format
(primary-register-set operands)

OR rsb,rsb



RR Format
(secondary-register-set operands)

Operation
If $r_2 \neq 0000$
  Then $(r_1) \leftarrow (r_1)$ v (r_2)
  Else $(r_1) \leftarrow (r_1)$ v 00000000

Description
The OR of the first- and second-operand bytes is placed in the first-operand location.

The operands are treated as unstructured logical quantities, and the connective OR is applied bit by bit. A bit position in the result is set to 1 if the corresponding bit position in one or both operands contains a 1; otherwise, the result bit is set to 0.

An implied second-operand byte of all 0's is used in place of the register contents when the r_2 field of the instruction is all 0's.

The operands are located in the same register set, designated with bit positions 12-15 of the instruction: hexadecimal "2" designates the primary set, and hexadecimal "3" designates the secondary set.

Result Conditions
8 Result is all 0's.
4 Result is all 1's.
2 Result is mixed 0's and 1's.
1 --
0 --

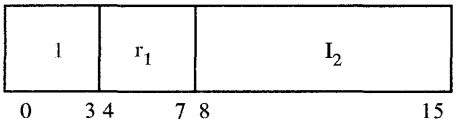
Program Exceptions
None

Programming Notes
This instruction may be used to set a bit to 1.

The byte in bit positions 16-23 of register 0 in the primary and secondary register sets can be designated only as the first operand; it cannot be designated as the second operand.

OR (byte, register-immediate)

ORI rpb,i8



RI Format

Operation
$(r_1) \leftarrow (r_1) \vee I_2$

Description
The OR of the first-operand byte and the byte of immediate data is placed in the first-operand location.

The operands are treated as unstructured logical quantities, and the connective OR is applied bit by bit. A bit position in the result is set to 1 if the corresponding bit position in one or both operands contains a 1; otherwise, the result bit is set to 0.

The first operand is located in the primary register set.

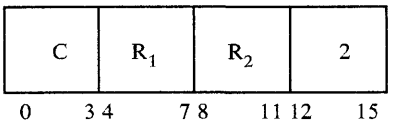
Result Conditions
8 Result is all 0's.
4 Result is all 1's.
2 Result is mixed 0's and 1's.
1 --
0 --

Program Exceptions
None

Programming Note
This instruction may be used to set a bit to 1.

OR (halfword, register)

OHR rh,rh



RR Format

Operation

$(R_1<16..31>) \leftarrow (R_1<16..31>) \vee (R_2<16..31>)$

Description

The OR of the first- and second-operand halfwords is placed in the first-operand location.

The operands are treated as unstructured logical quantities, and the connective OR is applied bit by bit. A bit position in the result is set to 1 if the corresponding bit position in one or both operands contains a 1; otherwise, the result bit is set to 0.

The operands occupy the low-order 16 bits of the registers specified by the R_1 and R_2 fields.

Result Conditions

8 Result is all 0's.
4 Result is all 1's.
2 Result is mixed 0's and 1's.
1 --
0 --

Program Exceptions

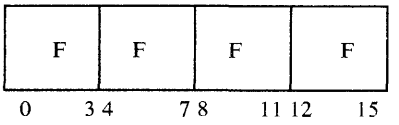
None

Programming Note

This instruction may be used to set a bit to 1.

PROGRAM EXCEPTION

PC



Operation

Operation Exception

Description

The PROGRAM EXCEPTION instruction will always cause a program exception to occur. If at any time an attempt is made to execute a halfword consisting of all ones, a program exception will result.

The instruction does not require any operands.

Result Conditions

The conditions remain unchanged.

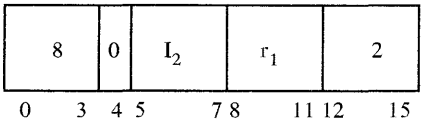
Program Exceptions (Suppression)
Operation

Programming Notes
The programmer can use this instruction as a known program exception, that is, as a forced termination of a program at a known location.

The results of this instruction are not different than if any undefined instruction execution is attempted.

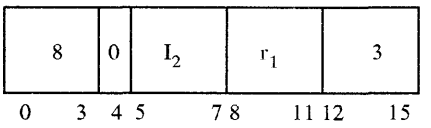
ROTATE LEFT (byte)

RL rpb,c3



RR Format
(primary-register-set operands)

RL rsb,c3



RR Format
(secondary-register-set operands)

Operation
(r_1) <— (r_1) rotated left by I_2 amount

Description
The first-operand byte is rotated left the number of bits specified by the I_2 field.

The I_2 field is an unsigned 3-bit positive binary integer. All 8 bits of the first operand participate in the operation. Bits rotated out of the high-order bit position of the operand are entered into the vacated low-order operand positions. An I_2 field of all 0's designates a 0 rotation amount, and the indicated result conditions are based on the contents of the operand.

The first operand is located in either the primary or secondary register set that is designated with bit positions 12-15 of the instruction: hexadecimal "2" designates the primary set, and hexadecimal "3" designates the secondary set.

Bit position 4 of the instruction is used as an extension to the operation code. The bit distinguishes this instruction from LOAD (halfwords, quadrant) and STORE (halfwords, quadrant).

Result Conditions
8 Result is all 0's.
4 Result has a 1 in the high-order bit position.
2 Result has a 0 in the high-order bit position and one or more 1's in the remaining bit positions.
1 One or more 1's were rotated out of the high-order bit position of the operand.
0 --

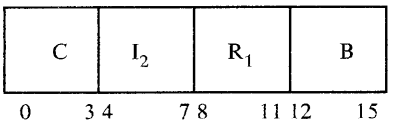
Program Exceptions
None

Programming Notes
The rotation amount (I_2) may be an integer number from 0 to 7. Specifying the maximum rotation amount effectively rotates the operand one bit position to the right.

Bit positions 5-7 and 8-11 of the instruction format contain the I_2 and r_1 fields, respectively. This is reversed from the normal left-to-right order for the RR instruction format.

ROTATE LEFT (halfword)

RLH rh,c4



RR Format

Operation
(R_1<16..31>) <— (R_1<16..31>) rotated left by I_2 amount

Description
The first-operand halfword is rotated left the number of bits specified by the I_2 field.

The I_2 field is an unsigned 4-bit positive binary integer. All 16 bits of the first operand participate in the operation. Bits rotated out of the high-order bit position of the operand are entered into the vacated low-order operand positions. An I_2 field of all 0's designates a 0 rotation amount, and the indicated result conditions are based on the contents of the operand.

The first operand occupies the low-order halfword of the register specified by R_1.

Result Conditions
8 Result is all 0's.
4 Result has a 1 in the high-order bit position.
2 Result has a 0 in the high-order bit position and one or more 1's in the remaining bit positions.
1 One or more 1's were rotated out of the high-order bit position of the operand.
0 --

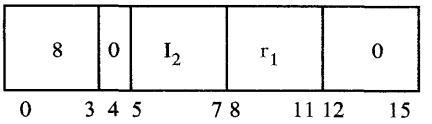
Program Exceptions
None

Programming Notes
The rotation amount (I_2) may be an integer number from 0 to 15. Specifying the maximum rotation amount effectively rotates the operand one bit position to the right.

Bit positions 4-7 and 8-11 of the instruction format contain the I_2 and R_1 fields, respectively. This is reversed from the normal left-to-right order for the RR instruction format.

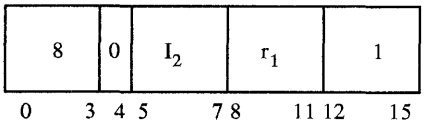
SHIFT LEFT (byte, logical)

SLL rpb,c3



RR Format
(primary-register-set operands)

SLL rsb,c3



RR Format
(secondary-register-set operands)

Operation
(r_1) <— (r_1) shifted left by I_2 amount

Description
The first-operand byte is shifted left the number of bits specified by the I_2 field.

The I_2 field is an unsigned 3-bit positive binary integer. All 8 bits of the first operand participate in the operation. Bits shifted out of the high-order bit position of the operand are lost. Zeros are supplied to the vacated low-order operand positions. An I_2 field of all 0's designates a 0 shift amount, and the indicated result conditions are based on the contents of the operand.

The first operand is located in either the primary or secondary register set, designated with bit positions 12-15 of the instruction: hexadecimal "0" designates the primary set, and hexadecimal "1" designates the secondary set.

Bit position 4 of the instruction is reserved and must contain a 0; otherwise, an operation exception is indicated.

Result Conditions
8 Result is all 0's.
4 Result has a 1 in the high-order bit position.
2 Result has a 0 in the high-order bit position and one or more 1's in the remaining bit positions.
1 One or more 1's were shifted out of the high-order bit position of the operand.
0 --

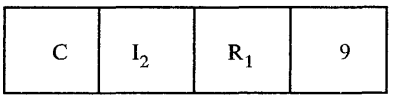
Program Exceptions (Suppression)
Operation (bit 4 of instruction is 1)

Programming Notes
The shift amount (I_2) may be an integer number from 0 to 7. Specifying the maximum shift amount shifts the low-order bit of the operand to the high-order bit position of the result.

Bit positions 5-7 and 8-11 of the instruction format contain the I_2 and r_1 fields, respectively. This is reversed from the normal left-to-right order for the RR instruction format.

SHIFT LEFT (halfword, logical)

SLHL rh,c4



RR Format

Operation

(R_1<16..31>) <— (R_1<16..31>) shifted left by I_2 amount

Description

The first-operand halfword is shifted left the number of bits specified by the I_2 field.

The I_2 field is an unsigned 4-bit positive binary integer. All 16 bits of the first operand participate in the operation. Bits shifted out of the high-order bit position of the operand are lost. Zeros are supplied to the low-order operand positions. An I_2 field of all 0's designates a 0 shift amount, and the indicated result conditions are based on the contents of the operand.

The first operand occupies the low-order halfword of the register specified by R_1.

Result Conditions

8 Result is all 0's.
4 Result has a 1 in the high-order bit position.
2 Result has a 0 in the high-order bit position and one or more 1's in the remaining bit positions.
1 One or more 1's were shifted out of the high-order bit position of the operand.
0 --

Program Exceptions

None

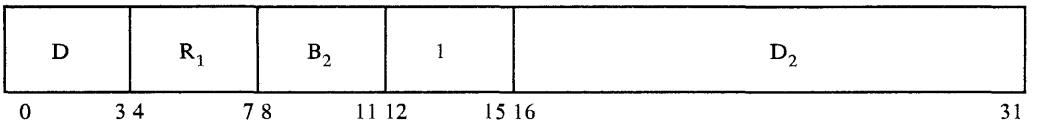
Programming Notes

The shift amount (I_2) may be an integer number from 0 to 15. Specifying the maximum shift amount shifts the low-order bit of the operand to the high-order bit position of the result.

Bit positions 4-7 and 8-11 of the instruction format contain the I_2 and R_1 fields, respectively. This is reversed from the normal left-to-right order for the RR instruction format.

STORE (byte)

ST rpb,db16s(ra)



RS-Long Format

Operation

If $B_2 \neq 0000$
  Then $MS[(B_2) + D_2] \leftarrow (r_1)$
  Else $MS[IA + D_2] \leftarrow (r_1)$

Description

The first-operand byte is stored unchanged at the second-operand location.

When the B_2 field contains all 0's, the second-operand address is computed using the updated instruction address in place of the contents of primary general register 0.

The first operand is located in the primary register set.

Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)

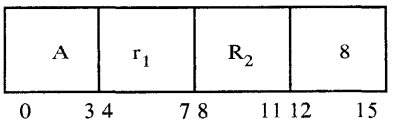
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Separation (operand 2)
Address (operand 2: all)

Programming Note

Relative to the base address for the second-operand location, the range, in bytes, covered by the displacement (D_2) is -32768 * D_2 * 32767.

STORE (byte, with index)

STN rpb,ra



RR Format

Operation

$MS[(R_2)] \leftarrow (r_1)$

Description

The first-operand byte is stored unchanged at the second-operand location.

The contents of the general register specified by the R_2 field are used as the second-operand address.

The first operand is located in the primary register set.

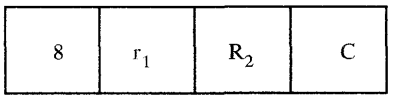
Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Separation (operand 2)
Address (operand 2: address limit)

STORE (byte, with index decremented)

STND rpb,ra



RR Format

Operation
TEMP <— (r_1)
(R_2) <— (R_2) - 1
MS[(R_2)]<— TEMP

Description
The contents of the general register designated by the R_2 field are decremented by 1, and the result is used as the second-operand address. The first-operand byte is then stored unchanged in the second-operand location.

The contents of the general register specified by R_2 are decremented by 1 after the byte is fetched from the first operand location and before it is placed in main storage.

Decrementing the contents of register R_2 through 0 causes a wraparound to 4,294,967,295 (hex FFFF FFFF).

Program exceptions pertain to the *decremented* second-operand address and are indicated only when the decremented second-operand field is inaccessible. Detection of the program exception occurs when a reference to the inaccessible location is attempted. The execution of the instruction is *suppressed;* that is, the second-operand address and main storage remain unchanged.

The first operand is located in the primary register set.

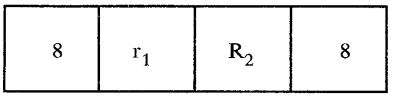
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Separation (operand 2)
Address (operand 2: address limit)

Programming Note
When the first operand is located in the general register specified by R_2, the initial contents of the first operand are placed in main storage.

STORE (byte, with index incremented)

STNI rpb,ra



RR Format

Operation
$MS[(R_2)] \leftarrow (r_1)$
$(R_2) \leftarrow (R_2) + 1$

Description
The first-operand byte is stored unchanged in the second-operand location. The contents of the general register designated by the R_2 field are then incremented by 1.

The initial contents of the general register specified by R_2 are used as the second-operand address. The contents of the general register specified by R_2 are incremented by 1 after the byte is placed in main storage.

Program exceptions pertain to the *initial* second-operand address and are indicated only when the initial second-operand field is inaccessible. Detection of the program exception occurs when a reference to the inaccessible location is attempted. The execution of the instruction is *suppressed;* that is, the second-operand address and main storage remain unchanged.

The first operand is located in the primary register set.

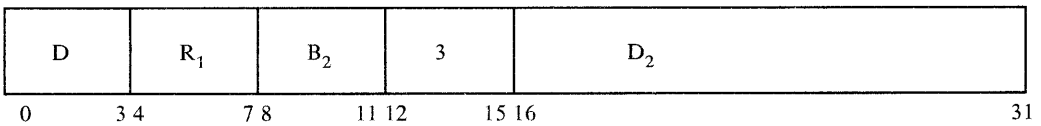
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Separation (operand 2)
Address (operand 2: address limit)

Programming Note
When the first operand is located in the general register specified by R_2, the initial contents of the first operand are placed in main storage.

STORE (halfword)

STH rh,db16s(ra)



RS-Long Format

Operation
If $B_2 \neq 0000$
Then $MS[(B_2) + D_2] \leftarrow (R_1<16..31>)$
Else $MS[(IA) + D_2] \leftarrow (R_1<16..31>)$

Description
The first-operand halfword is stored unchanged in the second-operand location.

When the B_2 field contains all 0's, the second-operand address is computed using the updated instruction address in place of the contents of primary general register 0.

The first operand occupies the low-order halfword of the general register specified by R_1.

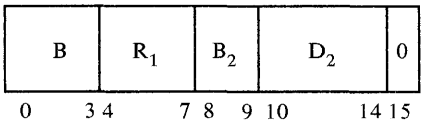
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Separation (operand 2)
Address (operand 2: all)

Programming Note
Relative to the base address for the second-operand location, the range, in bytes, covered by the displacement (D_2) is $-32768 \leq D_2 \leq 32767$.

STORE (halfword, short form)

STHS rh,dh5(ra)



RS Format

Operation
$MS[(B_2) + D_2 \times 2] \leftarrow (R_1\langle 16..31\rangle)$

Description
The first-operand halfword is stored unchanged in the second-operand location.

The first operand occupies the low-order halfword of the general register specified by R_1.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Separation (operand 2)
Address (operand 2: address limit)

Programming Notes
The short form of the STORE (halfword) instruction is provided to conserve program space. It can be used for base-plus-displacement addressing of data structures that comprise up to 32 contiguous halfwords.

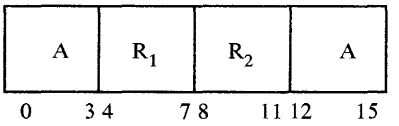
The contents of the B_2 field represent the 2 low-order bits of a 4-bit R field in which the 2 high-order bits are both 1's. The specification of the base register is therefore limited to primary general register 6 or 7, or secondary general register 6 or 7, as indicated by the following chart.

B_2 Field Bits 8, 9	Register Specified
0 0	Register 6, Primary Set
0 1	Register 6, Secondary Set
1 0	Register 7, Primary Set
1 1	Register 7, Secondary Set

Relative to the base address for the second-operand location, the range, in bytes, covered by the displacement (D_2) is $\times 0 \leq D_2 \times 2 \leq 62$. Note that the displacement can be specified only in terms of an even number of bytes.

STORE (halfword, with index)

STHN rh,ra



RR Format

Operation
MS[(R_2)] <— (R_1<16..31>)

Description
The first-operand halfword is stored unchanged at the second-operand location.

The contents of the general register specified by the R_2 field are used as the second-operand address.

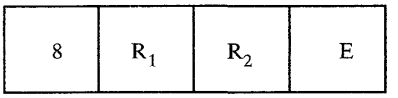
The first operand occupies the low-order halfword of the general register specified by R_1.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Separation (operand 2)
Address (operand 2: address limit)

STORE (halfword, with index decremented)

STHND rh,ra



RR Format

Operation

TEMP<— (R_1<16..31>)
(R_2)<— (R_2) - 2
MS[(R_2)]<— TEMP

Description

The contents of the general register designated by the R_2 field are decremented by 2, and the result is used as the second-operand address. The first-operand halfword is then stored unchanged in the second-operand location.

The contents of the general register specified by R_2 are decremented by 2 after the halfword is fetched from the first operand location and before it is placed in main storage.

Decrementing the contents of register R_2 through 0 causes a wraparound to 4,294,967,295 (hex FFFF FFFF).

Program exceptions pertain to the *decremented* second-operand address and are indicated only when the decremented second-operand field is inaccessible. Detection of the program exception occurs when a reference to the inaccessible location is attempted. The execution of the instruction is *suppressed;* that is, the second-operand address and main storage remain unchanged.

The first operand occupies the low-order halfword of the general register specified by R_1.

Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)

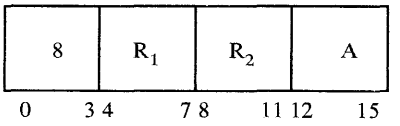
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Separation (operand 2)
Address (operand 2: address limit)

Programming Note

When the first operand is located in the general register specified by R_2, the initial contents of the first operand are placed in main storage.

STORE (halfword, with index incremented)

STHNI rh,ra



RR Format

Operation
MS[(R_2)]<— (R_1<16..31>)
(R_2) <— (R_2) + 2

Description
The first-operand halfword is stored unchanged in the second-operand location. The contents of the general register designated by the R_2 field are then incremented by 2.

The initial contents of the general register specified by R_2 are used as the second-operand address. The contents of the general register specified by R_2 are incremented by 2 after the halfword is placed in main storage.

Program exceptions pertain to the *initial* second-operand address and are indicated only when the initial second-operand field is inaccessible. Detection of the program exception occurs when a reference to the inaccessible location is attempted. The execution of the instruction is *suppressed;* that is, the second-operand address and main storage remain unchanged.

The first operand occupies the low-order halfword of the register specified by R_1.

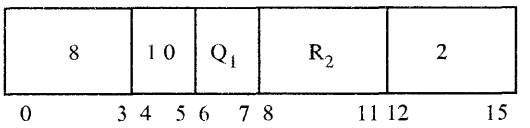
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Separation (operand 2)
Address (operand 2: address limit)

Programming Note
When the first operand is located in the general register specified by R_2, the initial contents of the first operand are placed in main storage.

STORE (halfwords, quadrant)

STHQ q2,ra



RR Format

Operation
MS[(R_2)]<— RQ<Q_1>
(R_2) <— (R_2) + 16

Description
The eight consecutive general-register halfword fields specified by the first operand (Q_1) are stored unchanged in the main storage locations designated by the second-operand address.

The contents of the register designated by R_2 are used as the second-operand address. At the completion of the operation, the second-operand address is increased by 16, and the updated address is placed back in the register specified by R_2.

The main storage area in which the halfwords are placed starts at the location designated by the second-operand address and includes eight consecutive halfword locations. The general-register halfword fields are stored in ascending order beginning with the first (lowest numbered) register of the set indicated by Q_1.

The register quadrant designated by Q_1 consists of the eight high-order or low-order halfwords of the general registers that make up the primary or secondary register set, as shown in the following table:

	Register Quadrant	
Q_1 Operand	**Register Set**	**Halfword Fields**
00	primary	low-order <16..31>
01	secondary	low-order <16..31>
10	primary	high-order <0..15>
11	secondary	high-order <0..15>

When any part of the second operand is inaccessible, a specification, access, separation, or address exception is indicated at the time the inaccessible location is referred to, and execution is suspended. That is, the initial second-operand address in register R_2 remains unchanged; the original contents of the main-storage locations into which the fields are stored, if any, are lost. Valid retry of the instruction is possible.

Bit position 4 of the instruction is used as an extension to the operation code. The bit distinguishes this instruction from ROTATE LEFT (byte). Bit position 5 of the instruction is reserved and must contain a 0; otherwise, an operation exception is indicated, and the operation is suppressed.

Result Conditions
The conditions remain unchanged.

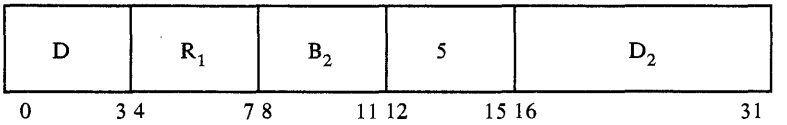
Program Exceptions (Suspension/Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Operation (bit 5 of instruction is a 1)
Separation (operand 2)
Address (operand 2: address limit)

Programming Notes
When the register designated by R_2 is in the register set indicated by Q_1, the initial contents of register R_2 are placed in main storage.

When the second-operand location (eight consecutive halfwords) is completely unavailable, execution of this instruction is *suppressed* due to the specification, access, separation, or address exception. Therefore, suspension of the operation can be avoided by locating the second operand starting at an address that is an integral multiple of 16.

STORE (word)

STU rw,db16s(ra)



RS-Long Format

Operation

If $B_2 \neq 0000$
  Then $MS[(B_2) + D_2] \leftarrow (R_1)$
  Else $MS[IA + D_2] \leftarrow (R_1)$

Description

The first operand is stored unchanged in the second-operand location.

When the B_2 field contains all 0's, the second-operand address is computed using the updated instruction address in place of the contents of primary general register 0.

Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)

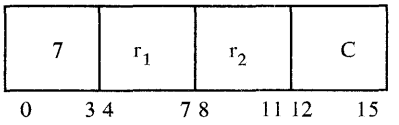
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Separation (operand 2)
Address (operand 2: all)

Programming Note

Relative to the base address for the second-operand location, the range, in bytes, covered by the displacement (D_2) is $-32768 \leq D_2 \leq 32767$.

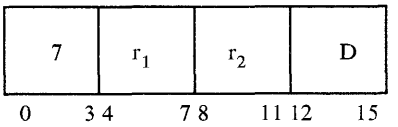
SUBTRACT (byte, register)

SR rpb,rpb



RR Format (primary-register-set operands)

SR rsb,rsb



RR Format (secondary-register-set operands)

Operation

If $r_2 \neq 0000$
  Then $(r_1) \leftarrow (r_1) + \neg(r_2) + 1$
  Else $(r_1) \leftarrow (r_1) + \neg 00000000 + 1$

Description
The second-operand byte is subtracted from the first-operand byte, and the difference is placed in the first-operand location. Subtraction is performed by adding the 1's complement of all 8 bits of the second operand and a low-order 1 to all 8 bits of the first operand. The two operands are considered to be signed fixed-point numbers.

An implied second operand byte of all 0's is used in place of the register contents when the r_2 field of the instruction is all 0's.

The operands are located in the same register set, designated with bit positions 12-15 of the instruction: hexadecimal "C" designates the primary set, and hexadecimal "D" designates the secondary set.

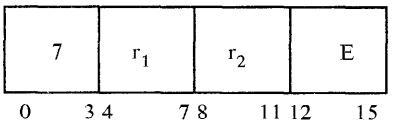
Result Conditions
8 Difference is 0.
4 Difference is less than 0.
2 Difference is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions
None

Programming Note
The byte in bit positions 16-23 of register 0 in the primary and secondary register sets can be designated only as the first operand; it cannot be designated as the second operand.

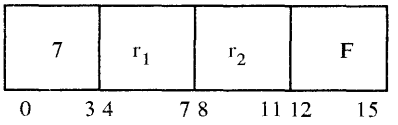
SUBTRACT WITH CARRY (byte, register)

SYR rpb,rpb



RR Format
(primary-register-set operands)

SYR rsb,rsb



RR Format
(secondary-register-set operands)

Operation
If $r_2 \neq 0000$
Then $(r_1) \leftarrow (r_1) + \neg(r_2) + C$
Else $(r_1) \leftarrow (r_1) + \neg 00000000 + C$

Description
The second-operand byte is subtracted from the first-operand byte, and the difference is placed in the first-operand location. Bit 56 (the carry-condition indicator) in the current PSV participates in the subtraction.

Subtraction is performed by adding the 1's complement of all 8 bits of the second operand and a low-order 0 or 1 taken from bit 56 (C) in the current PSV, to all 8 bits of the first operand. Algebraically, a borrow from the first operand occurs

(due to the previous subtract operation) when PSV-bit 56 is 0; no borrow occurs when PSV-bit 56 is 1. The two operands are considered to be signed fixed-point numbers.

An implied second operand byte of all 0's is used in place of the register contents when the r_2 field of the instruction is all 0's.

The operands are located in the same register set, designated with bit positions 12-15 of the instruction: hexadecimal "E" designates the primary set, and hexadecimal "F" designates the secondary set.

Result Conditions
8 Extended difference is 0.
4 Extended difference is less than 0.
2 Extended difference is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions
None

Programming Notes
The SUBTRACT WITH CARRY instructions are provided for subtraction of extended fixed-point numbers. A carry from any SUBTRACT instruction is accounted for by executing a subsequent SUBTRACT WITH CARRY instruction without executing an intervening instruction that changes the indicated result conditions.

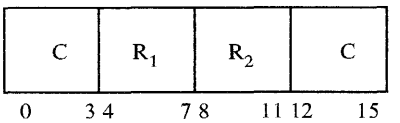
SUBTRACT WITH CARRY (byte, register) can be used to account for only the borrow, if any, due to the previous SUBTRACT by specifying an implied second-operand byte of all 0's.

Result condition 8 can be indicated (reflecting a result of 0) only if it was indicated at the beginning of the operation.

The byte in bit positions 16-23 of register 0 in the primary and secondary register sets can be designated only as the first operand; it cannot be designated as the second operand.

SUBTRACT (halfword, register)

SHR rh,rh



RR Format

Operation
$(R_1<16..31>) \leftarrow (R_1<16..31>) + \neg(R_2<16..31>) + 1$

Description
The second-operand halfword is subtracted from the first-operand halfword, and the difference is placed in the first-operand location. Subtraction is performed by adding the 1's complement of all 16 bits of the second operand and a low-order 1 to all 16 bits of the first operand. The two operands are considered to be signed fixed-point numbers.

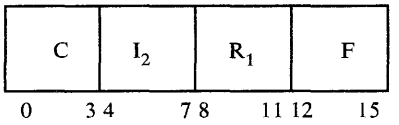
The operands occupy the low-order halfwords of the general registers specified by the R_1 and R_2 fields.

Result Conditions
8 Difference is 0.
4 Difference is less than 0.
2 Difference is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions
None

SUBTRACT (halfword, register-immediate)

SHRI rh,i4



RR Format

Operation
$(R_1<16..31>) \leftarrow (R_1<16..31>) + \neg 000000000000 \,||\, I_2 + 1$

Description
The 4 bits of immediate data, I_2, are subtracted from the first-operand halfword, and the difference is placed in the first-operand location. The immediate operand is treated as an unsigned 4-bit positive binary integer. The first operand is considered to be a signed fixed-point number.

Subtraction is considered to be performed by first expanding the immediate operand to 16 bits with 12 high-order 0's. Then, the 1's complement of all 16 bits of the expanded immediate operand and a low-order 1 are added to all 16 bits of the first operand.

The first operand occupies the low-order halfword of the general register specified by the R_1 field.

Result Conditions
8 Difference is 0.
4 Difference is less than 0.
2 Difference is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

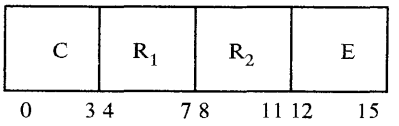
Program Exceptions
None

Programming Notes
The immediate operand can range in value from 0 to 15, inclusive.

Bit positions 4-7 and 8-11 of the instruction format contain the I_2 and R_1 fields, respectively. This is reversed from the normal left-to-right order for the RR instruction format.

SUBTRACT WITH CARRY (halfword, register)

SYHR rh,rh



RR Format

Operation

$(R_1<16..31>) \leftarrow (R_1<16..31>) + \neg(R_2<16..31>) + C$

Description

The second-operand halfword is subtracted from the first-operand halfword, and the difference is placed in the first-operand location. Bit 56 (the carry-condition indicator) in the current PSV participates in the subtraction.

Subtraction is performed by adding the 1's complement of all 16 bits of the second operand and a low-order 0 or 1 taken from bit 56 (C) in the current PSV, to all 16 bits of the first operand. Algebraically, a borrow from the first operand occurs (due to the previous subtract operation) when PSV-bit 56 is 0; no borrow occurs when PSV-bit 56 is 1. The two operands are considered to be signed fixed-point numbers.

The operands occupy the low-order halfwords of the registers specified by the R_1 and R_2 fields.

Result Conditions

8 Extended difference is 0.
4 Extended difference is less than 0.
2 Extended difference is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions

None

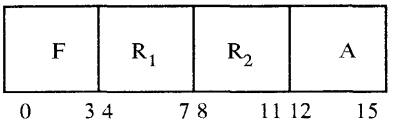
Programming Notes

The SUBTRACT WITH CARRY instructions are provided for subtraction of extended fixed-point numbers. A carry from any SUBTRACT instruction is accounted for by executing a subsequent SUBTRACT WITH CARRY instruction without executing an intervening instruction that changes the indicated result conditions.

Result condition 8 can be indicated (reflecting a result of 0) only if it was indicated at the beginning of the operation.

SUBTRACT WITH CARRY (halfword, register, extended)

SYHRE ruh,ruh



RR Format

Operation

If $R_2 \neq 0000$

Then $(R_1<0..15>) \leftarrow (R_1<0..15>) + \neg(R_2<0..15>) + C$

Else $(R_1<0..15>) \leftarrow (R_1<0..15>) + \neg 0000000000000000 + C$

Description

The second-operand halfword is subtracted from the first-operand halfword, and the difference is placed in the first-operand location. Bit 56 (the carry-condition indicator) in the current PSV participates in the subtraction.

Subtraction is performed by adding the 1's complement of all 16 bits of the second operand and a low-order 0 or 1 taken from bit 56 (C) in the current PSV, to all 16 bits of the first operand. Algebraically, a borrow from the first operand occurs (due to the previous subtract operation) when PSV-bit 56 is 0; no borrow occurs when PSV-bit 56 is 1. The two operands are considered to be signed fixed-point numbers.

An implied second-operand halfword of all 0's is used in place of the register contents when the R_2 field of the instruction is all 0's.

The operands occupy the high-order halfwords of the registers specified by the R_1 and R_2 fields.

Result Conditions

8 Extended difference is 0.
4 Extended difference is less than 0.
2 Extended difference is greater than 0.
1 Overflow.
0 Carry out of sign-bit position.

Program Exceptions

None

Programming Notes

The SUBTRACT WITH CARRY instructions are provided for subtraction of extended fixed-point numbers. A carry from any SUBTRACT instruction is accounted for by executing a subsequent SUBTRACT WITH CARRY instruction without executing an intervening instruction that changes the indicated result conditions.

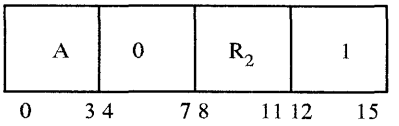
SUBTRACT WITH CARRY (halfword, register, extended) can be used following a SUBTRACT (halfword, register) instruction to perform word register-to-register subtraction. This instruction can also be used to account for only the borrow, if any, due to the previous SUBTRACT by specifying an implied second-operand of all 0's.

Result condition 8 can be indicated (reflecting a result of 0) only if it was indicated at the beginning of the operation.

The halfword in bit positions 0-15 of register 0 in the primary register set can be designated only as the first operand; it cannot be designated as the second operand.

TEST AND SET (byte)

TS 0,ra



RR Format

Operation
Result-Conditions <== MS[(R_2)]
MS[(R_2)] <— 11111111

Description
The indicated result conditions are determined by the contents of the byte located at the second-operand address, and then the entire addressed byte is set to all 1's.

The byte in storage is set to all 1's immediately after it is fetched for testing. An interruption is not allowed between the moment of fetching and the moment of storing all 1's. Additionally, in dual-PCE processors, accessing of this byte in storage by another PCE is not allowed between the fetching and storing operation.

Instruction bits 4-7 are reserved and must be all 0's; otherwise, an operation exception is indicated.

Result Conditions
8 Byte tested is all 0's.
4 Byte tested is all 1's.
2 Byte tested is mixed 0's and 1's.
1 --
0 --

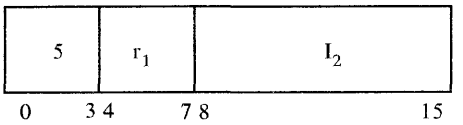
Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Operation (bits 4-7 of instruction not all 0's)
Separation (operand 2)
Address (operand 2: address limit)

Programming Notes
TEST AND SET can be used for controlled sharing of a common storage area by two or more programs in one or two PCEs. The interlock can be achieved by establishing a program convention. For example, an all-0's value in the byte indicates that the common area is available, and an all-1's value means that the area is being used. Each using program then must examine the byte by means of TEST AND SET before making access to the common area. If result condition 8 or 2 is indicated after the test, the area is available for use; if condition 4 is indicated, the area cannot be used.

Because TEST AND SET does not permit an interruption or a storage reference from another PCE to occur between the moment of fetching (for testing) and the moment of storing all 1's (setting), the possibility of a second program testing the byte before the first program is able to set it is eliminated.

TEST (byte, register-immediate)

TRI rpb,i8



RI Format

Operation
Result-Conditions <== (r_1) tested using mask I_2

Description
The state of the first-operand bits selected by a mask is used to determine the indicated result conditions. The first operand remains unchanged.

The byte of immediate data, I_2, is used as an 8-bit mask. The bits of the mask are made to correspond one for one with the bits of the first operand. A mask bit of 1 indicates that the corresponding operand bit is to be tested. A mask bit of 0 indicates that the corresponding operand bit is to be ignored.

When all operand bits thus selected are 0, result condition 8 is indicated. Condition 8 is also indicated when the mask is all 0's. When the selected bits are all 1's, result condition 4 is indicated; otherwise, result condition 2 is indicated. In addition, when the first-operand byte is identical to the mask, result condition 1 is indicated.

The first operand is located in the primary register set.

Result Conditions
8 Selected bits are all 0's, or the mask is all 0's.
4 Selected bits are all 1's.
2 Selected bits are mixed 0's and 1's.
1 Mask and first-operand byte are identical.
0 --

Program Exceptions
None

Chapter 5. Floating-Point Instructions

The 8100 system provides a set of floating-point instructions as an optional feature on certain processor models (see "Appendix G"). The floating-point instructions are used to perform calculations on operands with a wide range of magnitude. These instructions yield results scaled to preserve precision.

A floating-point number consists of a signed exponent (represented in the number's format by the characteristic) and a signed fraction. The quantity expressed by this number is the product of the fraction and the number 16 raised to the power of the exponent. The exponent is expressed in excess-64 binary notation (see "Number Representation" in this chapter); the fraction is expressed as a hexadecimal number having a radix point to the left of the high-order digit.

Four floating-point registers are available to the executing program. The floating-point instructions provide for the loading, rounding, adding, subtracting, comparing, multiplying, dividing, storing, and controlling the sign of short and long operands. Short operands generally provide faster processing and require less storage than long operands. On the other hand, long operands provide greater precision in computation. Operations may be either register-to-register or register-and-storage.

For addition, subtraction, multiplication, and division, instructions are provided that generate normalized results. Normalized results preserve the highest precision in the operation (see "Normalization" in this chapter). For addition and subtraction, instructions are also provided that generate unnormalized results. Normalized and unnormalized operands may be used in any floating-point operation.

Result conditions are indicated to reflect the outcome of all sign-control, add, subtract, and compare operations.

Instructions are also provided for setting the precision (short or long) for floating-point operations, and for enabling or disabling exponent-overflow, exponent-underflow, and significance exceptions.

One floating-point feature is allowed on dual-PCE processors and is restricted to a specific PCE.

Data Format

Floating-point data occupies a fixed-length format which may be either a 4-byte (short) format or an 8-byte (long) format (see Figure 5-1). The short and long formats may be designated as operands both in main storage and in the floating-point registers. Operands (both short and long) in main storage must be aligned on a word boundary; that is, the address of the leftmost byte of the operand must be a multiple of 4.

The first bit of both formats is the sign bit (S). The subsequent seven bit positions are occupied by the characteristic. The following field contains the fraction which, depending on the format, consists of 6 or 14 hexadecimal digits.

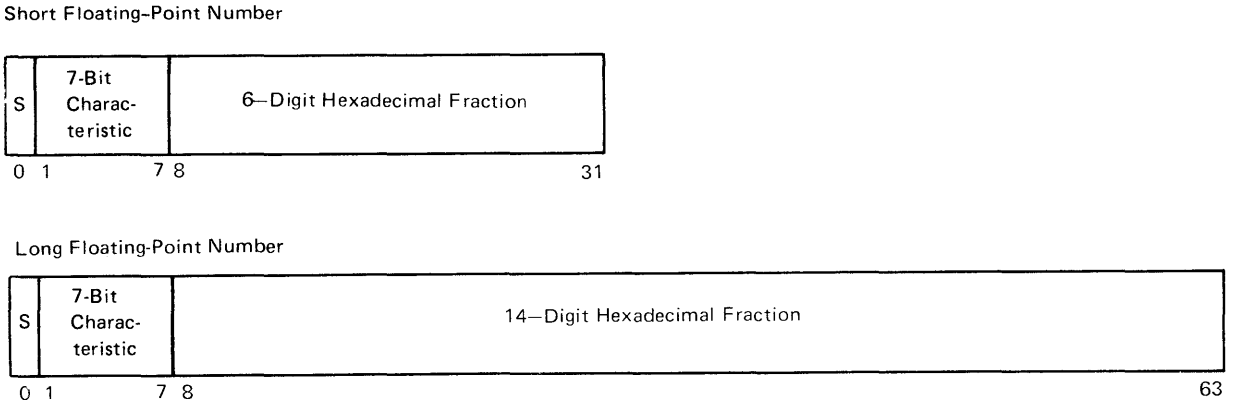


Figure 5-1. Formats of Short and Long Floating-Point Numbers

Short floating-point numbers occupy only the leftmost 32 bit positions of a 64-bit floating-point register. When a floating-point register is used as the source of a short floating-point number, the rightmost 32 bit positions of the register are ignored. When a floating-point register is used as the destination of a short floating-point number, the rightmost 32 bit positions of the register remain unchanged.

The entire set of floating-point arithmetic, load, and store instructions is available for short and long operands. These instructions generate a result that has the same format as the operands, except for MULTIPLY and LOAD ROUNDED. In the case of short MULTIPLY, a long product is produced from a short multiplier and multiplicand. LOAD ROUNDED provides for rounding from long to short format.

Guard Digit

Although final results of floating-point calculations have 6 fraction digits in the short format and 14 fraction digits in the long format, intermediate results in ADD NORMALIZED, SUBTRACT NORMALIZED, ADD UNNORMALIZED, SUBTRACT UNNORMALIZED, COMPARE, and MULTIPLY have one additional low-order digit. This low-order digit, the *guard digit,* increases the precision of the intermediate result. The final result is obtained when the intermediate result, including the guard digit, is shifted left as necessary and then truncated to the result-fraction length.

Number Representation

The fraction of a floating-point number is expressed in hexadecimal digits with the radix point of the fraction assumed to be immediately to the left of the high-order fraction digit. The fraction is considered to be multiplied by the power of 16 indicated by the characteristic portion, bits 1-7, of the floating-point formats. The bits within the characteristic field can represent numbers from 0 through 127. To accommodate large and small magnitudes, the characteristic is formed by adding 64 to the actual exponent. The range of the exponent is thus -64 through +63. This technique produces a characteristic in excess 64 notation.

Both positive and negative quantities have a true hexadecimal representation of the fraction, the sign being indicated by the sign bit. The number is positive or negative, depending on whether the sign bit is 0 or 1, respectively.

The range covered by the magnitude (M) of a normalized floating-point number is:

In the short format:

$$16^{-65} \leq M \leq (1-16^{-6}) \times 16^{63}$$

In the long format:

$$16^{-65} \leq M \leq (1-16^{-14}) \times 16^{63}$$

In both formats, approximately:

$$5.4 \times 10^{-79} \leq M \leq 7.2 \times 10^{75}$$

A number with a 0 characteristic, 0 fraction, and plus sign is called a true zero. A true zero may arise as the result of an arithmetic operation because of the particular magnitude of the operands. A result is forced to be true zero for any one of the following:

- An exponent underflow occurs and the exponent-underflow mask bit in the floating-point status vector is 1.
- The result fraction of an addition or subtraction operation is 0 and the significance mask bit in the floating-point status vector is 1.
- One or both operands of MULTIPLY, or the dividend in DIVIDE, has a 0 fraction.

When a program-exception interruption due to exponent underflow occurs, a true zero fraction is not forced; instead, the fraction and sign remain correct, and the characteristic is 128 too large. When a program-exception interruption due to the significance exception occurs, the fraction remains 0, the sign is positive, and the characteristic remains correct. When an exponent-overflow exception occurs, the fraction and sign remain correct and the characteristic is 128 less than the correct value, regardless of whether the exponent-overflow mask bit in the floating-point status vector is 0 or 1. The exponent-overflow and exponent-underflow exceptions are not recognized when the result has a 0 fraction. When a divisor has a 0 fraction, division is omitted, and a program-exception interruption for a floating-point-divide exception occurs. In addition and subtraction, an operand with a 0 fraction or characteristic participates as a normal number. The sign of a sum, difference, product, or quotient with 0 fraction is positive.

Normalization

A quantity can be represented with the greatest precision by a floating-point number of given fraction length when the number is normalized. A normalized floating-point number has a nonzero high-order hexadecimal fraction digit. If one or more high-order fraction digits are 0, the number is said to be unnormalized. The process of normalization consists of shifting the fraction left, one digit at a time, until the high-order hexadecimal digit is nonzero and reducing the characteristic by the number of hexadecimal digits shifted. A number with a 0 fraction cannot be normalized, and its characteristic therefore remains unchanged when normalization is called for.

Normalization usually takes place when the intermediate arithmetic result is changed to the final result. This function is called *postnormalization.* In performing multiplication and division, the operands are normalized before the arithmetic process. This function is called *prenormalization.*

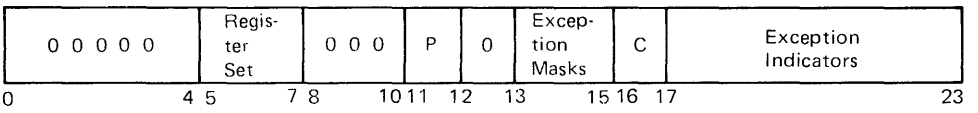
Floating-point arithmetic may be performed with or without normalization. Multiplication and division are performed only with normalization. Addition and subtraction may be specified either way. When unnormalized addition or subtraction is specified, high-order 0's in the result fraction are not eliminated. The result may or may not be normalized, depending upon the original operands.

In both normalized and unnormalized operations, the initial operands need not be in normalized form. Also, intermediate fraction results are shifted right when an overflow occurs, and the intermediate fraction result is truncated to the final result length after the shifting.

Programming Note: Since normalization applies to hexadecimal digits, the 3 high-order bits of the fraction of a normalized number may be 0's.

Floating-Point Status Vector

The floating-point status vector (FSV) is 24 bits long and contains information required for proper execution of floating-point operations. The FSV contains the floating-point register-set number, precision-mode bit, exception-mask bits, floating-point-check bit, and exception-indicator bits. The format of the FSV is:



The functions of the precision-mode, exception-mask, and exception-indicator bits are described in the following paragraphs. The remaining fields of the FSV are described in Chapter 9. The FSV associated with the current priority level is called the current FSV.

Referring to the FSV

Instructions are provided for referring to all of the FSV or to only a part of it. READ FLOATING-POINT STATUS VECTOR and WRITE FLOATING-POINT STATUS VECTOR are used, respectively, to access or change all of the FSV. These two instructions can be used to read or write the FSV associated with any priority level, including the current level. WRITE FLOATING-POINT STATUS VECTOR is supervisor-privileged. The instructions for referring to only a part of the FSV are: READ FLOATING-POINT CONTROL, WRITE FLOATING-POINT CONTROL, SET PRECISION MODE, SET UNDERFLOW MASK, SET OVERFLOW MASK, and SET SIGNIFICANCE MASK. These six instructions refer only to the current FSV.

Precision Modes

The floating-point arithmetic, load, and store instructions operate on data in both the short (4-byte) and long (8-byte) formats. Operations on short floating-point numbers are accomplished in short-precision mode; operations on long numbers are performed in long-precision mode. The precision mode is designated by bit 11 (P) of the FSV.

The SET PRECISION MODE instruction is provided for setting the precision-mode bit. When the bit is set to 0, subsequent operations are performed in short-precision mode; when the bit is set to 1, subsequent operations are performed in long-precision mode.

Exception Masks

Instructions are provided for selectively allowing or preventing program-exception interruptions due to exponent-underflow, exponent-overflow, and significance exceptions. The instructions are: SET OVERFLOW MASK, SET UNDERFLOW MASK, and SET SIGNIFICANCE MASK. These instructions provide for setting individual bits, called masks, in the FSV to 0 or 1. The significance, exponent-overflow, and exponent-underflow masks occupy bit positions 13, 14, and 15, respectively, in the FSV.

When a mask bit is set to 0, the associated floating-point exception results in a program-exception interruption, and the cause of the exception is identified with the exception-indicator bits in the FSV. When the mask bit is set to 1, the interruption does not occur, and the exception-indicator bits remain unchanged. The exponent-underflow and significance mask bits also determine the manner in which floating-point calculations are completed when the corresponding exception occurs.

Program Exceptions

The cause of a program-exception interruption for certain floating-point exceptions is indicated only in the PSV. These exceptions are: operation (when the floating-point feature is not installed), specification, access, and address. With the exception of the floating-point LOAD and STORE instructions, these program exceptions result in the operation's being suppressed. For the LOAD and STORE instructions, an operation exception results in the operation's being suppressed, while specification, access, and address exceptions result in the operation's being suspended. The details of the "Program Exception Conditions" are described in Chapter 3. For all other floating-point exceptions, the cause is indicated with both the PSV and the FSV. The PSV indicates the occurrence of a floating-point exception, and the FSV defines the specific cause.

Floating-point program exceptions are indicated with exception-indicator bits in the FSV (bits 17-23). When a floating-point program exception is detected resulting in a program-exception interruption, the appropriate exception-indicator

bit is set to 1 and the remaining indicator bits are set to 0's. The exception-indicator bits are defined in the following list:

Exception Indicator Bit	Program Exception
17	Floating-point-operation exception
18	Floating-point-privileged-operation exception
19	Floating-point-specification exception
20	Floating-point-divide exception
21	Significance exception
22	Exponent-overflow exception
23	Exponent-underflow exception

Each program exception is described in the following paragraphs. When an exception is detected, the current operation is either completed or suppressed. If the operation is suppressed, the contents of any result fields, including the condition indicators in the current PSV, are not changed.

Floating-Point-Operation Exception: This exception is detected when an undefined operation code is encountered in bit positions 6, 7, 10, and 11 of FF format instructions, and bit positions 6, 7, 30, and 31 of FS format instructions. For these exceptions, the operation is suppressed.

For the SET OVERFLOW MASK, SET PRECISION MODE, SET SIGNIFICANCE MASK, and SET UNDERFLOW MASK instructions, bit positions 4, 5, and 8 of the instruction represent an extension to the operation code. The four values of the undefined operation-code extensions are reserved and, if used, will produce unpredictable results.

Floating-Point-Privileged-Operation Exception: This exception is detected when WRITE FLOATING-POINT STATUS VECTOR is executed in either input/output or application mode.

For this exception, the operation is suppressed.

Floating-Point-Specification Exception: This exception is detected during execution of READ FLOATING-POINT STATUS VECTOR and WRITE FLOATING-POINT STATUS VECTOR when bit position 4 of the first-operand word does not contain 0. A floating-point-specification exception is also detected during WRITE FLOATING-POINT STATUS VECTOR when bits 0-4 of the FSV (bit positions 8-12 of the first-operand word) are not all 0's.

For this exception, the operation is suppressed.

Floating-Point-Divide Exception: This exception is detected during floating-point division when the divisor fraction is 0. For this exception, the operation is suppressed.

Significance Exception: This exception is detected when the result fraction in floating-point addition or subtraction is 0. The interruption can be disallowed by the significance mask (bit 13) in the FSV.

For this exception, the operation is completed. The significance mask also affects the result of the operation. When the mask is 0, the operation is completed

without further change to the characteristic and sign of the result. When the mask is 1, the operation is completed by replacing the result with a true zero, and the significance-exception indicator (bit 21) in the FSV is not made a 1.

Exponent-Overflow Exception: This exception is detected when the result characteristic in floating-point addition, subtraction, multiplication, or division exceeds 127 and the result fraction is not 0. The interruption can be disallowed by the exponent-overflow mask (bit 14) in the FSV.

For this exception, the operation is completed. The fraction is normalized, and the sign and result remain correct. The result characteristic is made 128 smaller than the correct characteristic. When the mask is 1, the exponent-overflow-exception indicator (bit 22) in the FSV is not made a 1.

Exponent-Underflow Exception: This exception is detected when the result characteristic in floating-point addition, subtraction, multiplication, or division is less than 0 and the result fraction is not 0. The interruption can be disallowed by the exponent-underflow mask (bit 15) in the FSV.

For this exception, the operation is completed. The setting of the exponent-underflow mask also affects the result of the operation. When the mask bit is 0, the fraction is normalized, the characteristic is made 128 larger than the correct characteristic, and the sign and fraction remain correct. When the mask bit is 1, the result is made a true zero, and the exponent-underflow-exception indicator (bit 23) in the FSV is not made a 1.

Instructions

The floating-point instructions and their mnemonics, formats, and operation codes follow. The procedure for describing the individual instructions, and the symbols used in the instruction formats and the expressions of the operations, are defined under "Instruction Descriptions" in Chapter 4.

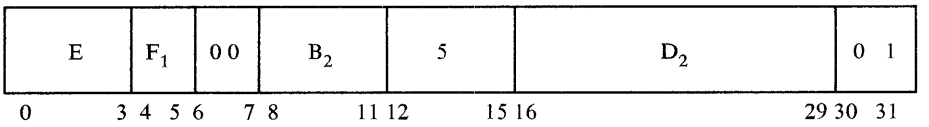
Note: *An assembler-language statement containing the mnemonic and the symbolic operand specifications is shown with each instruction. For a register-and-storage operation using LOAD, as an example, "LF" is the mnemonic and "f,dw14s(ra)" are the operand specifications. In the FS instruction format for LOAD, the F_1 field is derived from the first operand specification "f"; the D_2 and B_2 fields, designating the second operand, are derived from "dw14s(ra)". Refer to Appendix B for an explanation of the assembler language notation used in the instruction descriptions.*

Programming Note

Relative to the base address of the storage-operand location for the FS-format instructions, the range, in bytes, of the displacement (D) is $-32768 \leq D \times 4 \leq 32764$. Note that the displacement can be specified only in terms of an integral multiple of 4 bytes.

ADD NORMALIZED

AF f,dw14s(ra)



FS Format

Operation

(F_1) <— $(F_1) + MS[(B_2) + D_2 \times 4]$

Description

See the ADD NORMALIZED (register) instruction.

Result Conditions

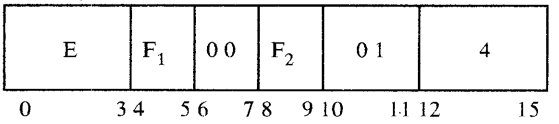
8 Result fraction is 0.
4 Result is less than 0.
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression/Completion)

Specification (operand 2: real address)
Access (operand 2: block invalid)
Operation (if the floating-point feature is not installed)
Separation (operand 2)
Address (operand 2: all)
Exponent Overflow
Exponent Underflow
Significance

ADD NORMALIZED (register)

AFR f,f



FF Format

Operation

(F_1) <— $(F_1) + (F_2)$

Description

The second operand is added to the first operand, and the normalized sum is placed in the first-operand location.

Addition of two floating-point numbers consists of characteristic comparison and fraction addition. The characteristics of the two operands are compared, and the fraction accompanying the smaller characteristic is shifted right, with its characteristic increased by 1 for each hexadecimal digit of shift until the two characteristics agree. When an operand is shifted right during alignment, the leftmost hexadecimal digit of the field shifted out is retained as a guard digit. The operand that is not shifted is considered to be extended to the right with a 0 guard digit. Both operands are considered to be extended to the right with a 0 when no alignment shift occurs. The fractions are then added algebraically to form an intermediate sum.

The short intermediate-sum fraction consists of 7 hexadecimal digits and a possible carry. The long intermediate-sum fraction consists of 15 hexadecimal digits and a possible carry. If a carry is present, the sum is shifted right one digit position, and the characteristic is increased by 1.

After the addition, the intermediate sum including the guard digit is shifted left as necessary to form a normalized number, provided the fraction is not 0. Vacated low-order digit positions are filled with 0's, and the characteristic is reduced by the number of hexadecimal digits of shift. The intermediate-sum fraction is subsequently truncated to the proper result-fraction length.

The sign of the sum is determined by the rules of algebra, unless all digits of the intermediate-sum fraction are 0, in which case the sign is made plus.

An exponent-overflow exception is detected when a carry from the high-order position of the intermediate-sum fraction causes the characteristic of the normalized sum to exceed 127. The operation is completed by making the characteristic 128 less than the correct value. The result is normalized and the sign and fraction remain correct. A program-exception interruption for exponent overflow occurs if the exponent-overflow mask bit is 0. When exponent overflow occurs and the exponent-overflow mask bit is 1, the interruption does not take place.

An exponent-underflow exception is detected when the characteristic of the normalized sum is less than 0 and the fraction is not 0. If the exponent-underflow mask bit is 0, the operation is completed by making the characteristic 128 larger than the correct value. The result is normalized, and the sign and fraction remain correct. A program-exception interruption for exponent underflow then takes place. When exponent underflow occurs and the exponent-underflow mask bit is 1, the interruption does not take place; instead, the operation is completed by making the result a true zero.

A significance exception is detected when the intermediate-sum fraction, including the guard digit, is 0. If the significance mask bit is 0, the intermediate-sum characteristic remains unchanged and becomes the characteristic of the result. No normalization occurs, and a program-exception interruption for significance takes place. If the significance mask bit is 1, the interruption does not occur; instead, the result is made a true zero.

Result Conditions

8 Result fraction is 0.
4 Result is less than 0.
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression/Completion)

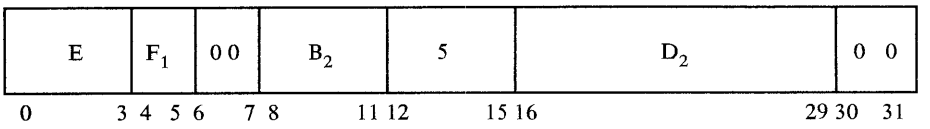
Operation (if the floating-point feature is not installed)
Exponent Overflow
Exponent Underflow
Significance

Programming Note

Interchanging the two operands in a floating-point addition does not affect the value of the sum.

ADD UNNORMALIZED

AU f,dw14s(ra)



FS Format

Operation

(F_1) <— $(F_1) + MS[(B_2) + D_2 \times 4]$

Description

See the ADD UNNORMALIZED (register) instruction.

Result Conditions

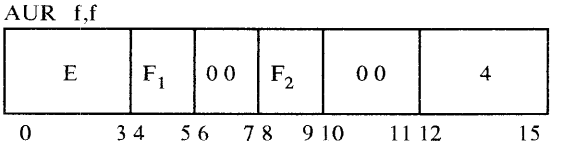
8 Result fraction is 0.
4 Result is less than 0.
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression/Completion)

Specification (operand 2: real address)
Access (operand 2: block invalid)
Operation (if the floating-point feature is not installed)
Separation (operand 2)
Address (operand 2: all)
Exponent Overflow
Significance

ADD UNNORMALIZED (register)

AUR f,f



FF Format

Operation

(F_1) <— $(F_1) + (F_2)$

Description

The second operand is added to the first operand, and the unnormalized sum is placed in the first-operand location.

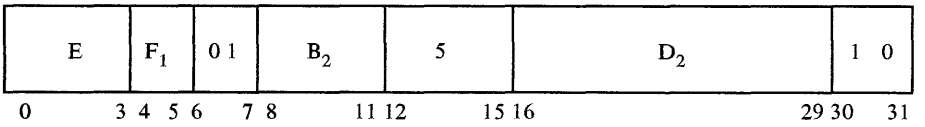
Execution of ADD UNNORMALIZED is similar to execution of ADD NORMALIZED, except that, after the addition, the intermediate-sum fraction is truncated to the proper result-fraction length without performing normalization. Leading 0's are not eliminated in the result fraction, exponent underflow cannot occur, and the guard digit does not participate in the detection of a significance exception. A significance exception is detected when the intermediate-sum fraction, not including the guard digit, is 0.

Result Conditions
8 Result fraction is 0.
4 Result is less than 0.
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression/Completion)
Operation (if the floating-point feature is not installed)
Exponent Overflow
Significance

COMPARE

CF f,dw14s(ra)



Operation
Result-Conditions <== (F_1) - MS[(B_2) + D_2 x 4]

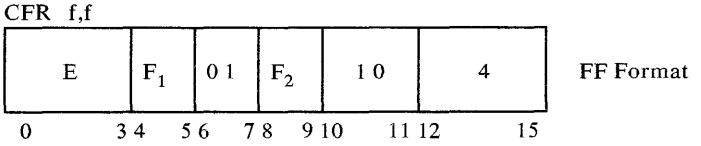
Description
See the COMPARE (register) instruction.

Result Conditions
8 Operands are equal
4 First operand is low
2 First operand is high
1 --
0 --

Program Exceptions (Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Operation (if the floating-point feature is not installed)
Separation (operand 2)
Address (operand 2: all)

COMPARE (register)

CFR f,f



Operation
Result-Conditions <== (F_1) - (F_2)

Description
The first operand is compared with the second operand; the comparison determines the indicated result condition.

Comparison is algebraic, taking into account the sign, fraction, and exponent of each number. An equality is established by following the rules for normalized

floating-point subtraction. When the intermediate sum, including the guard digit, is 0, the operands are equal. An exponent inequality is not indecisive for magnitude determination since the fractions may have different numbers of leading 0's. Neither operand is changed as a result of the operation.

Numbers with 0 fractions compare as equal quantities even when they differ in sign or characteristic.

An exponent-overflow, exponent-underflow, or significance exception cannot occur.

Result Conditions

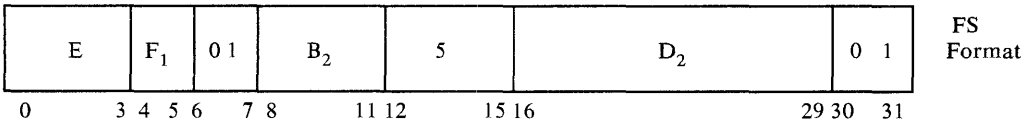
8 Operands are equal
4 First operand is low
2 First operand is high
1 --
0 --

Program Exceptions (Suppression)

Operation (if the floating-point feature is not installed)

DIVIDE

DF f,dw14s(ra)



Operation

$(F_1) \leftarrow (F_1) / MS[(B_2) + D_2 \times 4]$

Description

See the DIVIDE (register) instruction.

Result Conditions

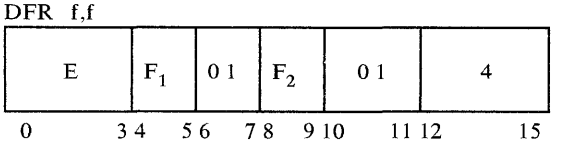
The conditions remain unchanged.

Program Exceptions (Suppression/Completion)

Specification (operand 2: real address)
Access (operand 2: block invalid)
Operation (if the floating-point feature is not installed)
Separation (operand 2)
Address (operand 2: all)
Exponent Overflow
Exponent Underflow
Floating-Point Divide

DIVIDE (register)

DFR f,f



FF Format

Operation

$(F_1) \leftarrow (F_1) / (F_2)$

Description

The first operand (the dividend) is divided by the second operand (the divisor) and replaced by the normalized quotient. No remainder is preserved. Floating-point division consists of characteristic subtraction and fraction division. The operands are prenormalized, and the difference between the dividend and divisor characteristics of the normalized operands, plus 64, is used as the characteristic of the intermediate quotient.

All dividend and divisor fraction digits participate in forming the fraction of the quotient. Postnormalizing the intermediate quotient is never necessary, but a right-shift of one digit position may be called for. The intermediate-quotient characteristic is adjusted for the shift. The intermediate-quotient fraction is subsequently truncated to the proper result-fraction length.

The sign of the quotient is determined by the rules of algebra, unless the quotient is made a true zero, in which case the sign is made plus.

An exponent-overflow exception is detected when the final-quotient characteristic exceeds 127 and the fraction is not 0. The result is normalized, the sign and fraction remain correct, and the characteristic is 128 less than the correct value. A program-exception interruption for exponent overflow occurs if the exponent-overflow mask bit is 0. When exponent overflow occurs and the exponent-overflow mask bit is 1, the interruption does not take place.

An exponent-underflow exception is detected when the characteristic of the normalized quotient is less than 0 and the fraction is not 0. If the exponent-underflow mask bit is 0, a program-exception interruption occurs. The result is normalized, its sign and fraction remain correct, and the characteristic is made 128 larger than the correct value. If the exponent-underflow mask bit is 1, the interruption does not take place; instead, the operation is completed by making the quotient a true zero.

Exponent underflow is not indicated when an operand characteristic becomes less than 0 during prenormalization or the intermediate-quotient characteristic is less than 0, but the final quotient can be expressed without encountering exponent underflow.

A floating-point divide exception is detected when the divisor fraction is 0. The operation is suppressed, and a program-exception interruption for floating-point divide occurs.

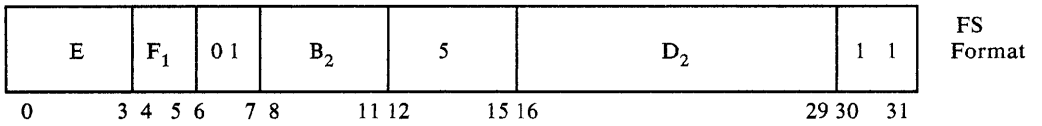
When the dividend fraction is 0, the quotient is made a true zero, and neither exponent overflow nor exponent underflow is indicated. A division of 0 by 0, however, causes the operation to be suppressed and a program-exception interruption for floating-point divide to occur.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression/Completion)
Operation (if the floating-point feature is not installed)
Exponent Overflow
Exponent Underflow
Floating-Point Divide

LOAD

LF f,dw14s(ra)



FS Format

Operation
$(F_1) \leftarrow MS [(B_2) + D_2 \times 4]$

Description
The second operand is placed unchanged in the first-operand location.

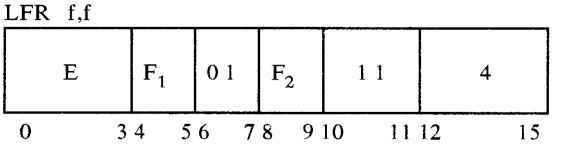
Specification, access, separation, and address program exceptions are detected at the time the inaccessible location is referred to, and execution is *suspended.*

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suspension/Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Operation (if the floating-point feature is not installed)
Separation (operand 2)
Address (operand 2: all)

LOAD (register)

LFR f,f



FF Format

Operation
$(F_1) \leftarrow (F_2)$

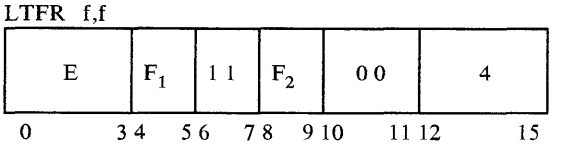
Description
The second operand is placed unchanged in the first-operand location.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (if the floating-point feature is not installed)

LOAD AND TEST (register)

LTFR f,f



FF Format

Operation

$(F_1) \leftarrow (F_2)$

Result-Conditions $<==$ (F_1)

Description

The second operand is placed unchanged in the first-operand location, and its sign and magnitude are tested to determine the indicated result condition.

Result Conditions

8 Result fraction is 0.
4 Result is less than 0.
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression)

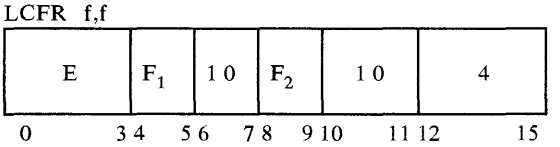
Operation (if the floating-point feature is not installed)

Programming Note

When the same register is specified for both the first-operand location and the second-operand location, the operation is equivalent to a test without data movement.

LOAD COMPLEMENT (register)

LCFR f,f



FF Format

Operation

$(F_1) \leftarrow (F_2)$

$(F_1<0>) \leftarrow \neg(F_1<0>)$

Description

The second operand is placed in the first-operand location with the sign changed to the opposite value.

The sign bit is inverted, even if the fraction is 0. The characteristic and fraction are not changed.

Result Conditions

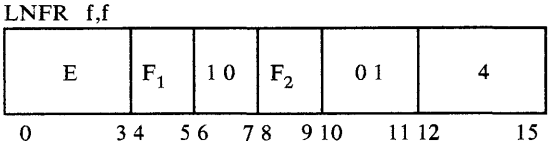
8 Result fraction is 0.
4 Result is less than 0.
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression)

Operation (if the floating-point feature is not installed)

LOAD NEGATIVE (register)

LNFR f,f



FF Format

Operation
$(F_1) \leftarrow (F_2)$
$(F_1\langle 0\rangle) \leftarrow 1$

Description
The second operand is placed in the first-operand location with the sign made minus.

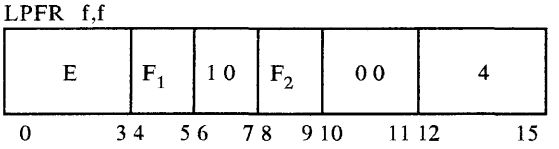
The sign bit is made 1, even if the fraction is 0. The characteristic and fraction are not changed.

Result Conditions
8 Result fraction is 0.
4 Result is less than 0.
2 --
1 --
0 --

Program Exceptions (Suppression)
Operation (if the floating-point feature is not installed)

LOAD POSITIVE (register)

LPFR f,f



FF Format

Operation
$(F_1) \leftarrow (F_2)$
$(F_1\langle 0\rangle) \leftarrow 0$

Description
The second operand is placed in the first-operand location with the sign made plus.

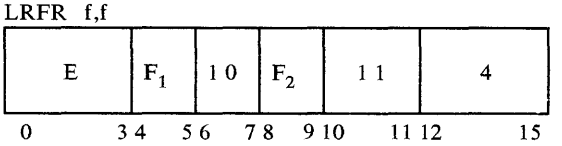
The sign bit is made 0. The characteristic and fraction are not changed.

Result Conditions
8 Result fraction is 0.
4 --
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression)
Operation (if the floating-point feature is not installed)

LOAD ROUNDED (register)

LRFR f,f



FF Format

Operation
(F_1) <— (F_2) rounded long to short

Description
The long second operand is rounded to the short format, and the result is placed in the first-operand location. The low-order 32 bits of the first-operand register remain unchanged. The second operand remains unchanged as a result of the operation.

Rounding consists of adding a 1 in bit position 32 of the long second operand, and propagating the carry, if any, 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 high-order digit position of the fraction, the fraction is shifted right one digit position, and the characteristic is increased by 1.

The sign of the result is the same as the sign of the second operand. No normalization takes place.

An exponent-overflow exception detected when shifting the fraction right causes the characteristic to exceed 127. The operation is completed by loading a number whose characteristic is 128 less than the correct value. The result is normalized, and the sign and fraction remain correct. A program-exception interruption for exponent overflow occurs if the exponent-overflow mask bit is 0. When exponent overflow occurs and the exponent-overflow mask bit is 1, the interruption does not take place.

Exponent-underflow and significance exceptions cannot occur.

Result Conditions
The conditions remain unchanged.

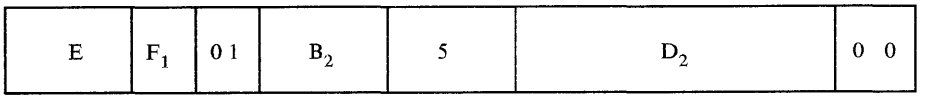
Program Exceptions (Suppression/Completion)
Operation (if the floating-point feature is not installed)
Exponent Overflow

Programming Notes
The operation of LOAD ROUNDED is the same in both short-precision and long-precision modes.

The result is rounded away from 0. That is, when rounding takes place (a carry occurs out of bit position 32 of the long second operand), the short-format result is increased in magnitude.

MULTIPLY

MF f,dw14s(ra)



FS Format

Operation

$(F_1) \leftarrow (F_1) \times MS[(B_2) + D_2 \times 4]$

Description

See the MULTIPLY (register) instruction.

Result Conditions

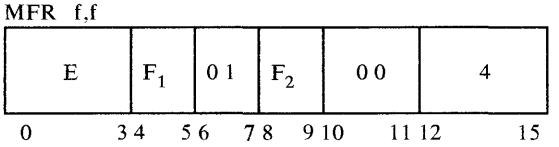
The conditions remain unchanged.

Program Exceptions (Suppression/Completion)

Specification (operand 2: real address)
Access (operand 2: block invalid)
Operation (if the floating-point feature is not installed)
Separation (operand 2)
Address (operand 2: all)
Exponent Overflow
Exponent Underflow

MULTIPLY (register)

MFR f,f



FF Format

Operation

$(F_1) \leftarrow (F_1) \times (F_2)$

Description

The normalized product of the second operand (the multiplier) and the first operand (the multiplicand) is placed in the first-operand location.

Multiplication of two floating-point numbers consists of exponent addition and fraction multiplication. The operands are prenormalized, and the sum of the characteristics of the normalized operands, less 64, is used as the characteristic of the intermediate product.

The product of the fractions is developed such that the result has the exact fraction product truncated to the proper result-fraction length. When the result is normalized without requiring any postnormalization, the intermediate-product fraction is truncated to the result-fraction length, and the intermediate-product characteristic becomes the final product characteristic. When the intermediate-product fraction has one leading 0 digit, it is shifted left one digit position, bringing the contents of the guard-digit position into the low-order position of the result fraction. The intermediate-product characteristic is reduced by 1 to account for the left shift. The intermediate-product fraction is subsequently truncated to the result-fraction length.

In short-precision mode, the multiplier and multiplicand have 6-digit fractions, and the product fraction has the full 14 digits of the long format, with the 2 low-order fraction digits always 0. In long-precision mode, the multiplier and multiplicand fractions have 14 digits and the result product fraction is truncated to 14 digits.

The sign of the product is determined by the rules of algebra, unless all digits of the product fraction are 0, in which case the sign is made plus.

An exponent-overflow exception is detected when the characteristic of the normalized product exceeds 127 and the fraction of the product is not 0. 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-exception interruption for exponent overflow occurs if the exponent-overflow mask bit is 0. When exponent overflow occurs and the exponent-overflow mask bit is 1, the interruption does not take place.

Exponent overflow is not indicated if the intermediate-product characteristic exceeds 127 but is brought within range by normalization.

An exponent-underflow exception is detected when the characteristic of the normalized product is less than 0 and the fraction of the product is not 0. If the exponent-underflow mask bit is 0, the operation is completed by making the characteristic 128 larger than the correct value, and a program-exception interruption for exponent underflow occurs. The result is normalized, and the sign and fraction remain correct. If the exponent-underflow mask bit is 1, the interruption does not take place; instead, the operation is completed by making the product a true zero.

Exponent underflow is not indicated when the characteristic of an operand becomes less than 0 during prenormalization, but the characteristic of the normalized product is within range.

When either or both operand fractions are 0, the result is made a true zero, and neither exponent overflow nor exponent underflow is indicated.

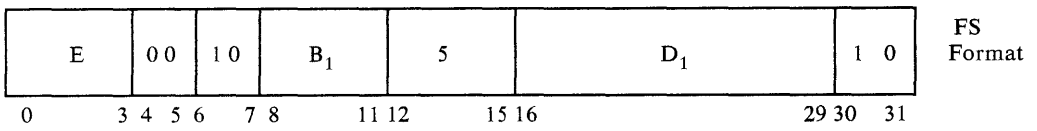
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression/Completion)
Operation (if the floating-point feature is not installed)
Exponent Overflow
Exponent Underflow

Programming Note
Interchanging the two operands in a floating-point multiplication does not affect the value of the product.

READ FLOATING-POINT CONTROL

RFC dw14s(ra)



Operation

$MS[(B_1) + D_1 \times 4 + 2] \leftarrow$ Current-FSV<8..23>

Description

The contents of bit positions 8-23 of the current FSV are placed in the low-order half of the word location designated by the first-operand address. The high-order 16 bits of the word at the first-operand location remain unchanged.

Bit positions 8-23 of the FSV contain the precision-mode bit, exception-mask bits, the floating-point-check bit, and exception-indicator bits.

Bit positions 4 and 5 of the instruction are reserved and should contain 0's.

Result Conditions

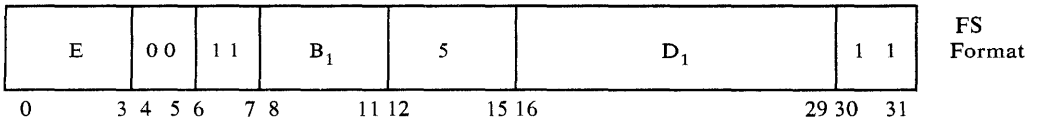
The conditions remain unchanged.

Program Exceptions (Suppression)

Specification (operand 1: real address)
Access (operand 1: block invalid, store protection)
Operation (if the floating-point feature is not installed)
Separation (operand 1)
Address (operand 1: all)

READ FLOATING-POINT STATUS VECTOR

RFS dw14s(ra)



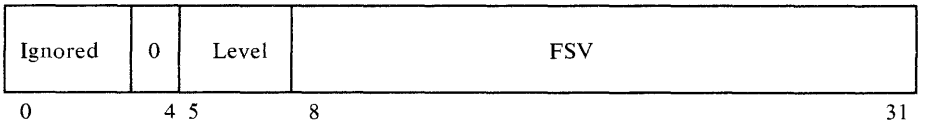
Operation

$MS[(B_1) + D_1 \times 4 + 1] \leftarrow FSV[MS[(B_1) + D_1 \times 4]<5..7>]$

Description

The contents of the FSV associated with the specified priority level are placed right-justified in the word storage location designated by the first-operand address.

The following illustrates the format of the first-operand word:



The priority level associated with the FSV to be read is specified by the contents of bit positions 5-7 of the first operand.

Bit positions 0-3 of the first operand are ignored. Bit position 4 is reserved and must contain a 0; otherwise, a floating-point specification exception is indicated. The contents of bit positions 0-7 of the word in storage remain unchanged.

Bit positions 4 and 5 of the instruction are reserved and should contain 0's.

Result Conditions
The conditions remain unchanged.

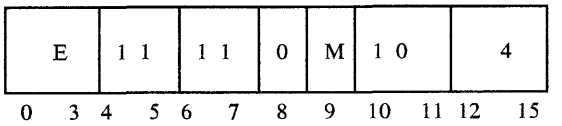
Program Exceptions (Suppression)
Specification (operand 1: real address)
Access (operand 1: block invalid, store protection)
Operation (if the floating-point feature is not installed)
Separation (operand 1)
Address (operand 1: all)
Floating-point specification (operand 1: bit 4 of operand is 1)

Programming Notes
This instruction is intended for use by a supervisory program.

When the current priority level is specified in the first operand, the current FSV is read.

SET OVERFLOW MASK

SFOM m1



FF Format

Operation
Current-FSV<14> <— M

Description
The exponent-overflow mask bit in the current FSV is replaced by the content of the M field of the instruction.

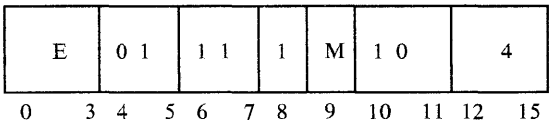
The M field can contain either a 0 or a 1. When the M field contains a 0, subsequent exponent-overflow exceptions result in a program-exception interruption; when the M field contains a 1, the interruption for exponent-overflow does not occur.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (if the floating-point feature is not installed)

SET PRECISION MODE

SFPM m1



FF Format

Operation
Current-FSV<11> <— M

Description
The precision-mode bit in the current FSV is replaced by the content of the M field of the instruction.

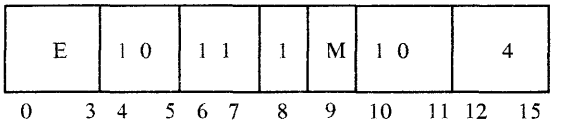
The M field can contain either a 0 or a 1. When the M field contains a 0, subsequent floating-point arithmetic, load and store operations are performed in short-precision mode; when the M field contains a 1, long-precision mode is used.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (if the floating-point feature is not installed)

SET SIGNIFICANCE MASK

SFSM m1



FF Format

Operation
Current-FSV<13> <— M

Description
The significance mask bit in the current FSV is replaced by the content of the M field of the instruction.

The M field can contain either a 0 or a 1. When the M field contains a 0, subsequent significance exceptions result in a program-exception interruption; when the M field contains a 1, the interruption for significance does not occur.

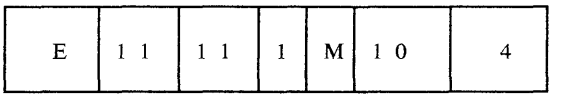
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (if the floating-point feature is not installed)

Programming Note
The significance mask bit also determines the manner in which floating-point addition and subtraction are completed when a significance exception occurs. Refer to the individual instruction descriptions for details.

SET UNDERFLOW MASK

SFUM m1



FF Format

Operation
Current-FSV<15> <— M

Description
The exponent-underflow mask bit in the current FSV is replaced by the content of the M field of the instruction.

The M field can contain either a 0 or a 1. When the M field contains a 0, subsequent exponent-underflow exceptions result in a program-exception interruption; when the M field contains a 1, the interruption for exponent-underflow does not occur.

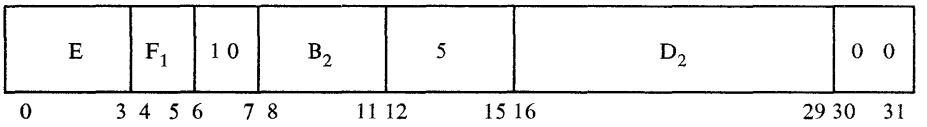
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (if the floating-point feature is not installed)

Programming Note
The exponent-underflow mask bit also determines the manner in which floating-point normalized addition, normalized subtraction, multiplication, and division are completed when an exponent-underflow exception occurs. Refer to the individual instruction descriptions for details.

STORE

STF f,dw14s(ra)



FS Format

Operation
MS[(B_2) + D_2 x 4] <— (F_1)

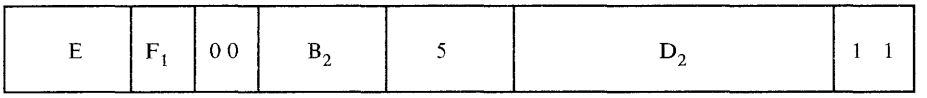
Description
The first operand is placed unchanged at the second-operand location.
Specification, access, separation, and address program exceptions are detected at the time the inaccessible location is referred to and execution is *suspended.*

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suspension/Suppression)
Specification (operand 2: real address)
Access (operand 2: block invalid, store protection)
Operation (if the floating-point feature is not installed)
Separation (operand 2)
Address (operand 2: all)

SUBTRACT NORMALIZED

SF f,dw14s(ra)



FS Format

Operation

$(F_1) \leftarrow (F_1) - MS[(B_2) + D_2 \times 4]$

Description

See the SUBTRACT NORMALIZED (register) instruction.

Result Conditions

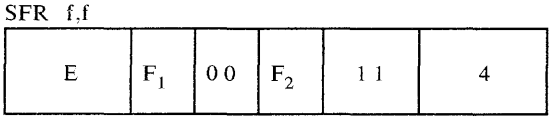
8 Result fraction is 0.
4 Result is less than 0.
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression/Completion)

Specification (operand 2: real address)
Access (operand 2: block invalid)
Operation (if the floating-point feature is not installed)
Separation (operand 2)
Address (operand 2: all)
Exponent Overflow
Exponent Underflow
Significance

SUBTRACT NORMALIZED (register)

SFR f,f



FF Format

Operation

$(F_1) \leftarrow (F_1) - (F_2)$

Description

The second operand is subtracted from the first operand, and the normalized difference is placed in the first-operand location.

Execution of SUBTRACT NORMALIZED is similar to execution of ADD NORMALIZED except that the second operand participates in the operation with its sign bit inverted.

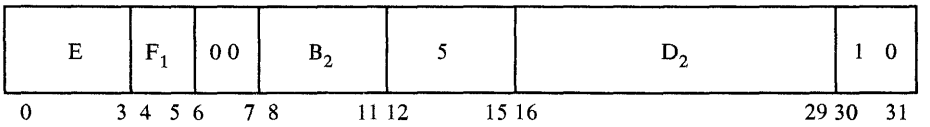
Result Conditions

8 Result fraction is 0.
4 Result is less than 0.
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression/Completion)
Operation (if the floating-point feature is not installed)
Exponent Overflow
Exponent Underflow
Significance

SUBTRACT UNNORMALIZED

SU f,dw14s(ra)



FS Format

Operation
$(F_1) \leftarrow (F_1) - MS[(B_2) + D_2 \times 4]$

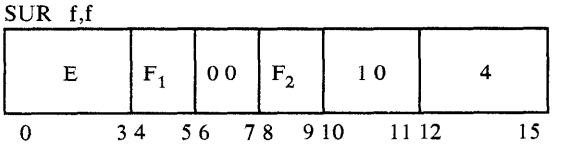
Description
See the SUBTRACT UNNORMALIZED (register) instruction.

Result Conditions
8 Result fraction is 0.
4 Result is less than 0.
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression/Completion)
Specification (operand 2: real address)
Access (operand 2: block invalid)
Operation (if the floating-point feature is not installed)
Separation (operand 2)
Address (operand 2: all)
Exponent Overflow
Significance

SUBTRACT UNNORMALIZED (register)

SUR f,f



FF Format

Operation
$(F_1) \leftarrow (F_1) - (F_2)$

Description
The second operand is subtracted from the first operand, and the unnormalized difference is placed in the first-operand location.

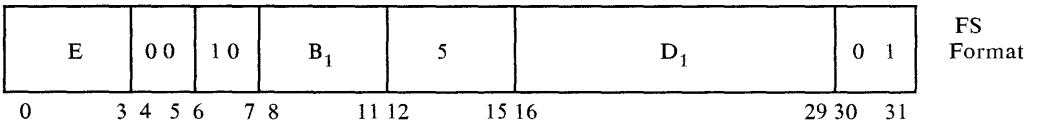
Execution of SUBTRACT UNNORMALIZED is similar to that of ADD UNNORMALIZED except that the second operand participates in the operation with its sign bit inverted.

Result Conditions
8 Result fraction is 0.
4 Result is less than 0.
2 Result is greater than 0.
1 --
0 --

Program Exceptions (Suppression/Completion)
Operation (if the floating-point feature is not installed)
Exponent Overflow
Significance

WRITE FLOATING-POINT CONTROL

WFC dw14s(ra)



Operation
Current-FSV<8..23> <— MS[(B_1) + D_1 x 4 + 2]

Description
The contents of bit positions 8-23 of the current FSV are replaced with the contents of the low-order half of the word designated by the first-operand address. The high-order 16 bits of the word at the first-operand location are ignored.

Bit positions 8-23 of the FSV contain the precision-mode bit, exception-mask bits, floating-point-check bit, and exception-indicator bits.

Bit positions 4 and 5 of the instruction are reserved and should contain 0's.

Bits 8-10 and 12 of the FSV (bit positions 16-18 and 20 of the first-operand word) are reserved and should be written as 0's.

Result Conditions
The conditions remain unchanged.

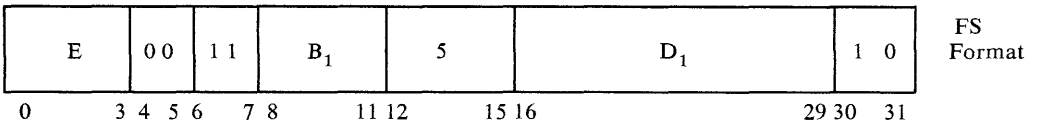
Program Exceptions (Suppression)
Specification (operand 1: real address)
Access (operand 1: block invalid)
Operation (if the floating-point feature is not installed)
Separation (operand 1)
Address (operand 1: all)

Programming Notes
WRITE FLOATING-POINT CONTROL can be used to alter the state of the precision-mode and exception-mask bits. The instruction can also be used to clear the floating-point-check and exception-indicator bits.

An interruption will not occur as a result of making the floating-point-check bit, or any of the exception-indicator bits, a 1 with this instruction.

WRITE FLOATING-POINT STATUS VECTOR

WFS dw14s(ra)



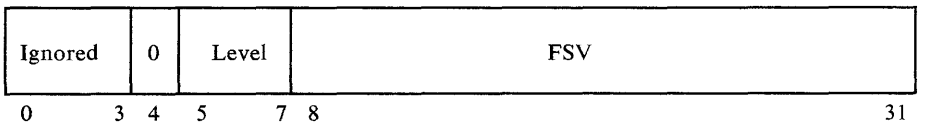
Operation

$FSV[MS[(B_1) + D_1 \times 4]<5..7>] <— MS[(B_1) + D_1 \times 4 + 1]$

Description

The contents of the FSV associated with the specified priority level are replaced with the contents of the rightmost 24 bit positions of the word designated by the first-operand address.

The following illustrates the format of the first-operand word:



The priority level associated with the FSV to be written is specified by the contents of bit positions 5-7 of the first operand.

Bit positions 0-3 of the first operand are ignored. Bit position 4 of the first operand, and bits 0-4 of the FSV (bit positions 8-12 of the operand), are reserved and must contain 0's; otherwise, a floating-point specification exception is indicated. The word in storage remains unchanged.

Bits 8-10 and 12 of the FSV (bit positions 16-18 and 20 of the first-operand word) are reserved and should be written as 0's.

Bit positions 4 and 5 of the instruction are reserved and should contain 0's.

WRITE FLOATING-POINT STATUS VECTOR is supervisor-privileged. An attempt to execute this instruction in input/output or application mode causes a program-exception interruption due to floating-point privileged-operation exception.

Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)

Specification (operand 1: real address)
Access (operand 1: block invalid)
Operation (if the floating-point feature is not installed)
Separation (operand 1)
Address (operand 1: all)
Floating-point privileged operation
Floating-point specification (operand 1: bits 4 and 8-12 of operand are not all 0's)

Programming Notes

WRITE FLOATING-POINT STATUS VECTOR can be used to assign a floating-point register set for use on the specified priority level, or to alter the state of the precision-mode and exception-mask bits. The instruction can also be used to clear the floating-point-check bit and exception-indicator bits.

An interruption will not occur as a result of making the floating-point-check bit, or any of the exception-indicator bits, a 1 with this instruction.

When the current priority level is specified in the first operand, the current FSV is replaced with the new FSV from main storage.

PART III. SUPERVISORY FACILITIES

Chapter 6. Register Organization

This chapter describes the logical organization of three groups of registers provided in each PCE of an 8100 system. The permanent assignment of certain registers to hold control information is discussed. The supervisor-privileged operations used to refer to information in any register in the principal and adjunct register groups are also described. These operations are called the *register-indirect* operations.

Organization

The PCE can address information held in various registers which are organized in groups and sets. Two groups of registers are always provided by the PCE: the *principal register group* and the *adjunct register group.* When the optional floating-point feature is installed, a third group of registers, the *floating-point register group,* is provided.

Registers are organized in *register sets* within each group. Each principal or adjunct register set consists of eight registers numbered consecutively 0-7. Each principal or adjunct register contains 32 bits. The floating-point register sets each consist of four registers numbered consecutively 0-3. Each floating-point register contains 64 bits. This organization is illustrated in Figure 6-1, which also illustrates selection of information in the principal and adjunct register groups with register-indirect operations (see "Access to Register Contents" in this chapter).

Principal Registers

The group of principal registers consists of 64 register sets numbered consecutively 0-63 (with eight registers in each set). Principal register sets 0, 1, 4, 5, and 8-15 are permanently assigned to hold information necessary for PCE control and channel input/output operations. Principal register sets 16-63 are available for assignment to programs as general registers. Sets 2, 3, 6, and 7 are reserved. Figure 6-2 shows the permanent assignments of principal register sets.

A program has two principal register sets assigned to it as general-register sets: a *primary* set and a *secondary* set. These two sets provide a program with 16 general registers of 32 bits each. Two fields in the program's PSV designate, respectively, the numbers of the primary and secondary register sets assigned to the program. Chapter 9, "PCE Control," describes the format and functions of the PSV.

Principal register sets 0, 1, 4, and 5 hold PSV information. Information in the PSV is switched whenever the current PSV is stored and a new PSV is introduced. PSV information is stored in, and introduced from, permanently assigned principal register locations.

Two PSVs, referred to as the *primary* PSV and the *secondary* PSV, are associated with each priority level. Accordingly, each priority level is assigned principal register locations that hold two PSVs. The primary PSVs are held in principal register sets 0 and 1. Sets 4 and 5 provide the register locations for secondary PSVs. Chapter 9, "PCE Control," describes priority levels and the dual PSV facility.

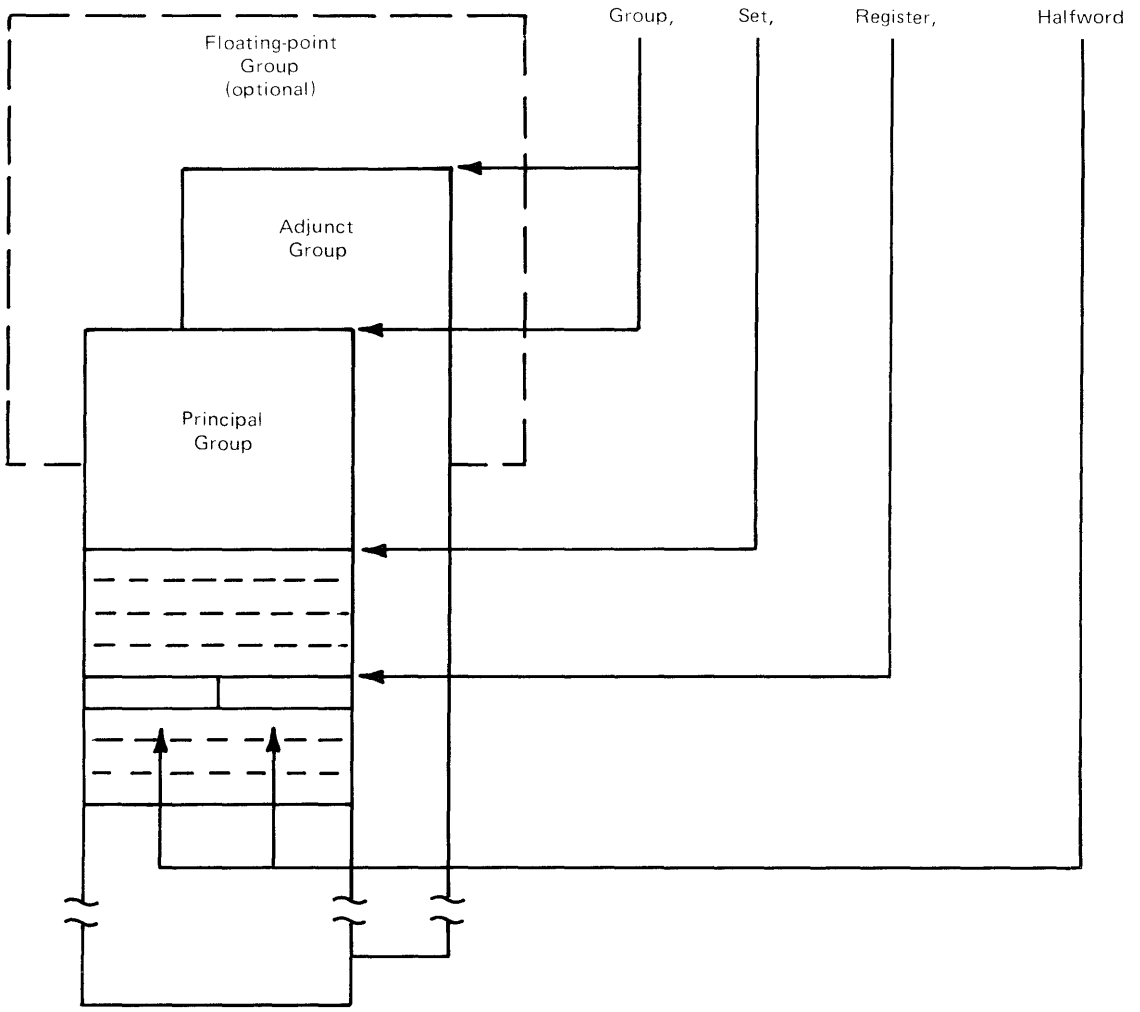


Figure 6-1. Register Organization and Information Selection

Each 64-bit PSV occupies a specific pair of registers in its assigned principal register set. Bit positions 0-31 of the PSV are held in the even-numbered register of the pair; bits 32-63 are held in the odd-numbered register of the pair. Figure 6-3 shows the assignment of register locations that hold the primary and secondary PSV for each priority level.

Principal register sets 8-15 are permanently assigned as channel pointers. Channel pointers are used during channel I/O operations to hold the addresses of main-storage locations referred to during the input/output operation. Each 32-bit channel pointer occupies one register. Channel pointers 0-31 are assigned to register sets 12-15; channel pointers 32-63 are assigned to register sets 8-11. Figure 6-4 shows the assignment of channel pointers in principal register sets.

Programming Notes:

- Any principal register set may be assigned as a primary or secondary general-register set.
- Because principal register sets 0, 1, 4, 5, and 8-15 are assigned as PSV locations and channel pointers, these register sets should not be assigned as general registers during normal system operation.

Principal register sets 2, 3, 6, and 7 are reserved. Reserved register sets should not be used by a program or channel.

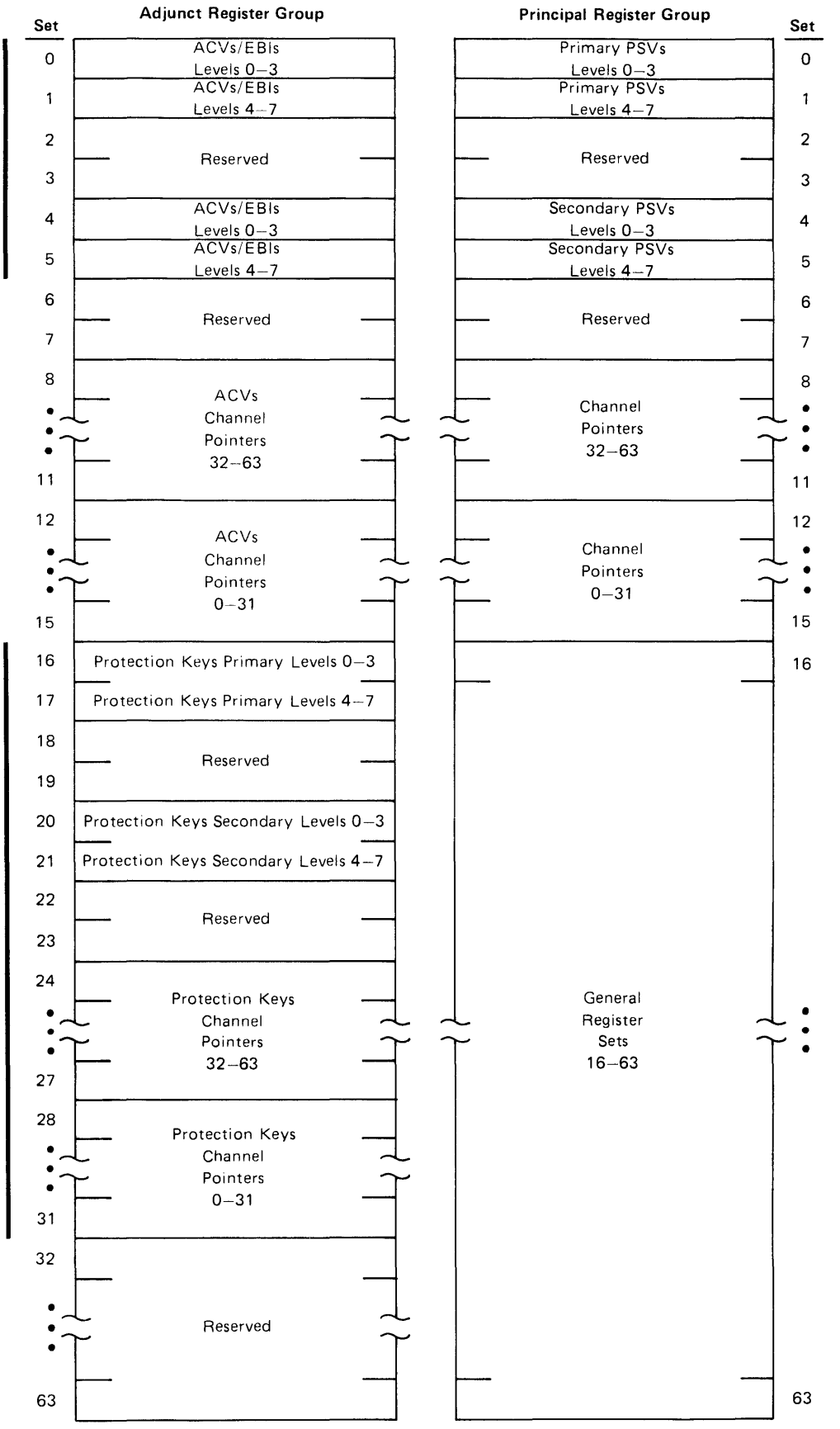


Figure 6-2. Assignment of Principal and Adjunct Register Sets

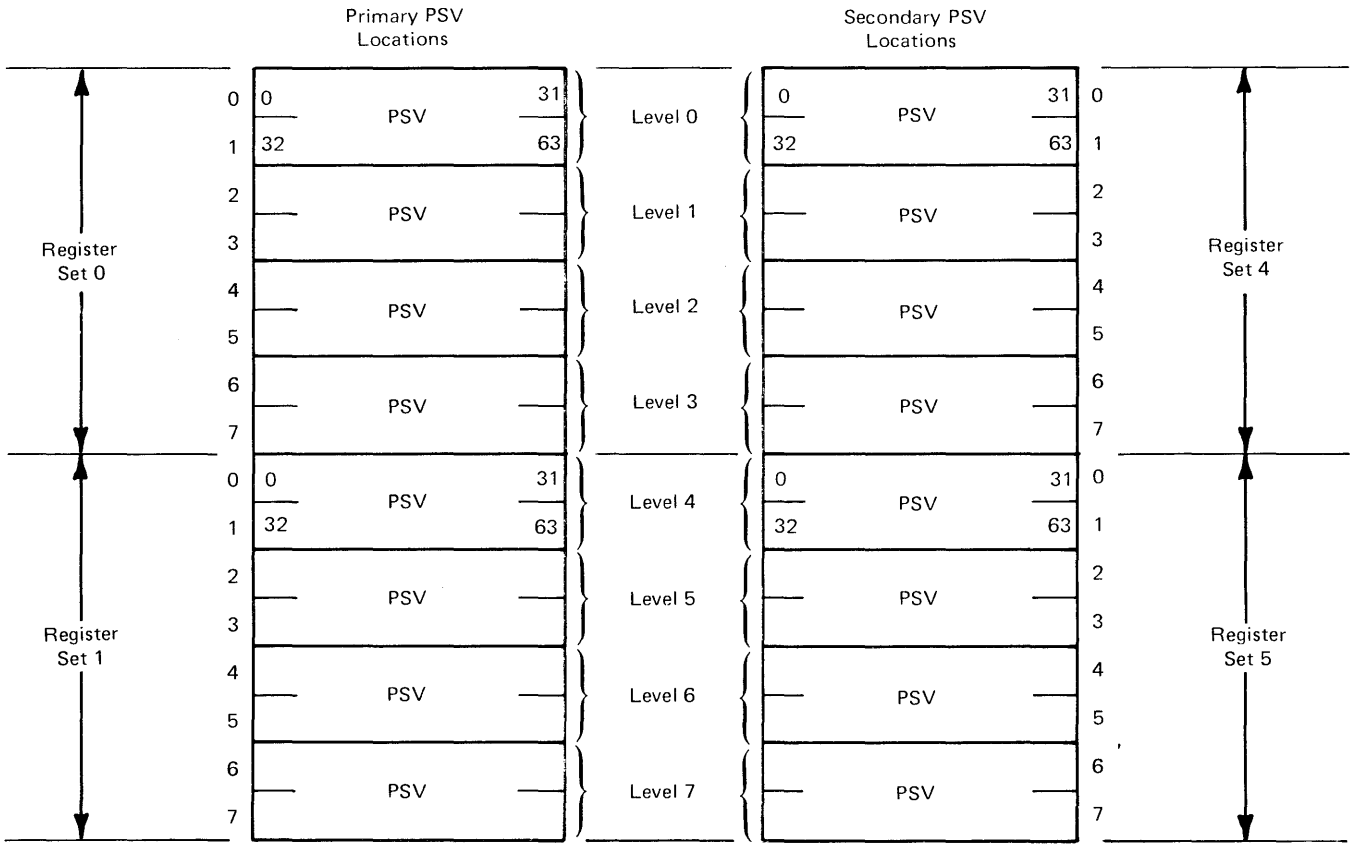


Figure 6-3. PSV Locations in Principal Register Sets

Adjunct Registers

The group of adjunct registers consists of 64 register sets numbered consecutively 0-63 (with eight registers in each set). Adjunct registers are provided to hold address control vector (ACV), exception block index (EBI), and protection key information. An ACV contains information required for dynamic address relocation and for activation of dynamic address translation. Adjunct register sets 0, 1, 4, and 5 contain ACVs and EBIs associated with PSVs; sets 8-15 contain ACVs associated with channel pointers; the even-numbered registers in sets 16, 17, 20, 21, and all registers in sets 24-31 contain protection keys. Adjunct register sets 2, 3, 6, 7, 18, 19, 22, 23, and 32-63, register 1 in adjunct register set 0, and the odd-numbered registers in sets 16, 17, 20, and 21 are reserved. Reserved register sets should not be used by a program.

One ACV is associated with each PSV and is introduced as the current ACV whenever the corresponding PSV is made active. Chapter 7, "Dynamic Address Relocation and Translation," describes the format and functions of the ACV. An ACV is also associated with each channel pointer, and is used to control addressing during channel I/O operations that use the corresponding channel pointer. Chapter 8, "Input/Output Operations," discusses the role of the ACV and channel pointer in channel I/O operations. The permanent assignments and pairings of principal and adjunct register sets are shown in Figure 6-2.

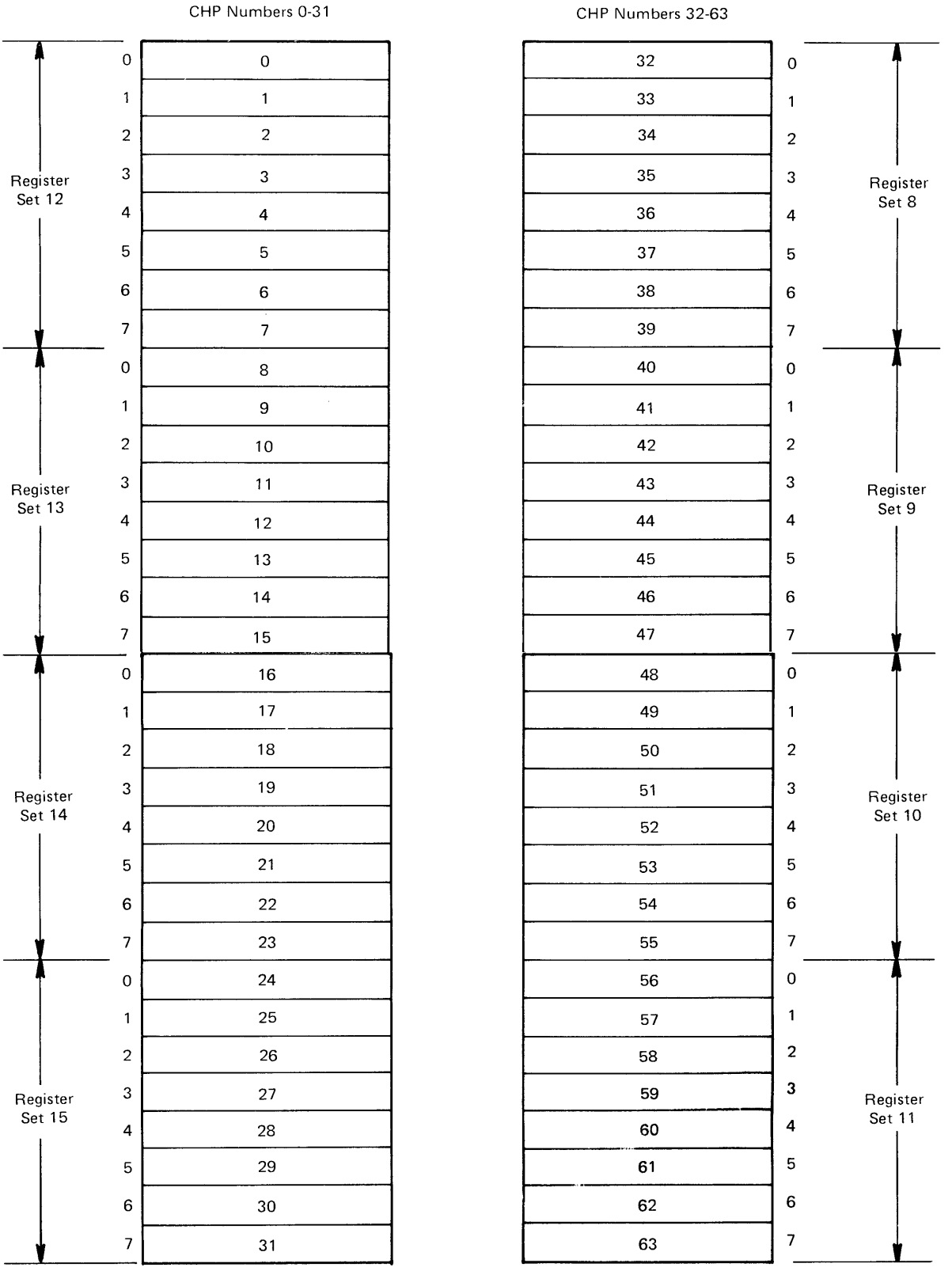


Figure 6-4. Channel Pointers

Adjunct register sets 0, 1, 4, and 5 are associated with principal register sets 0, 1, 4, and 5, respectively, so that one 32-bit ACV/EBI pair corresponds to each PSV. Even-numbered registers in these adjunct register sets are used for ACVs; odd-numbered registers, with the exception of register 1 in register set 0, are used for EBIs. There is no EBI register associated with the primary PSV/ACV for priority level 0. Figures 6-5 and 6-6 show the association of ACV register locations with PSV register locations.

Adjunct register sets 8-15 are associated with principal register sets 8-15 so that one ACV corresponds to each channel pointer. Figure 6-7 shows the ACV register location associated with each channel pointer.

Adjunct register sets 16, 17, 20, and 21 are associated with adjunct register sets 0, 1, 4, and 5, respectively, so that one protection key corresponds to each PSV/ACV. Even-numbered registers in adjunct register sets 16, 17, 20, and 21 are used for protection keys; odd-numbered registers in these sets are reserved. Figures 6-8 and 6-9 show how protection keys correspond to ACVs associated with PSVs.

Adjunct register sets 24-31 are associated with adjunct register sets 8-15, respectively, so that one protection key corresponds to each channel pointer/ACV. All registers in adjunct register sets 24-31 are used for protection keys. Figure 6-10 shows how protection keys correspond to ACVs associated with channel pointers.

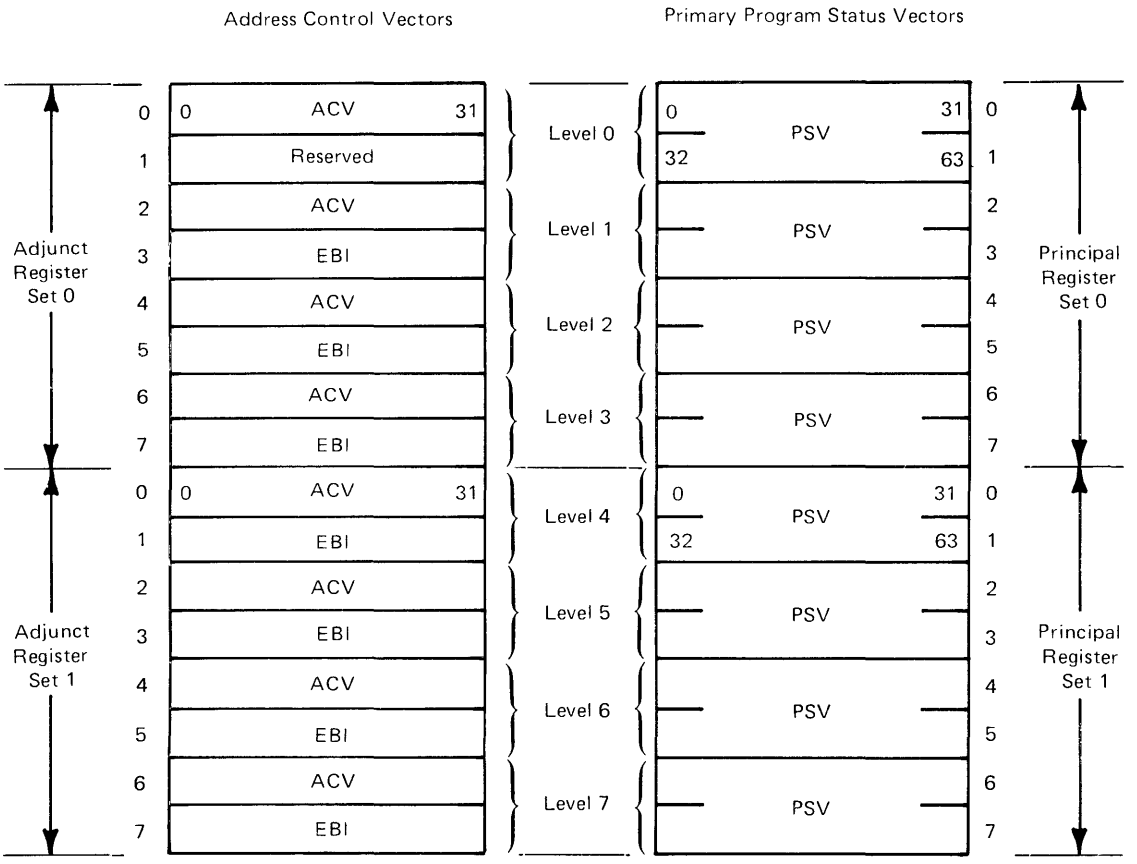


Figure 6-5. ACV/EBI Pairs Associated with Primary PSVs

Depending on processor model, certain byte locations in adjunct registers are not necessarily made available to the program when the locations are used only to hold ACV information represented by all 0 bits. In particular, the two high-order byte locations (bits 0-15) are not made available on any processor model having a PCE address space that is never larger than 16,777,216 bytes. These two byte locations are not made available because they always contain 0's when the adjunct register is used to hold an ACV that describes a logical address space of 16,777,216 or fewer bytes. When an ACV is referred to by the PCE or channel, unavailable byte locations in the adjunct register are assumed to contain 0's.

An attempt by the program to refer to unavailable adjunct-register byte locations (using the register-indirect instructions) results in a program-exception interruption.

Programming Note: Adjunct register sets 2, 3, 6, 7, 18, 19, 22, 23, and 32-63, register 1 in adjunct register set 0, and the odd-numbered registers in sets 16, 17, 20, and 21 are reserved. Reserved register sets should not be used by a program.

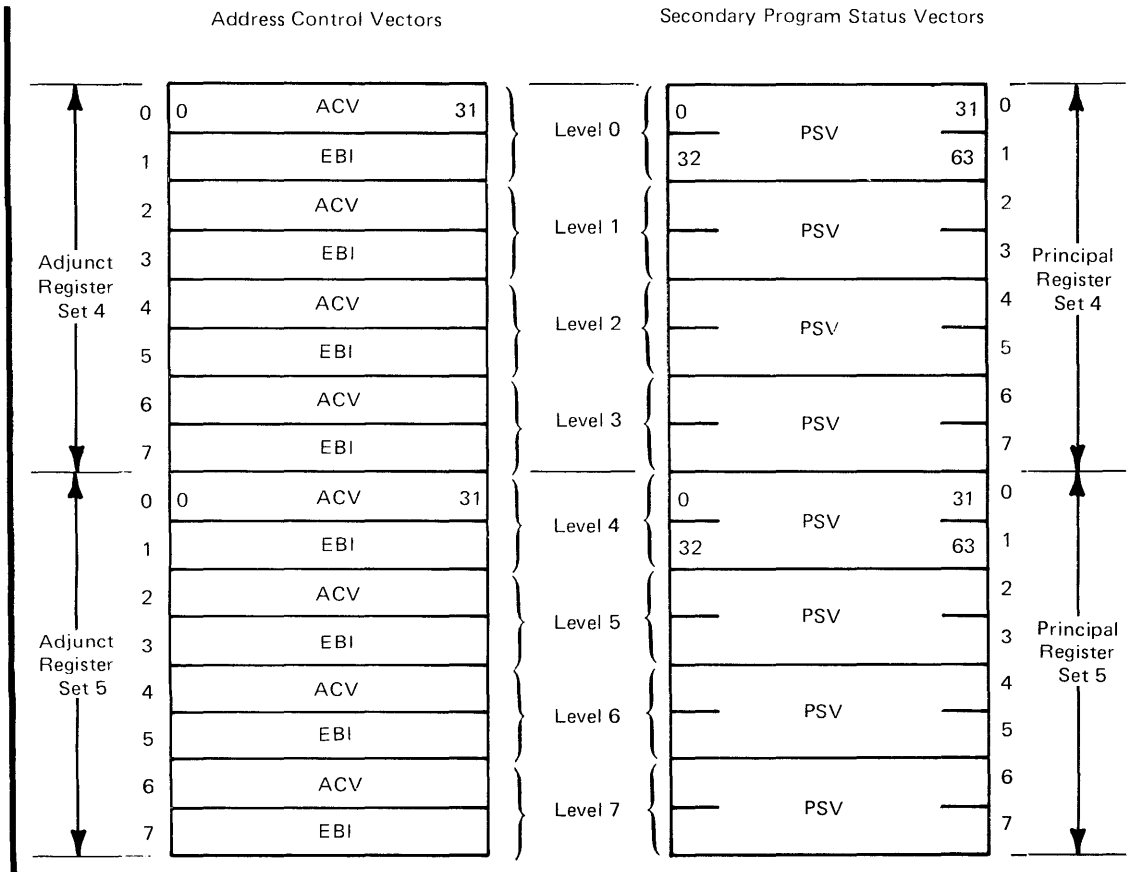


Figure 6-6. ACV/EBI Pairs Associated with Secondary PSVs

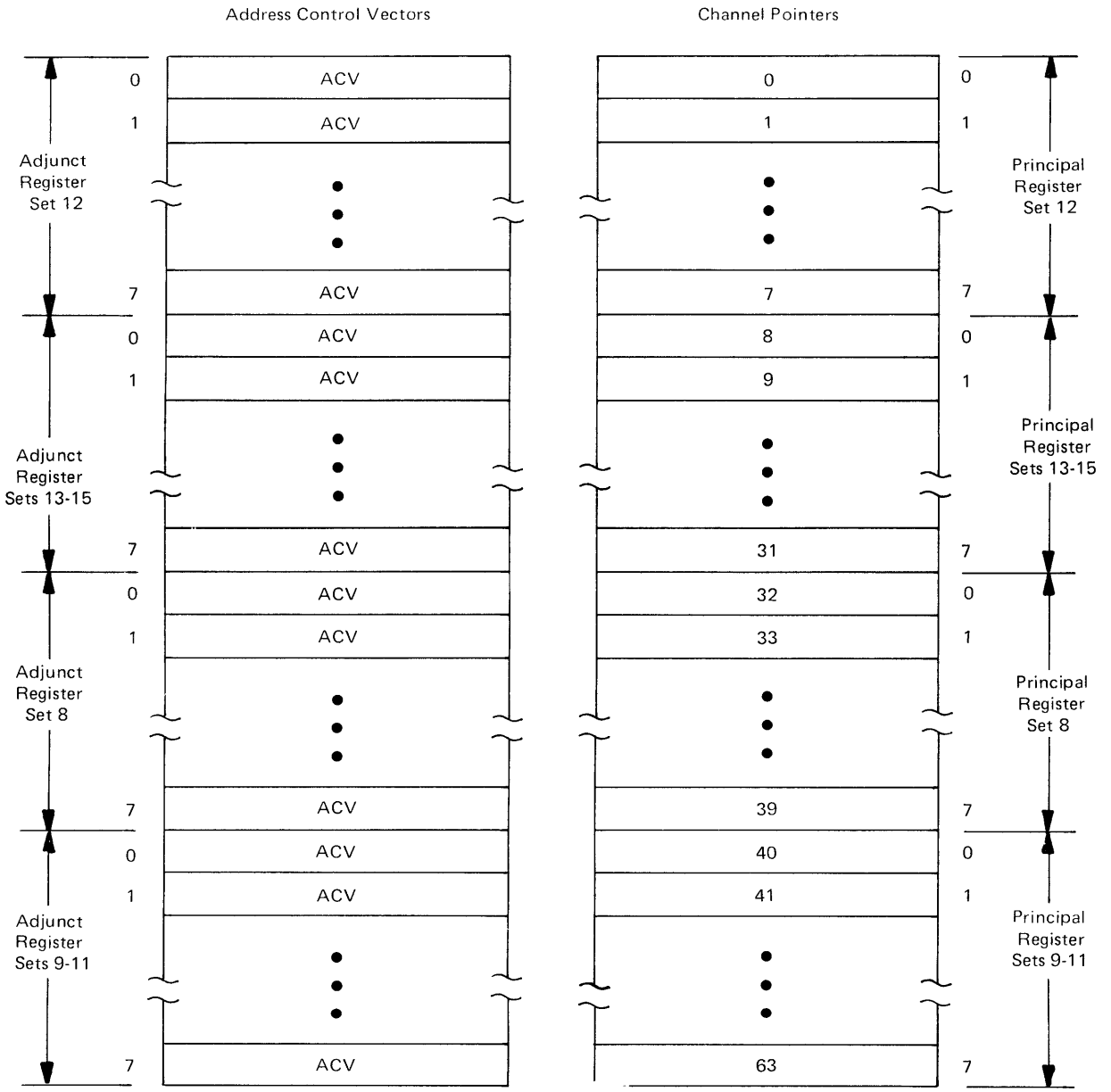


Figure 6-7. ACVs Associated with Channel Pointers

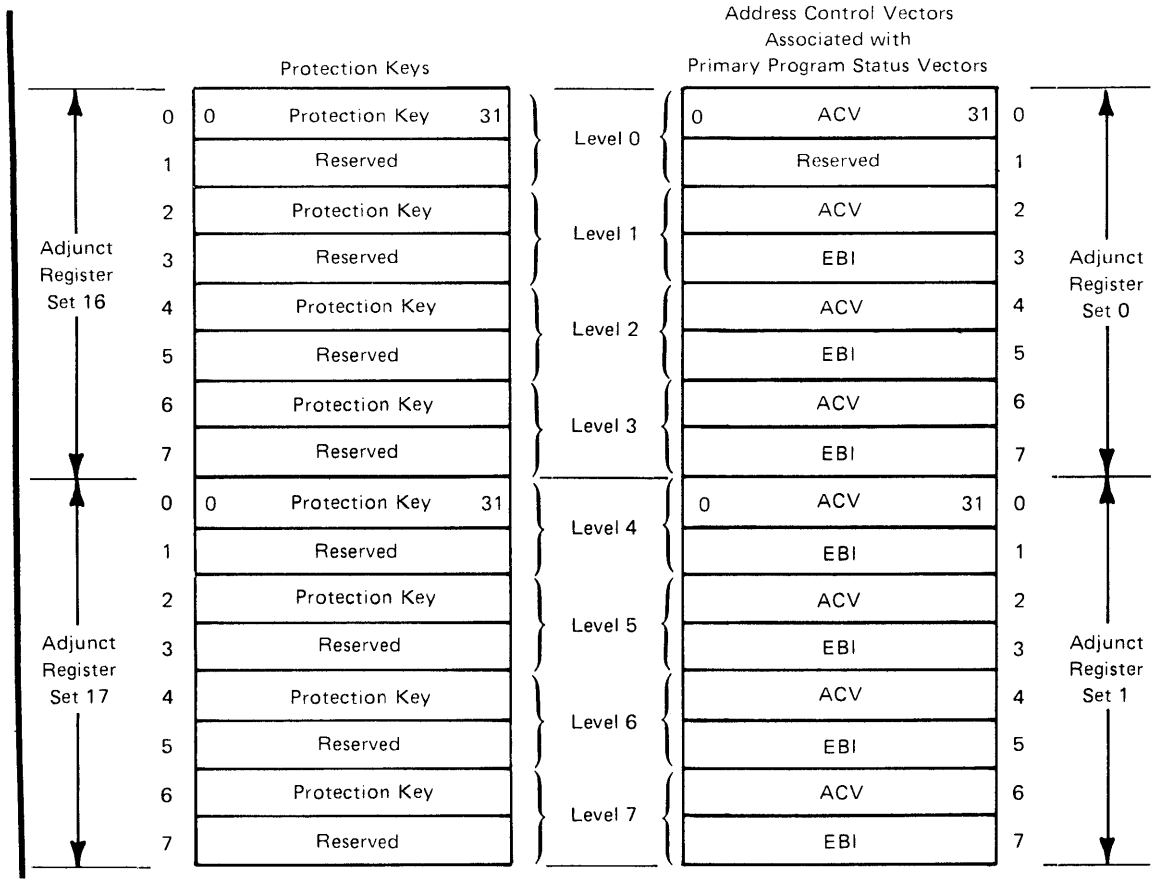


Figure 6-8. How Protection Keys Correspond to ACVs Associated with Primary PSVs

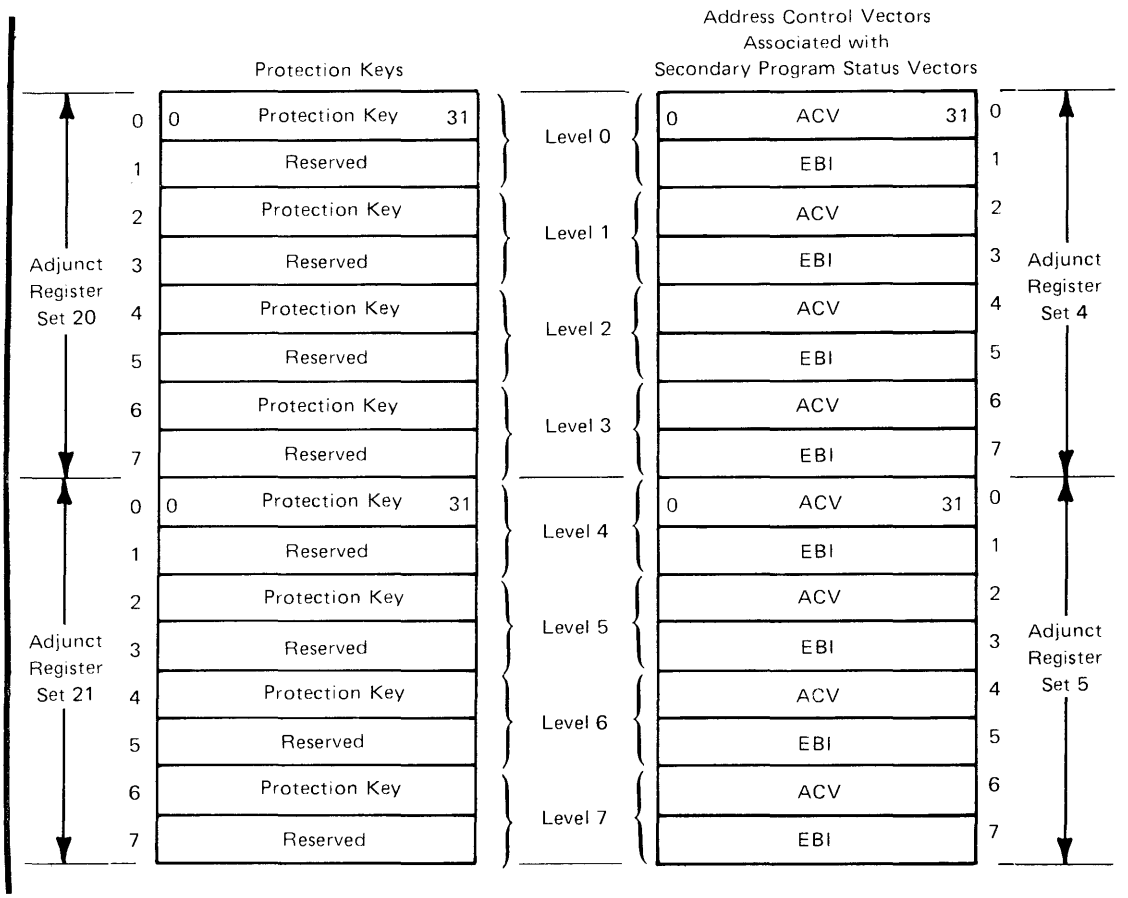


Figure 6-9. How Protection Keys Correspond to ACVs Associated with Secondary PSVs

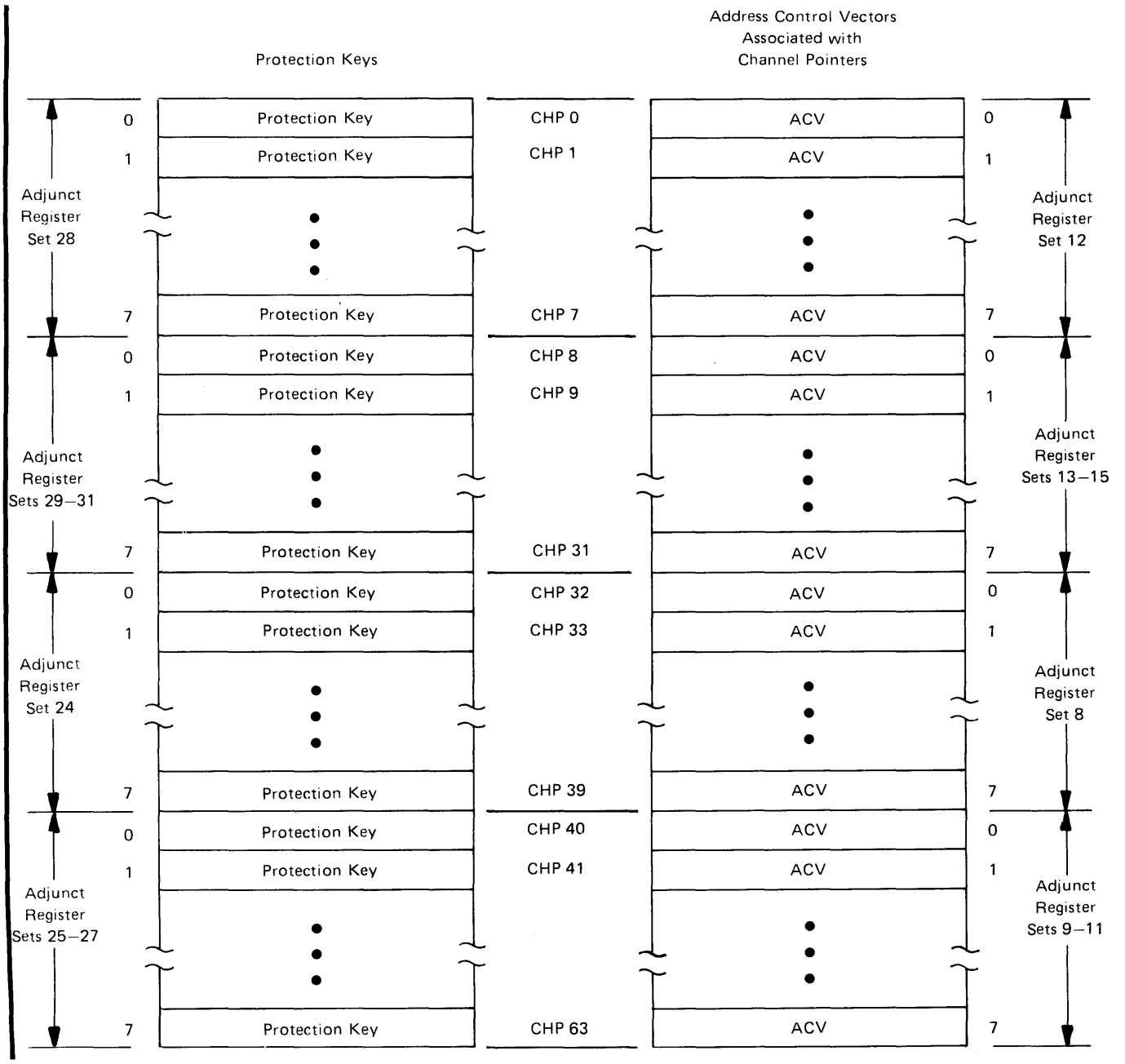


Figure 6-10. How Protection Keys Correspond to ACVs Associated with Channel Pointers

Floating-Point Registers

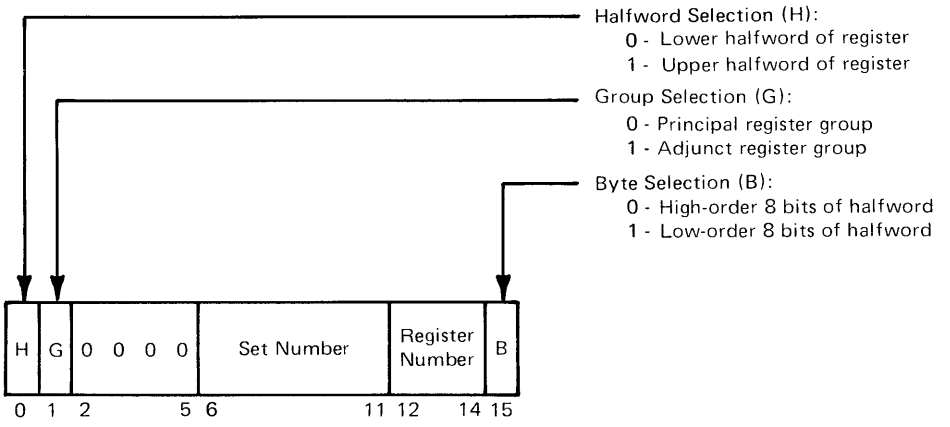
The group of floating-point registers consists of eight register sets numbered consecutively 0-7. All floating-point register sets are available for assignment to programs. A program has one floating-point register set assigned to it. A field in the floating-point status vector designates the number of the floating-point register set assigned to the program. The format and functions of the floating-point status vector are described in Chapter 9, "PCE Control."

Access to Register Contents

In general, instructions can refer to only the 16 active general-purpose registers assigned to the program. Exceptions exist for the supervisor-privileged register-indirect operations which can refer to any register in the principal and adjunct register groups. Information can be transferred between an active general register and any principal or adjunct register by means of the register-indirect instructions. These instructions can be executed only when the program-mode field of the current PSV specifies master or supervisor mode. Register-indirect operations are described under "Instructions" in this chapter.

Information in floating-point registers can be referred to using instructions provided with the floating-point feature. Only the contents of the active floating-point registers can be accessed.

The register-indirect operations use a *register-indirect addressing vector* to address a byte or halfword location in any principal or adjunct register. The register-indirect addressing vector is a 16-bit quantity organized as five fields, as follows:



The fields of the register-indirect addressing vector are allocated as follows:

- ***Halfword Selection (H):*** A value of 0 in bit position 0 selects the lower halfword of a register; a value of 1 selects the upper halfword. Bit position 0 must contain 0 for references to an adjunct register when the two high-order byte locations (bit positions 0-15) of the register are not available to the program.

- ***Group Selection (G):*** A value of 0 in bit position 1 selects the principal register group; a value of 1 selects the adjunct register group.

- ***Set Number:*** Bit positions 6-11 contain an unsigned, positive, binary integer that is the number of the principal or adjunct register set containing the register to be accessed.

- ***Register Number:*** Bit positions 12-14 contain an unsigned, binary integer that is the number of the register within a principal or adjunct register set.

- ***Byte Selection (B):*** A value of 0 in bit position 15 selects the byte contained in bit positions 0-7 of the halfword selected by bit 0 (H); a value of 1 selects the byte contained in bit positions 8-15. Bit position 15 is ignored for operations on halfword data units.

- ***Reserved Bits:*** Bit positions 2-5 of the register-indirect vector are reserved and must contain 0's.

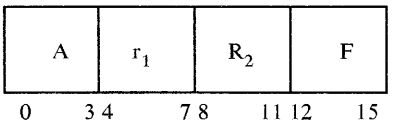
Programming Note: Bit positions 6-15 of the register-indirect addressing vector, considered as one field, represent a consecutive numbering of byte locations in either the upper or the lower half of all registers in a group.

Instructions

The register-indirect instructions, including their mnemonics, formats, and operation codes, follow. The procedure for describing the individual instructions, and the symbols used in the instruction formats and the expressions of operations, are defined under "Instruction Descriptions" in Chapter 4. Refer to Appendix B for an explanation of the assembler language notation used in the instruction descriptions.

LOAD (byte, register-indirect)

LRN rpb,ra



RR Format

Operation
$(r_1) \leftarrow RG[(R_2<16..31>)]$

Description
The byte at the second-operand location is placed unchanged in the first-operand location. The contents of the low-order 16 bit positions of the general register designated by the R_2 field are used as a register-indirect addressing vector specifying the second-operand location.

The first operand is located in the primary general register set. The second operand is located in any principal or adjunct register set.

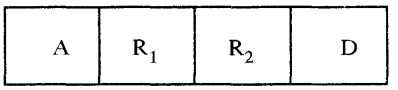
This instruction is supervisor-privileged.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (privileged operation)
Register-Indirect (operand 2)

LOAD (halfword, register-indirect)

LHRN rpb,ra



RR Format

Operation
$(R_1<16..31>) <— RG[(R_2<16..31>)]$

Description
The halfword at the second-operand location is placed unchanged in the first-operand location. The contents of the low-order 16 bit positions of the general register designated by the R_2 field are used as a register-indirect addressing vector specifying the second-operand location.

The first operand occupies the lower halfword of the general register designated by the R_1 field. The second operand is located in any principal or adjunct register set.

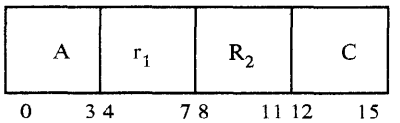
This instruction is supervisor-privileged.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (privileged operation)
Register-Indirect (operand 2)

STORE (byte, register-indirect)

STRN rpb,ra



RR Format

Operation
$RG[(R_2<16..31>)] <— (r_1)$

Description
The first-operand byte is stored unchanged at the second-operand location. The contents of the low-order 16 bit positions of the general register designated by the R_2 field are used as a register-indirect addressing vector specifying the second-operand location.

The first operand is located in the primary general register set. The second operand is located in any principal or adjunct register set.

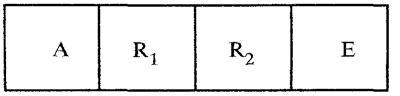
This instruction is supervisor-privileged.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (privileged operation)
Register-Indirect (operand 2)

STORE (halfword, register-indirect)

STHRN rh,ra



RR Format

Operation
$RG[(R_2<16..31>)] \leftarrow (R_1<16..31>)$

Description
The halfword first-operand is stored unchanged at the second-operand location. The contents of the low-order 16 bit positions of the general register designated by the R_2 field are used as a register-indirect addressing vector specifying the second-operand location.

The first operand occupies the lower halfword of the general register designated by the R_1 field. The second operand is located in any principal or adjunct register set.

This instruction is supervisor-privileged.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (privileged operation)
Register-Indirect (operand 2)

Chapter 7. Dynamic Address Relocation and Translation

This chapter describes the facilities of the IBM 8100 system PCE provided for storage management by a supervisory program. Included in this chapter are discussions of dynamic address relocation, dynamic address translation, storage protection, and the information used to control these facilities. Also described are the supervisor-privileged instructions that allow access to information used for address translation and storage protection.

Logical Addressing of Main Storage

The addressing arrangement of the PCE is based on a logical separation of the addresses used by the program and channel from the addresses assigned to the physical locations in main storage. A set of addresses used to refer to main storage is called an *address space.* Byte locations are numbered consecutively, left to right, from 0 to the maximum address defined for the address space. Each number is considered the address of the corresponding byte location.

Each program can have a logically distinct address space assigned to it. Similarly, logically distinct address spaces can be assigned to each channel I/O operation that refers to main storage. These distinct address spaces are called *logical address spaces.* All main-storage addresses used by the program or channel are 32 bits long and are treated as logical addresses — they are not used directly to refer to physical main storage.

The *real address space* is the set of addresses assigned to the physical main-storage locations. Byte locations in the physical main storage are numbered consecutively, left to right, from 0 to the highest-numbered installed location.

Also defined is an address space, called the *PCE address space,* which is the complete set of addresses provided by the PCE. The size of the PCE address space depends on the processor model. Depending on processor model, the PCE address space may have two sizes: one size that applies only when dynamic address translation is active and one that applies only when translation is not active.

During program execution and channel I/O operations, all logical addresses used to refer to main storage are always dynamically relocated in the PCE address space. When dynamic address translation is not active, the relocated addresses are used directly as real addresses to refer to main storage. When dynamic address translation is active, a translation table is used to assign blocks of relocated addresses to blocks of real main storage. The dynamic address relocation and translation facilities are described in the following sections.

Dynamic Address Relocation

During every storage reference, the logical address supplied by the program or channel is mapped into the PCE address space. This mapping process is called *dynamic address relocation,* and is illustrated in Figure 7-1.

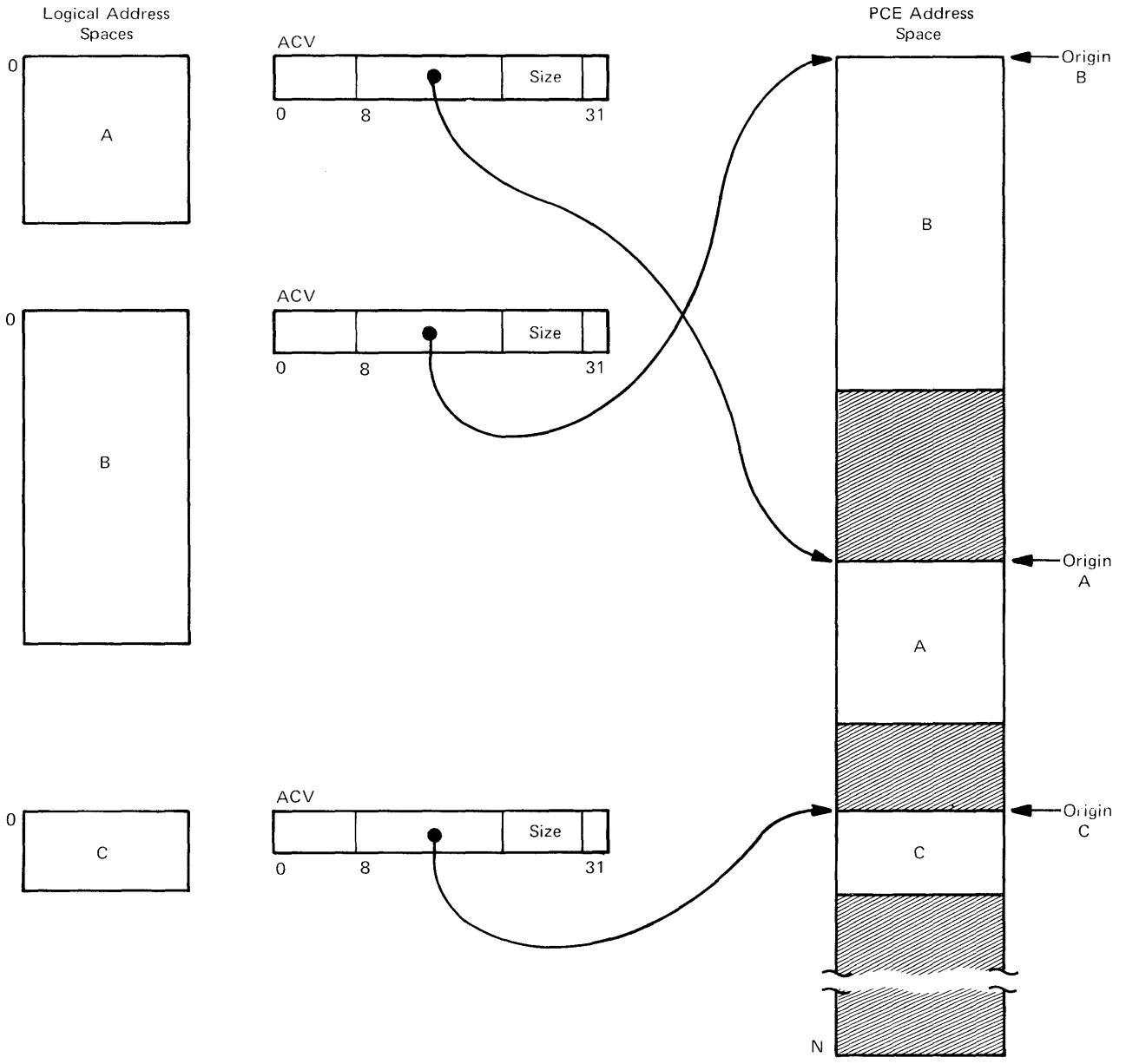


Figure 7-1. Dynamic Address Relocation

The information that controls dynamic address relocation is contained in an address control vector (ACV). The activation of dynamic address translation is also controlled by information in the ACV. One ACV is associated with each program status vector (PSV) and is made the *current* ACV when the corresponding PSV is introduced as the current PSV. The switching of PSV and ACV information is described in Chapter 9.

An ACV is also associated with each channel pointer. The logical address space for a channel I/O operation is defined by the ACV corresponding to the channel pointer used in the operation. This ACV is independent of, and does not replace, the current ACV which controls dynamic address relocation and translation during program execution. The ACV associated with a channel pointer controls dynamic address relocation and translation only within the scope of the channel

I/O operation which uses the corresponding channel pointer. The relationship of the channel pointer and ACV in channel I/O operations is described in Chapter 8, "Input/Output Operations."

Dynamic address relocation is controlled by two elements of information: the size of the logical address space, and its assigned beginning location in the PCE address space. The address in the PCE address space of the beginning of a logical address space, after relocation, is called the *origin* address. The relocation mechanism associates the contiguous addresses of a logical address space with a set of contiguous addresses in the PCE address space.

The address-relocation facility tests the logical address used in each storage reference. If the logical address exceeds the maximum address in the logical address space assigned to the program or channel I/O operation, an address exception is detected. If the logical address does not exceed the maximum address, it is combined with the origin address to produce a relocated address in the PCE address space.

Address Control Vector

An ACV is 32 bits long and contains an address-space origin field, an address-space size field, and a translation-control field (see Figure 7-2). Bit position 31 controls the operation of dynamic address translation and is described under "Dynamic Address Translation" in this chapter. Bit positions 8-30 are allocated to the two variable-length fields: origin and size. Bit positions 0-7 are reserved and must contain 0's.

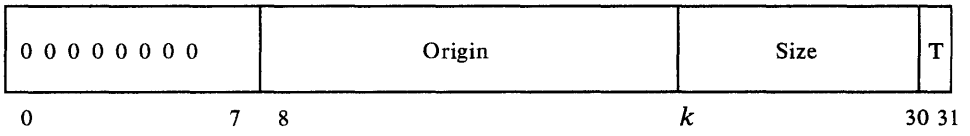


Figure 7-2. Address Control Vector

Address-Space Size

The size of a logical address space is represented with a variable-length code in the size field of the ACV. The size must be an integral power of 2 that is not less than 2^{11} (2048) and not greater than 2^{32} (4,294,967,296). The size of the PCE address space is also an integral power of 2 in the same range. The size of a logical address space, however, may not exceed the size of the PCE address space. The number of ACV bit positions used to represent the size increases by 1 for each power-of-2 increase in the size of the logical address space; the number of bits in the origin field decreases by 1 accordingly. The format of the ACV for each address-space size is given in Figure 7-3.

In Figures 7-2 and 7-3, bit-position k marks the variable boundary between the origin and size fields. Bit positions 8 through k-1 contain the origin; bit positions k through 30 contain the size. The boundary (bit position k) is determined by the address space size. For the smallest size (2048), 2 bits are used to define the size and k, therefore, is bit position 29. The maximum address-space size (4,294,967,296) requires 23 bits to define the size and k is bit position 8.

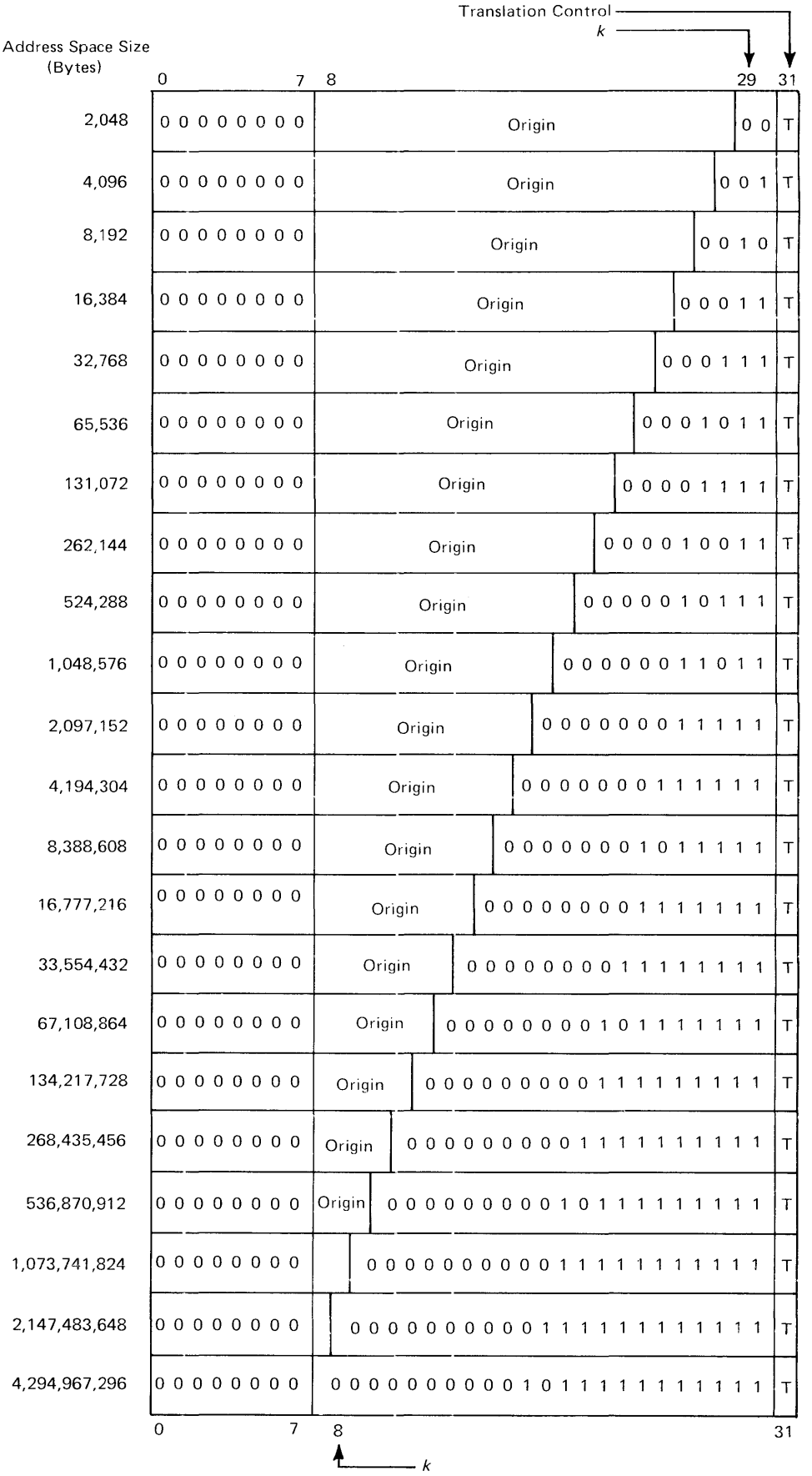


Figure 7-3. Address Control Vector Formats

Before a logical address is relocated, it is compared with the maximum address in the address space. If the logical address does not exceed the maximum address, it is relocated. A logical address does not exceed the maximum address when the required number of its high-order bits are 0's. The number of high-order bits that must be 0's is determined by the size of the address space, as explained in the following paragraph.

Figure 7-4 shows a representation of the 32-bit logical address used by the program or channel. In Figure 7-4, bit position *m* of the logical address marks the variable boundary to the left of which the high-order bits must be 0's. Bit positions 0 through *m*-1 must contain 0's; bit positions *m* through 31 contain the significant bits of the logical address. Bit position *m* may be any of bit positions 0-21, corresponding to the address-space sizes represented in the size field of the ACV, as shown in Figure 7-5.

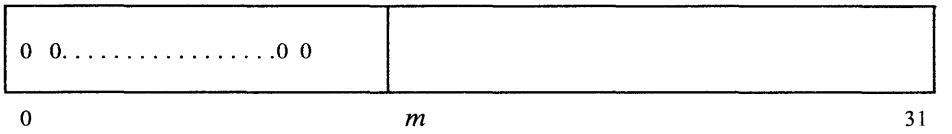


Figure 7-4. Logical Address Used by the Program or Channel

Address Space Size	*m*	*k*
2,048	21	29
4,096	20	28
8,192	19	27
16,384	18	26
32,768	17	25
65,536	16	24
131,072	15	23
262,144	14	22
524,288	13	21
1,048,576	12	20
2,097,152	11	19
4,194,304	10	18
8,388,608	9	17
16,777,216	8	16
33,554,432	7	15
67,108,864	6	14
134,217,728	5	13
268,435,456	4	12
536,870,912	3	11
1,073,741,824	2	10
2,147,483,648	1	9
4,294,967,296	0	8

Figure 7-5. Correspondence of *m* and *k* to Address Space Sizes

Address-Space Origin

The origin field in the ACV defines the beginning address (after relocation) of the logical address space within the PCE address space. A logical address space is located at an address in the PCE address space that is an integral multiple of the size of the logical address space. Thus, the representation of the origin address is determined by the size of the logical address space. The origin address necessarily has a number of low-order 0's equal to the number of low-order 0's in the binary representation of the address-space size (which must be a power of 2 between 2^{11} and 2^{32}, inclusive).

A 32-bit representation of an origin address is shown in Figure 7-6. Bit position *m* marks the variable boundary to the right of which all bits, including *m,* are 0's. Thus for the smallest address-space size (2048) *m* is bit position 21 and bits 21-31 of the origin address are 0's. The largest size (4,294,967,296) has an origin address of 0 and *m* is bit position 0. The correspondence of *m* to address-space sizes is shown in Figure 7-5.

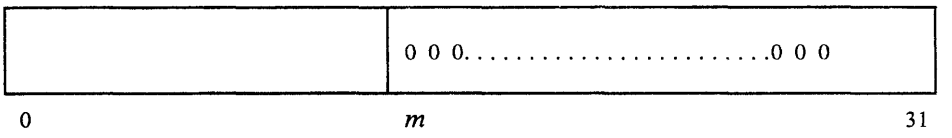


Figure 7-6. Origin Address

Bits 0 through *m*-1 of the origin address can be used directly in bit positions 8 through *k*-1 of the ACV to represent the origin. Bits *m* through 31, which are 0's, are not explicitly represented in the origin field of the ACV. The origin address cannot exceed the maximum address in the PCE address space. Thus, the origin field necessarily contains a number of high-order 0's equal to the number of high-order 0's in a 32-bit binary representation of the maximum address in the PCE address space.

Programming Notes:

- ACV information is held in permanently assigned adjunct-register locations. These register locations are described in Chapter 6, "Register Organization," as are the supervisor-privileged instructions used to access the adjunct-register locations.

- Depending on processor model, certain byte locations in adjunct registers are not necessarily made available to the program when the locations are used only to hold ACV information represented by all 0 bits. In particular, the high-order byte location (bits 0-7) of an adjunct register is not necessarily made available on any processor model. This byte location may not be made available because it is used only to hold reserved (0) bits in an ACV. Further, the two high-order byte locations (bits 0-15) are not made available on any processor model having a PCE address space that is never larger than 16,777,216 bytes. These two byte locations are not made available because they always contain 0's when the adjunct register is used to hold an ACV describing a logical address space of 16,777,216 or fewer bytes. When an ACV is referred to by the PCE or channel, unavailable byte locations in the adjunct register are assumed to contain 0's.

Relocation Process

Address relocation is performed during each reference to main storage by the program or channel. The relocation process yields a relocated address in the PCE address space. The relocated address is obtained by concatenating the origin field from the ACV with bits *m* through 31 from the logical address, with the origin field forming the high-order part. The relocation process is shown in Figure 7-7.

When dynamic address translation is active (bit 31 of the ACV is 1), the relocated address is translated into the corresponding real address before the storage reference. When dynamic address translation is not active (bit 31 of the ACV is 0), the relocated address is used directly as a real address to refer to a location in the physical main storage. If an attempt is made to refer to a physical main-storage location that is not installed, a specification exception is detected.

Programming Note: Physical main-storage locations are installed, and real addresses are assigned, in integral multiples of 2048 bytes. The PCE address space size, however, is always a power of 2 between 2^{11} and 2^{32}, inclusive. Therefore, in order to allow references to all physical locations, the size of the PCE address space is always equal to or greater than the size of installed main storage.

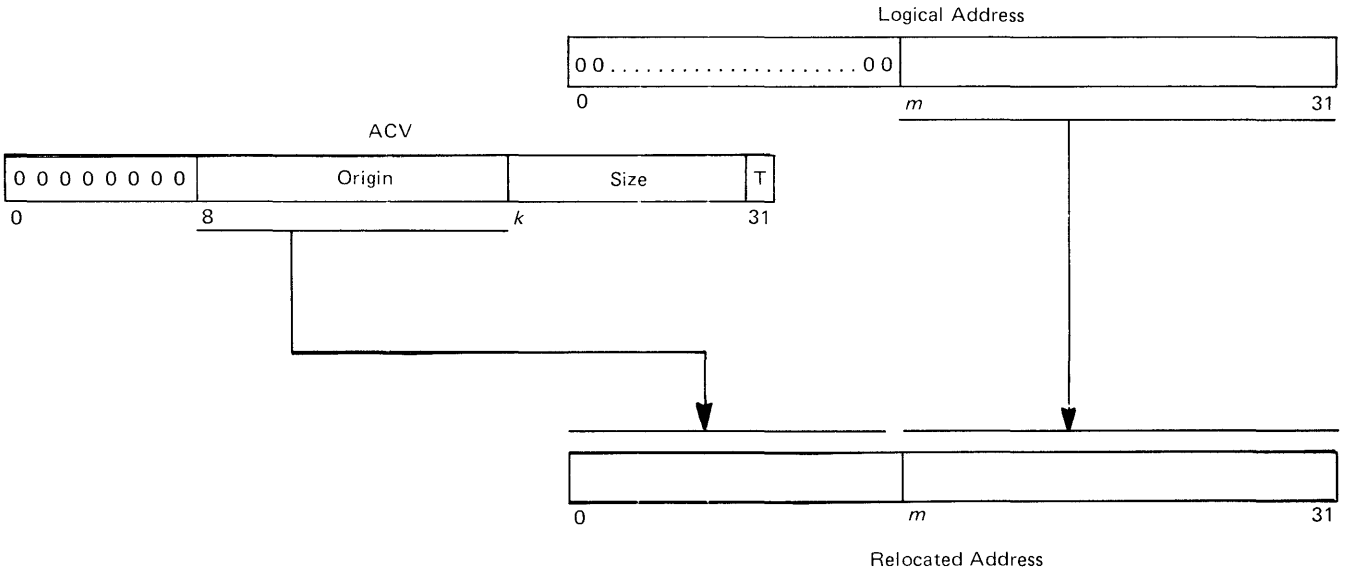


Figure 7-7. Dynamic Address Relocation Process

Dynamic Address Translation

Dynamic address translation provides the ability to assign noncontiguous blocks of real storage addresses to a set of contiguous logical storage addresses. The translation function is performed without change or inspection of the program and its data, does not require any explicit programming conventions, and does not disturb the execution of the program. Also provided with this facility are means for controlling access to storage.

Address translation is controlled by the translation-control field (bit position 31) in the ACV. When this bit is 1, translation is specified; when this bit is 0, no dynamic address translation takes place and relocated addresses are used as real addresses. When dynamic address translation is specified in the current ACV (during program execution) or in the ACV paired with a channel pointer (during a channel I/O operation), each relocated address is translated before a storage reference.

The unit of information recognized for dynamic address translation is the *block*. A block is a set of 2048 (2K) consecutive byte addresses beginning with an address that is a multiple of 2048. A relocated address that is to be translated, is

logically divided into a block-index field and a byte-index field. The block index starts with bit 0 of the address and extends through bit 20. The byte index comprises the remaining 11 low-order bits of the address as shown in Figure 7-8.

Relocated addresses are translated into real addresses by means of a *translation table,* which reflects the current assignment of real addresses. The assignment of real addresses occurs in units of blocks; byte addresses are assigned sequentially within a block. Each entry in the translation table associates a block of addresses in the PCE address space with a block of addresses in the real address space. The translation-table entries are organized in the same sequence as the adjacent blocks in the PCE address space; there is one entry in the translation table for each block in the PCE address space. Thus, the number of entries in the translation table is equal to the size of the PCE address space (with translation active) divided by 2048. The blocks assigned to adjacent blocks in the PCE address space need not be adjacent in the real address space.

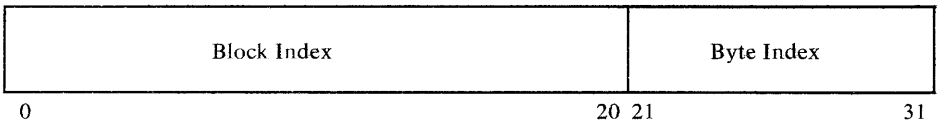


Figure 7-8. Block-Index and Byte-Index Fields of an Address to Be Translated

Translation-Table Entries

Entries in the translation table (see Figure 7-9) are 32 bits long and contain two fields: a block-address field, which provides the high-order bits of the real address; and an access-control field, which provides information for access protection.

Special considerations for the translation table in dual-PCE processors are described in Chapter 10, "Dual-Mode Processing".

The translation table is not addressed as part of main storage. Two instructions, LOAD FROM ADDRESS TRANSLATION TABLE and STORE TO ADDRESS TRANSLATION TABLE, are provided to access or modify information in the translation table. These instructions are supervisor-privileged.

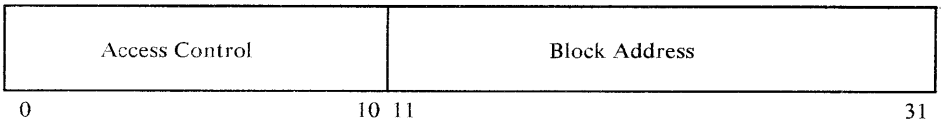


Figure 7-9. Translation-Table Entry

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

- ***Access Control:*** Bit positions 0-10 provide information used for storage access protection. The dynamic address translation facility provides protection against erroneous or unauthorized storing, instruction execution, or references of any type by the program or channel. Access exceptions are recognized for improper types of access. The operations of access protection are described under "Storage Access Protection" in this chapter.

- ***Block Address:*** Bit positions 11-31 provide the leftmost 21 bits of a real storage address. When the block address and the 11 bits from the byte-index field of the relocated address are concatenated, with the block address forming the high-order part, the real address is obtained.

A specification exception is recognized when the real address designated by a translation-table entry does not correspond to an installed physical location. Depending on the particular processor model and on the block-address value, either (1) a specification exception (for operand) is recognized when an attempt is made to store the entry, or (2) a specification exception (for real address) is recognized when the PCE attempts to refer to the uninstalled location.

Translation Process

Dynamic address translation is performed by means of the translation table. The block-index portion of a relocated address is used as an index to select an entry from the translation table. This entry contains the high-order bits of the real address that corresponds to the relocated address. The byte-index field is used unchanged for the low-order bit positions of the real address. The translation process is shown graphically in Figure 7-10.

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

Programming Note: When two or more relocated addresses are translated to the same real address, the results obtained are the same as if the relocated addresses were identical.

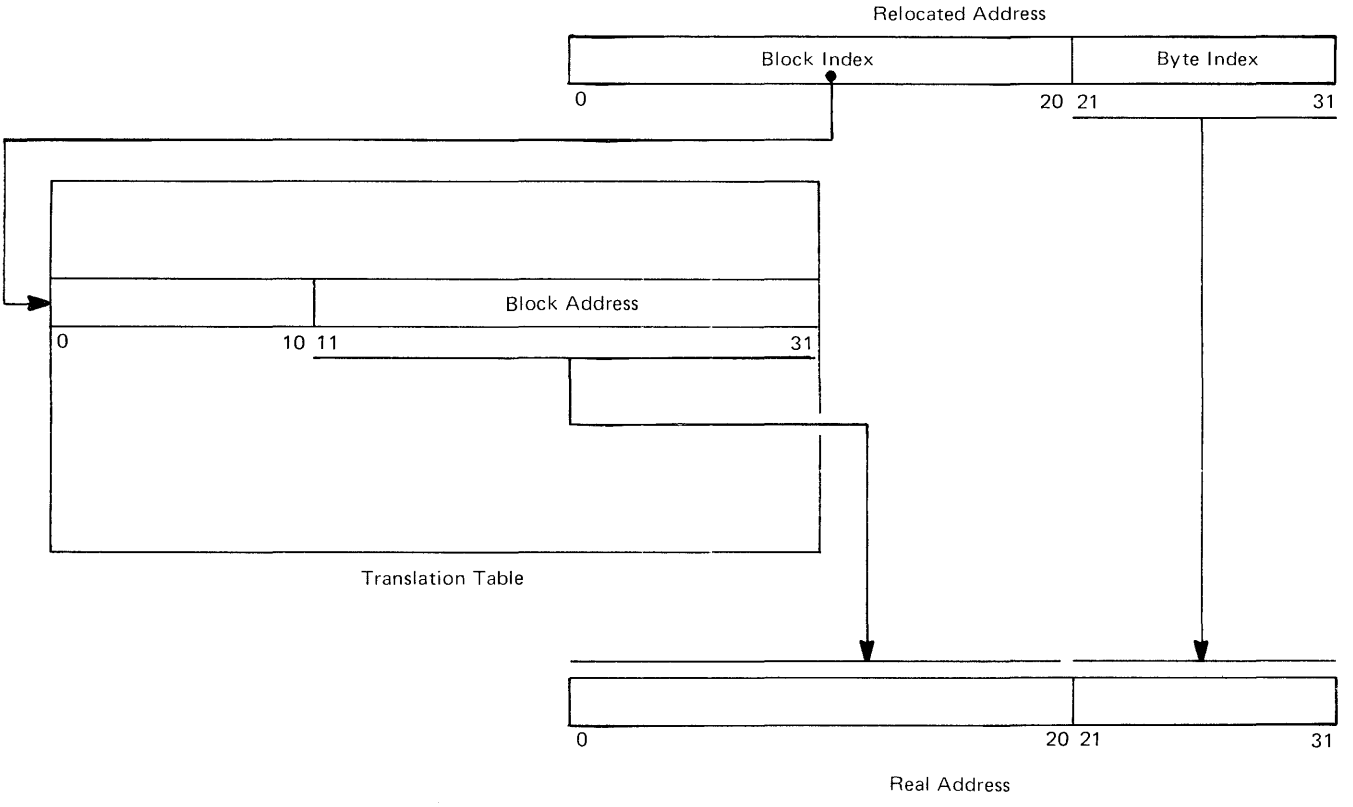


Figure 7-10. Dynamic Address Translation

Storage Access Protection

Dynamic address translation includes functions for protection against erroneous or unauthorized store operations, instruction execution, or references of any type to main storage. Access protection is provided only when dynamic address translation is specified in the ACV. Protection is applied to blocks in the PCE address space. The access-control field in the translation-table entry (see Figure 7-9) designates protection for the corresponding block. The format of the access-control field is given in Figure 7-11.

When an access exception is indicated, the block index of the PCE address in error is stored in the EBI register associated with the active ACV. The address is *not* stored after an exception caused by a channel-store protection violation.

The bit positions in the access-control field are allocated as follows:

- ***Block Invalid (I):*** Bit position 0 controls whether any reference is allowed: a 0 indicates that controlled access is permitted as specified by bit positions 1, 2, and 4 of the access-control field; a 1 in bit position 0 indicates that no access is permitted.

- ***Store Protection (S):*** Bit position 1 controls whether a store reference by the program is allowed: a 0 indicates that store references by the program are permitted; a 1 indicates that store references are not permitted.

- ***Execution Protection (E):*** Bit position 2 controls whether information may be fetched and interpreted as an instruction to be executed: a 0 indicates that the information may be interpreted as an instruction; a 1 indicates that the information may not be interpreted as an instruction.

- ***Channel-Store Protection (C):*** Bit position 4 controls whether a store reference by a channel I/O operation that refers to main storage is allowed: a 0 indicates that store references are permitted; a 1 indicates that store references are not permitted.

- ***Reserved Bits:*** Bit positions 3 and 5-10 are reserved and must contain 0's.

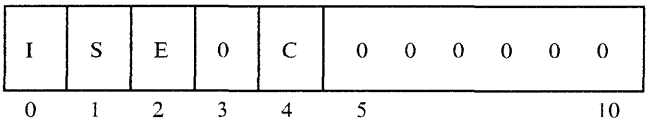


Figure 7-11. Format of Access-Control Field

When master mode is indicated in the program-mode field of the current PSV, all references to main storage by the program are allowed, regardless of the state of bit positions 1 and 2, provided that bit position 0 contains 0.

Each form of access protection is defined independently of the others, and more than one form may be designated by using multiple access-control bit positions. For example, 1's in bit positions 1 and 4 define protection for a block against all store references — from both program execution and channel I/O operations. If the type of access attempted by the program or channel is not permitted, an access exception is recognized.

The dynamic address relocation and translation facilities allow more than one logical address to be translated to the same real address. Accordingly, access protection may be defined to allow different types of access to the same real-storage location when it is referred to using different logical addresses.

Programming Note: All data-fetch references by the program or a channel I/O operation are allowed regardless of the state of bit positions 1, 2, and 4, provided that bit position 0 contains 0.

Separation Protection

Depending on processor model and only with dynamic address translation active, multiple programs can coexist within the logical address space defined by a particular ACV (see Figure 7-12). Translation locks and protection keys allow the logical separation of programs within this address space. The following paragraphs explain how protection keys and translation locks are used to provide separation protection.

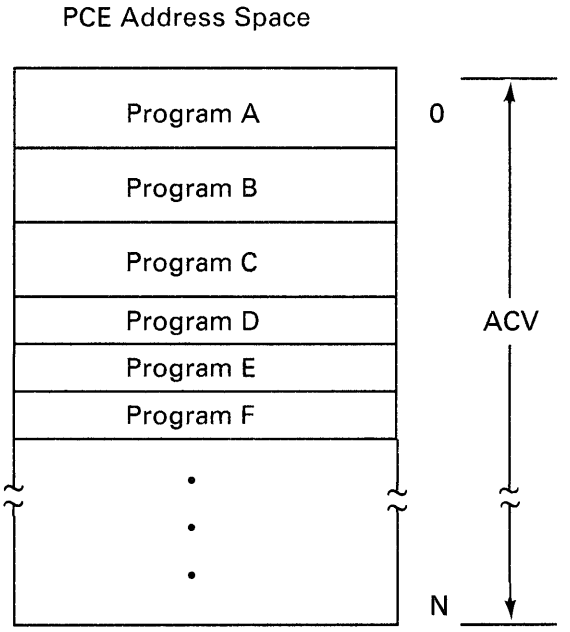


Figure 7-12. Multiple Programs within a Logical Address Space

Protection Keys

Protection keys, which are used only with dynamic address translation active, are contained in register sets within the adjunct register group. Each 8-bit protection key is logically associated with a PSV/ACV or CHP/ACV pair. Therefore, each 8-bit protection key can specify up to 255 unique user keys. This allows up to 255 unique programs and/or CHIO data buffers within the logical address space defined by a particular ACV. Additionally, a protection key value of all zeros (master protection key) allows references to any address within the logical address space.

Each key resides in the low-order halfword of its corresponding register and is offset 16 register sets from its associated PSV/ACV or CHP/ACV. Only bits 24-31 of the register contain the protection key; bits 16-23 must be zeros. Even-numbered registers are associated with a PSV/ACV, while both even- and odd-numbered registers are associated with a CHP/ACV. The supervisor-privileged register-indirect instructions are used to access the adjunct-register locations that hold protection key information. Refer to Figures 6-8, 6-9 and 6-10 for how protection keys correspond to ACVs.

Translation Locks

Translation locks, which are used only with dynamic address translation active, are contained in the translation lock table. Each 8-bit entry in this table is logically associated with a single entry in the translation table, while each translation-table entry corresponds to a specific 2K-byte block in the PCE address space. Therefore, each translation-lock-table entry can specify up to 255 unique translation-lock values, in addition to a value of all zeros. This zero value allows all programs and CHIO operations within the logical address space to access the corresponding block of main storage.

The LOAD FROM ADDRESS TRANSLATION LOCK TABLE and STORE TO ADDRESS TRANSLATION LOCK TABLE instructions are used to access or modify the translation-lock-table entries. However, these instructions may be executed only by a program that has the proper authorization as indicated by the program-mode field of its PSV.

Translation Lock and Protection Key Operation

Address relocation is performed during each program or CHIO reference to main storage. This process also checks the logical address to determine if it is within the logical address space defined by either the active ACV during program execution or the ACV paired with a channel pointer during CHIO operation. If the logical address is greater than the maximum available address, an address exception is indicated.

With dynamic address translation active, the translation-lock-table entry of the corresponding block in the PCE address space is compared to the protection key associated with the active ACV.

- If the values are identical, the lock value is zero, or the protection key value is zero, dynamic address translation continues as described earlier in this chapter.
- If none of these conditions exist, a program or channel exception occurs, causing an interruption. Additionally, during program execution, the stored PSV indicates a PEC value of 3 (separation exception) and, except for primary priority level 0, the block index of the PCE address in error is placed in the EBI register that corresponds to the active ACV.

Addresses Relocated and Translated

All addresses specified by the program and used to refer to main storage, for an instruction or an operand, are logical addresses. Logical addresses are always subject to dynamic address relocation and, when specified in the current ACV, are subject to dynamic address translation. Depending on processor model, this may also include the logical separation of programs through the use of translation locks and protection keys. Similarly, all main storage addresses used by the channel during channel I/O operations are logical addresses, are always subject to dynamic address relocation and, when specified in the ACV paired with the channel pointer, are subject to dynamic address translation. Also, depending on processor model, this may also include translation lock and protection key separation. Addresses indicated to the program on an interruption or as the result of executing an instruction, or addresses returned to the channel pointer at the end of a channel I/O operation, are logical addresses.

Programming Note: Relocation and translation are not applied to the logical address generated during execution of the LOAD ADDRESS instruction.

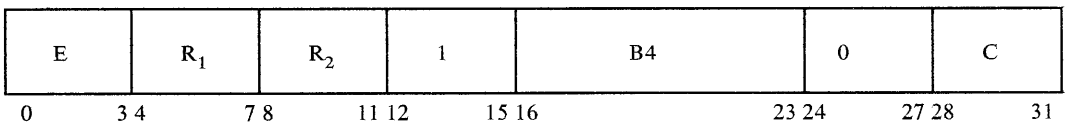
Translation-Table and Translation-Lock-Table Instructions

The LOAD FROM ADDRESS TRANSLATION TABLE, STORE TO ADDRESS TRANSLATION TABLE, LOAD FROM ADDRESS TRANSLATION LOCK TABLE, and STORE TO ADDRESS TRANSLATION LOCK TABLE instructions are provided to access and modify entries in their respective tables. These instructions are supervisor-privileged and can be executed only when the program-mode field of the current PSV specifies master or supervisor mode. A change to a table entry takes effect immediately. All main storage references, including those for instruction fetch, that are associated with an instruction following a STORE TO ADDRESS TRANSLATION TABLE are translated using the new table contents; all instructions that are associated with an instruction following a STORE TO ADDRESS TRANSLATION LOCK TABLE are logically separated using the new translation-lock-table contents.

The instructions for manipulating the translation table and translation lock table, including their mnemonics, formats, and operation codes, follow. The procedure for describing the individual instructions, and the symbols used in the instruction formats and the expressions of operations, are defined under "Instruction Descriptions" in Chapter 4. Refer to Appendix B for an explanation of the assembly language notation used in the instruction descriptions.

LOAD FROM ADDRESS TRANSLATION TABLE

LAT rw,ra



RR-Long Format

Operation

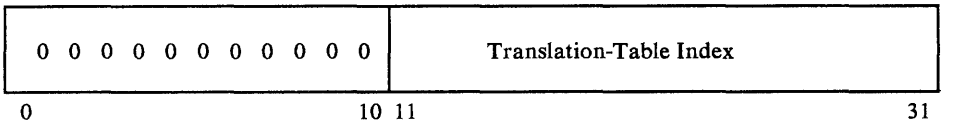
TEMP <— TT[(R_2<11..31>)]
(R_2<11..31>) <— (R_2<11..31>) + 1
(R_1) <— TEMP

Description

The translation-table entry at the second-operand location is placed unchanged in the first-operand location.

The initial contents of bit positions 11-31 of the general register designated by the R_2 field are used as an index into the translation table. The index is incremented by 1 and placed back in bit positions 11-31 of the general register after the translation-table entry is fetched and before it is placed in the first-operand location. The contents of bit positions 0-10 of the general register remain unchanged.

The format of the general register specified by the R_2 field is:



Bits 0-10 are reserved and should be all 0's. Bits 11-31 contain the translation-table index. On processor models that provide PCE address spaces less than 134,217,728 bytes, bits 16-31 contain the translation-table index and bits 11-15 may not be used during execution of this instruction. In this case, bits 11-15 remain unchanged. When bits 11-15 are not used, all 16 high-order bits of the register contents (bits 0-15) are reserved and should be all 0's. Depending on processor model, a specification exception may be indicated when bits 0-15 are not all zeros.

This instruction is supervisor-privileged.

A specification exception is recognized when the initial index value exceeds the number of entries in the translation table provided by the PCE.

Bit positions 24-27 of the instruction are reserved and must contain 0's; otherwise, an operation exception is recognized.

Result Conditions
The conditions remain unchanged.

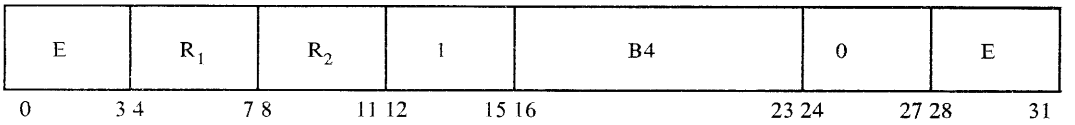
Program Exceptions (Suppression)
Operation (privileged operation; bits 24-27 of instruction not 0)
Specification (operand 2: invalid translation-table index or, depending on processor model, bits 0-15 of register R_2 not all 0)

Programming Notes
The translation-table index corresponds to the block-index field of a relocated address. The index designates the number of an entry in the translation table. Entries are numbered sequentially from 0 to the highest-numbered block in the PCE address space.

If the same general register is specified in both the R_1 and R_2 fields, the incremented index is overwritten by the entry fetched from the translation table.

LOAD FROM ADDRESS TRANSLATION LOCK TABLE

LATL rh,ra



RR-Long Format

Operation

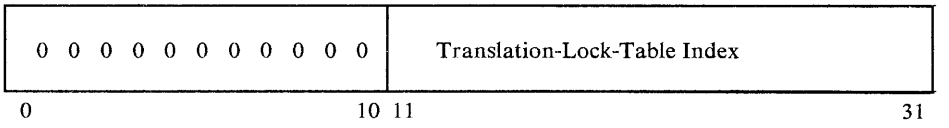
TEMP <— TL[(R_2<11..31>)]
(R_2<11..31>) <— (R_2<11..31>) + 1
(R_1<16..23>) <— 00000000
(R_1<24..31>) <— TEMP

Description

The translation-lock-table entry at the second-operand location is placed unchanged in bit positions 24-31 of the first-operand location; bits 16-23 are set to zeros.

The initial contents of bit positions 11-31 of the general register designated by the R_2 field are used as an index into the translation lock table. The index is incremented by 1 and placed back in bit positions 11-31 of the general register after the translation-lock-table entry is fetched and before it is placed in the first-operand location. The contents of bit positions 0-10 of the general register remain unchanged.

The format of the general register specified by the R_2 field is:



Bits 0-10 are reserved and must be all 0's; otherwise a specification exception is indicated. Bit positions 11-31 contain the translation-lock-table index. A specification exception is also indicated when the index value exceeds the number of entries in the translation lock table provided by the PCE.

This instruction is supervisor-privileged.

Bit positions 24-27 of the instruction are reserved and must contain 0's; otherwise, an operation exception is recognized.

Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)

Operation (privileged operation; bits 24-27 of instruction not 0)
Specification (operand 2: invalid translation-lock-table index, or bits 0-10 of R_2 not all 0)

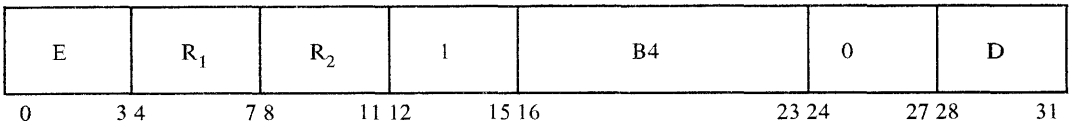
Programming Notes

The translation-lock-table index corresponds to the block-index field of a relocated address. The index designates the number of an entry in the translation lock table. Entries are numbered sequentially from 0 to the highest-numbered block in the PCE address space.

If the same general register is specified in both the R_1 and R_2 fields, the incremented index is overwritten by the entry fetched from the translation lock table.

STORE TO ADDRESS TRANSLATION TABLE

STAT rw,ra



Operation

$TT[(R_2<11..31>)] <— (R_1)$

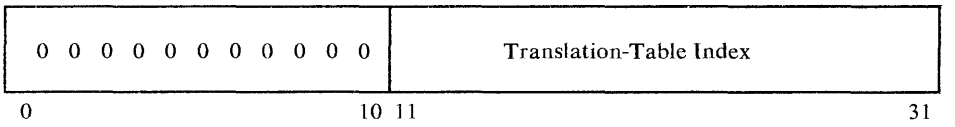
$(R_2<11..31>) <— (R_2<11..31>) + 1$

Description

The first operand is stored unchanged in the translation-table entry at the second-operand location.

The initial contents of bit positions 11-31 of the general register designated by the R_2 field are used as an index into the translation table. The index is incremented by 1 and placed back in bit positions 11-31 of the general register. The contents of bit positions 0-10 of the general register remain unchanged.

The format of the general register specified by the R_2 field is:



Bits 0-10 are reserved and should be all 0's. Bits 11-31 contain the translation-table index. On processor models that provide PCE address spaces less than 134,217,728 bytes, bits 16-31 contain the translation-table index and bits 11-15 may not be used during execution of this instruction. In this case, bits 11-15 remain unchanged. When bits 11-15 are not used, all 16 high-order bits of the register contents (bits 0-15) are reserved and should be all 0's. Depending on processor model, a specification exception may be indicated when bits 0-15 are not all zeros.

This instruction is supervisor-privileged.

A specification exception is recognized when the initial index value exceeds the number of entries in the translation table provided by the PCE. A specification exception is also recognized when an attempt is made to store an invalid translation-table entry.

Bit positions 24-27 of the instruction are reserved and must contain 0's; otherwise, an operation exception is recognized.

Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)

Operation (privileged operation; bits 24-27 of instruction not 0)

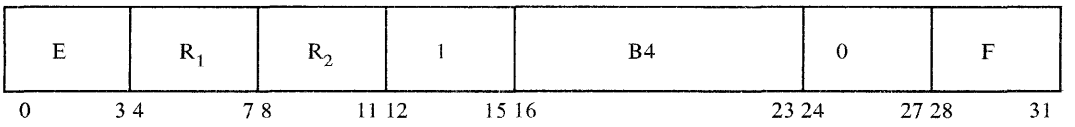
Specification (operand 1: invalid translation-table entry; operand 2: invalid translation-table index or, depending on processor model, bits 0-15 of register R_2 not all 0)

Programming Note

The translation-table index corresponds to the block-index field of a relocated address. The index designates the number of an entry in the translation table. Entries are numbered sequentially from 0 to the highest-numbered block in the PCE address space.

STORE TO ADDRESS TRANSLATION LOCK TABLE

STATL rh,ra



RR-Long Format

Operation

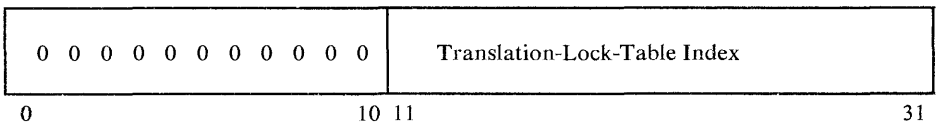
TL[(R_2<11..31>)] <— (R_1<23..31>)
(R_2<11..31>) <— (R_2<11..31>) + 1

Description

The first operand is stored unchanged in the translation-lock-table entry at the second-operand location. The translation-lock-table entry is contained in bit positions 24-31 of the general register specified by R_1. Bit positions 16-23 must contain zeros; otherwise, a specification exception is indicated.

The initial contents of bit positions 11-31 of the general register designated by the R_2 field are used as an index into the translation lock table. The index is incremented by 1 and placed back in bit positions 11-31 of the general register. The contents of bit positions 0-10 of the general register remain unchanged.

The format of the general register specified by the R_2 field is:



Bits 0-10 are reserved and must be all 0's; otherwise a specification exception is indicated. Bits 11-31 contain the translation-lock-table index. A specification exception is also indicated when the initial index value exceeds the number of entries in the translation lock table provided by the PCE.

This instruction is supervisor-privileged.

Bit positions 24-27 of the instruction are reserved and must contain 0's; otherwise, an operation exception is recognized.

Result Conditions

The conditions remain unchanged.

Program Exceptions (Suppression)

Operation (privileged operation; bits 24-27 of instruction not 0)
Specification (operand 1: bits 16-23 of R_1 not all 0's; operand 2: invalid translation-lock-table index, or bits 0-10 of R_2 not all 0)

Programming Note

The translation-lock-table index corresponds to the block-index field of a relocated address. The index designates the number of an entry in the translation lock table. Entries are numbered sequentially from 0 to the highest-numbered block in the PCE address space.

Chapter 8. Input/Output Operations

Input/output (I/O) operations involve the transfer of information between main storage and an external I/O device, or between general registers in the PCE and the external I/O device. I/O devices are logically attached to the PCE and main storage by means of a channel and adapters.

The channel provides the logical data path and control-signal path used for transferring information between adapters and the PCE or main storage. The adapters provide the logical attachment of I/O devices to the channel. Figure 8-1 illustrates the logical interconnection of I/O devices to the PCE and main storage. Depending on dual-PCE processor model, a channel is available on either one or both PCEs.

This chapter describes the control of I/O devices by the program and the channel. Formats are defined for the various types of I/O control information. The formats apply to all I/O operations and are independent of the type of I/O device, its speed, and its mode of operation.

The formats described include provisions for functions unique to some I/O devices. The way in which a device makes use of the format depends on the particular device.

Note: *Throughout this chapter, references are made to operations that are device specific, or to operations that depend on the particular device. Wherever such a reference appears, the user should refer to the 8100 System Library (SL) publication for the device for further details.*

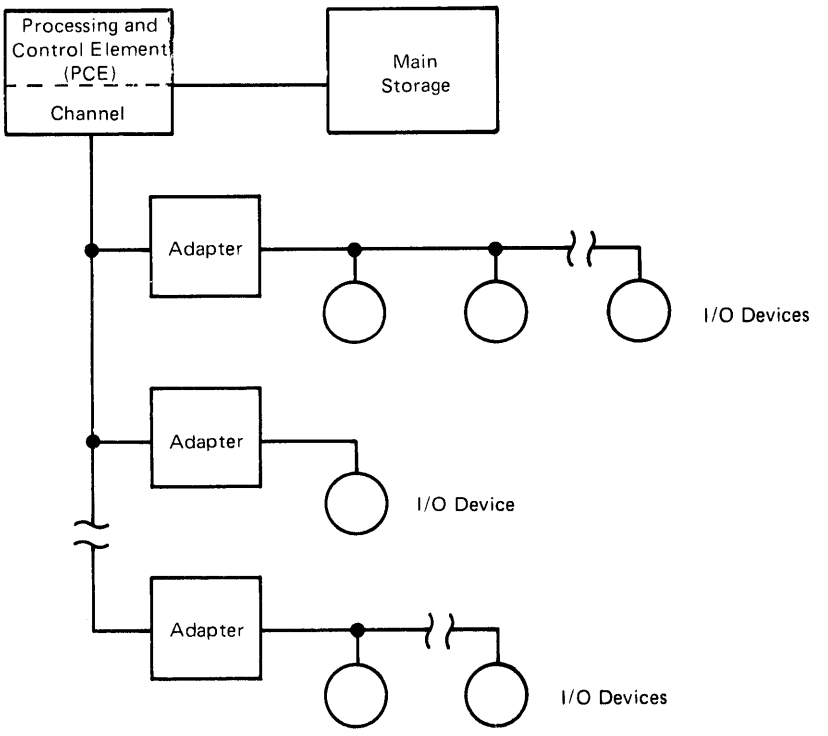


Figure 8-1. Logical Interconnection of I/O Devices to PCE and Main Storage

Attachment of Input/Output Devices

Input/Output Devices

Input/output devices provide either external storage or a means of communication between data processing systems or between a system and its users. Input/output devices include such equipment as card readers, card punches, magnetic tape units, direct-access storage devices (disks), typewriter-keyboard devices, printers, display devices, loop communication equipment, and telecommunication equipment.

Most types of I/O devices, such as printers, disk devices, or magnetic tape devices, deal directly with external media and are physically identifiable. Other types consist only of electronic equipment and do not directly handle physical recording media. The IBM SDLC Communications feature, for example, provides for the transmission of information between the 8100 system and a remote station, and its input and output are signals on a communication line.

An I/O device attaches to one adapter (see Figure 8-1). For some device types, two or more devices may be attached to a single adapter.

Adapters

The characteristics of an I/O device are adapted to the common form of control provided by the channel by means of an adapter. The adapter accepts control signals from the channel, controls the timing of data transfer to and from the channel, and provides indications concerning the status of the device. One or more devices may be attached to an adapter.

The I/O device attached to the adapter may be able to perform only certain limited operations, or it may perform many different operations. A typical operation is moving the recording medium and recording the data. To accomplish these functions, the device needs detailed signal sequences peculiar to the type of device. The adapter 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.

From the program's point of view, most functions performed by the adapter can be merged with those performed by the I/O device. Therefore, this publication normally does not make specific mention of the adapter 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 adapter only when emphasizing a function performed by it, or when sharing of the adapter among a number of devices affects the execution of I/O operations.

Channel

The channel provides for the logical attachment of different types of I/O devices, by means of their adapters, to the PCE and main storage. It accepts formatted control information and changes it into a sequence of signals acceptable to an adapter. For I/O operations to or from main storage, the channel maintains and updates an address that designates the destination or source of data in main storage. Similarly, when an I/O device signals an interruption, the channel

transforms the signal to I/O-interrupt-request information that can be used in the PCE. When the channel is not involved in an I/O operation, it monitors the attached devices for channel I/O service requests and I/O interrupt requests.

Types of Input/Output Operations

An I/O operation can be characterized in two ways:

- By the method used to transfer data
- By the size of the data unit transferred during each cycle of the I/O operation

Methods of Data Transfer

Two methods may be used to transfer data to or from an I/O device. They are called *programmed input/output* and *channel input/output.*

Programmed input/output refers to the transfer of data between the I/O device and the PCE. Specifically, the transfer occurs between the device and a register-operand location designated in the I/O instruction being executed by the PCE. Three I/O instructions are provided, two of which transfer a single byte of data to or from the I/O device; the third instruction transfers a single halfword (2 bytes). Programmed input/output operations are described in detail under "Programmed Input/Output."

Channel input/output refers to the transfer of one or more units of data between the I/O device and main storage. After the program initiates the channel I/O operation, the program is free to perform other work, and the channel and I/O adapter synchronize the transfer of data. The data transfer may begin immediately after the program initiates the channel I/O operation, or later. The moment at which the data transfer begins depends on the particular I/O device. Channel input/output operations are discussed in detail under "Channel Input/Output."

All devices can execute the programmed I/O operations described later in this chapter under "Programmed Input/Output Commands." The capability of executing other, device-specific programmed I/O operations, or of executing channel I/O operations, depends on the particular device.

Programming Note: Programmed I/O is used for such functions as reading status information from the device or writing control information to the device. Other functions may also be accomplished using programmed I/O, depending on the device. For a device that executes only programmed I/O operations, data is transferred to or from the device solely by means of I/O instructions. Channel I/O operations are executed by devices that must transmit or receive data at a high data rate.

Data-Unit Size

The amount of data transferred during each cycle of an I/O operation is either 1 or 2 bytes. For purposes of error detection, one check bit is transmitted with each byte. The check bit, called a *parity* bit, is generated automatically and is not directly controlled by, or made available to, the program. An I/O device that transmits or accepts 1 byte of data during a cycle of an I/O operation is said to operate in byte mode. Similarly, a device that operates in halfword mode transfers 2 bytes of data during a cycle of an I/O operation. The mode of an I/O device is normally fixed as part of its design.

Programmed Input/Output

Programmed I/O (PIO) operations are executed directly by means of I/O instructions. These instructions are decoded by the PCE and are part of the program. Three instructions are provided: INPUT/OUTPUT (byte), INPUT/OUTPUT (byte, immediate), and INPUT/OUTPUT (halfword). The instruction formats are common for all types of I/O devices. Each of the three instructions specifies a PIO address, a command, and a register-operand location from or into which data is transferred.

INPUT/OUTPUT (byte) and INPUT/OUTPUT (byte, immediate) designate a byte operand for data and may be used for operations with byte-mode and halfword-mode devices. INPUT/OUTPUT (halfword) specifies a halfword data operand and is provided for use with halfword-mode devices only. These three instructions are described in detail later in this chapter.

Execution of an I/O instruction consists of the logical selection (connection) of the addressed device, the transfer of the command to the device, and the transfer of one unit of data to or from the device. Execution of the I/O instruction is completed after the data unit is transferred. The condition indicators in the current PSV are set to reflect certain results of the PIO operation. Refer to "Result Conditions" in this chapter for a complete discussion of the indicated results.

When an error is detected during execution of an I/O instruction, execution is terminated and a system-check interruption is generated. The cause of the interruption is indicated with the error interrupt request vector, in conjunction with status information provided by the device. Refer to "System-Check Interruption" in Chapter 9, "PCE Control," for the meanings of the bits in the error interrupt request vector. The status information provided by the device is described under "Basic Status Register" in this chapter. Device-specific status information is described in the SL publication for the particular device.

Programming Note: A PIO operation consists of the execution of an I/O instruction, and the PIO operation is concluded when the I/O-instruction execution is completed (before execution of the next sequential instruction). The PIO operation may cause a subsequent I/O interruption, depending on the type of operation and the particular device. I/O interruptions, however, are not a specified part of PIO operations.

Compatibility of Operation

As part of the execution of an I/O instruction, the channel informs the device that the PIO operation is to be executed as either a byte-mode or a halfword-mode operation: byte mode for the INPUT/OUTPUT (byte) and INPUT/OUTPUT (byte, immediate) instructions, and halfword mode for the INPUT/OUTPUT (halfword) instruction. When the operating mode of the device is compatible with the instruction, the results obtained by the program are those described for the individual instruction. When the device and instruction are not compatible, the operation is considered invalid.

INPUT/OUTPUT (byte) and INPUT/OUTPUT (byte, immediate) are compatible with both byte-mode and halfword-mode devices, whereas INPUT/OUTPUT (halfword) is compatible only with halfword-mode devices. When a byte-mode device is addressed with INPUT/OUTPUT (halfword), the results are unpredictable.

Programming Note: Care should be taken to ensure that the PIO address specified with INPUT/OUTPUT (halfword) is assigned to a halfword-mode device.

Programmed Input/Output Addressing

The following description of programmed I/O addressing is presented in terms of the I/O device. Logically, devices that do not share an adapter with other devices are not distinguishable from their adapter, and both are identified by the same PIO address. When two or more devices are attached to a single adapter, the PIO address is assigned to the adapter. In this situation, the address of the device appears either in the command code or as part of the data, depending on the operation and the particular adapter and device.

An I/O device is designated by a PIO address. The address is an 8-bit binary number associated with the device. The PIO address is specified by a general-register operand of an I/O instruction. For INPUT/OUTPUT (byte) and INPUT/OUTPUT (halfword), the register operand is designated in the instruction; for INPUT/OUTPUT (byte, immediate), the operand is contained in an implied register.

The PIO address identifies the particular I/O device and adapter attached to the channel. Any number in the range 0-255 can be used as a PIO address. A device may be assigned only one PIO address.

When an I/O instruction is executed, one of three situations relating to the specification of the PIO address is possible:

- The I/O instruction specifies the address of a device whose operating mode (byte or halfword) is compatible with the instruction. In this case, the device becomes selected for the operation.

- The I/O instruction specifies the address of a device whose operating mode is not compatible with the instruction. In particular, this occurs when the PIO address specified with INPUT/OUTPUT (halfword) is assigned to a device that operates in byte mode. The resulting operation in this situation is unpredictable. (See the discussion of compatibility under "Compatibility of Operation" in this chapter.)

- The I/O instruction specifies the address of a device that either is not configured to the system or is not operational. No device is selected, and a system-check interruption occurs with I/O timeout check indicated.

Programmed Input/Output Commands

PIO commands are designated by an 8-bit command code and specify to the I/O device the PIO operation to be performed. The command code is specified as a register operand in INPUT/OUTPUT (byte) and INPUT/OUTPUT (halfword), and as an immediate operand in INPUT/OUTPUT (byte, immediate).

The low-order bit of the command code identifies to the channel the direction of data transfer. When the low-order bit is 0, the channel transfers the data from the PCE to the device; when the bit is 1, the channel transfers the data from the device to the PCE. The channel does not decode the high-order 7 bits of the command code.

All 8 bits of the command code are transferred to the I/O device. The high-order 7 bit positions specify how the command is to be executed. Except for the commands listed in the following table, the significance of the high-order 7 bit positions of the command code depends on the particular device:

Code	Command
0000 0010	Reset Device
0000 0100	Reset BSTAT Under Mask
0000 0110	Set BSTAT Under Mask
0000 0111	Read BSTAT

These four commands are executed by all devices. The command Reset Device is used to selectively reset I/O devices. The Reset Device COMMAND is described later in this chapter under "I/O Selective Reset." The remaining three commands are used to modify or inspect the basic status register (BSTAT) associated with the device. The basic status register is 8 bits long and indicates the status of the I/O device. The basic status register and the associated three PIO commands are described later in this chapter under "Basic Status Register."

The PIO commands listed in the preceding table can be issued to both byte-mode and halfword-mode devices using either INPUT/OUTPUT (byte) or INPUT/OUTPUT (byte, immediate). Whether the four commands can be issued to a halfword-mode device using INPUT/OUTPUT (halfword) depends on the device.

Any PIO command codes not listed above are considered device specific. The I/O instructions that may be used to issue a device-specific command to a halfword-mode device depends on the command and on the particular device. In some cases, any of the three I/O instructions may be used. In other cases, the device may execute the command properly only if it is issued with INPUT/OUTPUT (byte) or INPUT/OUTPUT (byte, immediate), and not if it is issued with INPUT/OUTPUT (halfword); or the device may properly execute the command only if INPUT/OUTPUT (halfword) is used. Whether a system-check interruption is generated when an improper I/O instruction is used depends on the device.

When more than one device is attached to a single adapter, Reset Device will reset the adapter and all attached devices. Similarly, the commands that refer to the basic status register pertain to the adapter and all attached devices.

Result Conditions

Certain conditions resulting from execution of PIO operations are indicated with the condition indicators in the current PSV. The result conditions are indicated when execution of the I/O instruction is completed. The result conditions indicate the completion of the PIO operation, whether a data check occurred during an input operation, and whether the device detected any exceptional condition during the PIO operation. The indication of an exceptional condition does not necessarily indicate an error, and it has only one meaning for any particular command and type of device.

The following result conditions are indicated only when a program-exception, system check, or system-check interruption does not occur because of the execution of the I/O instruction:

Result Condition	Meaning
8	--
4	--
2	PIO operation completed.
1	Data check on inbound data.
0	Exception indicated by device.

Result condition 2 is indicated when execution of the I/O instruction is completed. (Result conditions 8 and 4 are not indicated.)

Result condition 1 is indicated when: (1) the channel detects a data check (invalid parity) on the data transferred during an input operation, and; (2) the device indicates, by means of a control signal, that the system-check indication because of the data check, is to be suppressed. Correct parity is assigned to the data and the operation is completed. When the channel detects a data check during an input operation and the device does not signal that the system-check indication is to be suppressed, then: (1) the operation is suspended (storing of the data is inhibited) and; (2) a system check or a system-check interruption occurs with I/O control check indicated.

Result condition 1 is not indicated when an output operation is executed or when an input operation is executed without a data check.

Result condition 0 is indicated when the device signals the channel that it encountered an exceptional condition that normally does not occur. The reasons for the result condition 0 indication depend on the command and particular device.

Programming Note: The capability to cause result condition 0 to be indicated is device-specific.

Program-Exception Interruptions

Before the channel is signaled to execute an I/O instruction, the instruction is tested for validity by the PCE. A program exception detected at this time causes a program-exception interruption. The program-exception code in the stored PSV identifies the cause of the interruption.

An operation exception causes a program-exception interruption. This exception is indicated when an I/O instruction is encountered and the current PSV specifies application mode. The instruction is suppressed before the channel is signaled to execute it. The condition indicators in the PSV and the state of the addressed I/O device are not affected by the attempt to execute an I/O instruction while in application mode.

Abnormal Ending of Programmed Input/Output Operations

A PIO operation is terminated if one of the following occurs:

- The channel or PCE detects equipment malfunctioning related to the PIO operation.
- The channel receives no response or an incorrect control-signal response from the device.

A PIO operation is suspended when the channel receives invalid parity during an input operation and the device has not signaled that the system-check indication is to be suppressed.

At termination or suspension of the PIO operation, the channel signals halt to the device and logically disconnects the device. A system-check interruption is generated, with the type of error indicated in the error interrupt request vector (EIRV). The storing of data for input operations is inhibited. For output operations, data may be written to the device, depending on when in the sequence of the operation the error occurs. If the device had become selected, it acknowledges receipt of the halt signal by indicating equipment check in the basic status register associated with the device.

Either 1 or 2 bits in the EIRV are set to 1's, indicating the specific type of system check. The following system checks related to PIO operations are indicated in the EIRV:

EIRV Bit(s)	System Check
0	I/O Control Check (PIO operation is suspended)
1	I/O Timeout Check
3	Exception
5	Internal Control Check
2 and 5	Internal Data Check

System checks are described in detail under "System Checks" in Chapter 9, "PCE Control."

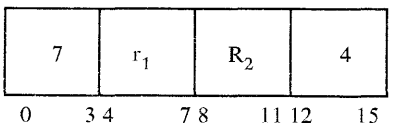
Instructions

The I/O instructions and their mnemonics, formats, and operation codes follow. The procedure for describing the individual instructions, and the symbols used in the instruction formats and the expressions of the operations, are defined under "Instruction Descriptions" in Chapter 4.

Note: *An assembler-language statement containing the mnemonic and the symbolic operand specifications is shown with each instruction. For a byte-mode operation using INPUT/OUTPUT (byte, immediate) as an example, "IOI" is the mnemonic and "rlpb,i8" are the operand specifications. In the instruction format for INPUT/OUTPUT (byte, immediate), the r_1 field is derived from the first operand specification, "rlpb"; the I_2 field, designating the second operand, is derived from "i8". Refer to Appendix B for an explanation of the assembler-language notation used in the instruction descriptions.*

INPUT/OUTPUT (byte)

IO rpb,rh



RR Format

Operation

IOD[(R_2<16..23>)] <— (R_2<24..31>)
If (R_2<31>) = 0
  Then IOD[(R_2<16..23>)] <— (r_1)
  Else (r_1) <— IOD[(R_2<16..23>)]

Description
A write or read operation is executed with the addressed I/O device. The instruction is executed only when the program mode field in the current PSV specifies master, supervisor, or I/O mode.

Bit positions 16-23 of the general register designated by R_2 contain the PIO address of the device to which the instruction applies. Bit positions 24-31 of register R_2 contain the command code. The low-order bit of the command code (bit position 31 of register R_2) designates whether the data is to be written to the device or read from the device: a 0 designates a write (output) operation, and a 1 designates a read (input) operation.

The I/O operation consists of selecting the addressed device, sending the command code to the selected device, and then transferring the byte of data. For a write operation, the byte of data is transferred from the first-operand location (designated by r_1) to the device. For a read operation, the direction of transfer is from the device to the first-operand location.

During a write operation to a halfword-mode device, two copies of the byte of data are concatenated together and transferred as a halfword. Whether the device uses both copies depends on the particular device. During a read operation from a halfword-mode device, the low-order 8 bits of the halfword of data transferred by the device are placed in the first-operand location, and the high-order 8 bits are ignored. Only the low-order 8 bits are inspected for data check.

The first operand is located in the primary register set.

Result Conditions
8 --
4 --
2 PIO operation completed.
1 Data check on inbound data (system-check indication suppressed).
0 Exception indicated by I/O device.

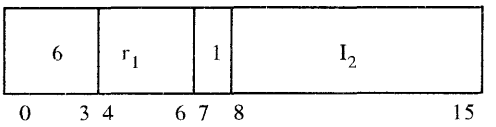
Program Exceptions (Suppression)
Operation (privileged)

Programming Notes
INPUT/OUTPUT (byte) may be used with either a byte-mode device or a halfword-mode device.

A data check on inbound data normally results in a system-check interruption, and the operation is suspended (storing of the data is inhibited). Otherwise, when the device causes the system-check indication to be suppressed, correct parity is assigned to the data and the operation is completed with result condition 1 indicated.

INPUT/OUTPUT (byte, immediate)

IOI rlpb,i8



RI Format

Operation

IOD[(PGR0<16..23>)] <— I_2
If I_2<7> = 0
  Then IOD[(PGR0<16..23>)] <— (r_1)
  Else (r_1) <— IOD[(PGR0<16..23>)]

Description

A write or read operation is executed with the addressed I/O device. The instruction is executed only when the program mode field in the current PSV specifies master, supervisor, or I/O mode.

Bit positions 16-23 of primary general register 0 contain the PIO address of the device to which the instruction applies. The immediate field, I_2, contains the command code. Bit 7 of the command code (bit position 15 of the instruction) designates whether the data is to be written to the device or read from the device: a 0 designates a write (output) operation, and a 1 designates a read (input) operation.

The I/O operation consists of selecting the addressed device, sending the command code to the selected device, and then transferring the byte of data. For a write operation, the byte of data is transferred from the first-operand location (designated by r_1) to the device. For a read operation, the direction of transfer is from the device to the first-operand location.

During a write operation to a halfword-mode device, two copies of the byte of data are concatenated together and transferred as a halfword. Whether the device uses both copies depends on the particular device. During a read operation from a halfword-mode device, the low-order 8 bits of the halfword of data transferred by the device are placed in the first-operand location, and the high-order 8 bits are ignored. Only the low-order 8 bits are inspected for data check.

The first operand is located in bit positions 24-31 (the lower byte-operand location) of a general register in the primary register set. This operand is designated by the r_1 field.

Bit position 7 of the instruction is used both as the low-order bit of a 4-bit r-field and as an extension of the operation code. In the latter case, the bit distinguishes this instruction from the CALL PSV instruction and the PCE-control (KI) instructions.

Result Conditions

8 --
4 --
2 PIO operation completed.
1 Data check on inbound data (system-check indication suppressed).
0 Exception indicated by I/O device.

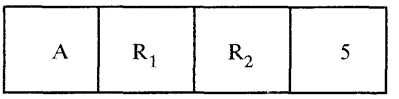
Program Exceptions (Suppression)
Operation (privileged)

Programming Notes
INPUT/OUTPUT (byte, immediate) may be used with either a byte-mode device or a halfword-mode device.

A data check on inbound data normally results in a system-check interruption, and the operation is suspended (storing of the data is inhibited). Otherwise, when the device causes the system-check indication to be suppressed, correct parity is assigned to the data and the operation is completed with result condition 1 indicated.

INPUT/OUTPUT (halfword)

IOH rh,rh



RR Format

Operation
IOD[(R_2<16..23>)] <— (R_2<24..31>)
If (R_2<31>) = 0
  Then IOD[(R_2<16..23>)] <— (R_1<16..31>)
  Else (R_1<16..31>) <— IOD[(R_2<16..23>)]

Description
A write or read operation is executed with the addressed I/O device. The instruction is executed only when the program mode field in the current PSV specifies master, supervisor, or I/O mode.

Bit positions 16-23 of the general register designated by R_2 contain the PIO address of the device to which the instruction applies. Bit positions 24-31 of register R_2 contain the command code. The low-order bit of the command code (bit position 31 of register R_2) designates whether the data is to be written to the device or read from the device: a 0 designates a write (output) operation, and a 1 designates a read (input) operation.

The I/O operation consists of selecting the addressed device, sending the command code to the selected device, and then transferring the halfword of data. For a write operation, the halfword of data is transferred from the first-operand location (designated by R_1) to the device. For a read operation, the direction of transfer is from the device to the first-operand location.

The first operand occupies the low-order 16 bit positions of the register designated by the R_1 field.

Result Conditions
8 --
4 --
2 PIO operation completed.
1 Data check on inbound data (system-check indication suppressed).
0 Exception indicated by I/O device.

Program Exceptions (Suppression)
Operation (privileged)

Programming Notes
INPUT/OUTPUT (halfword) should be used only with halfword-mode devices. Use of this instruction with a byte-mode device may produce unpredictable results.

A data check on inbound data normally results in a system-check interruption, and the operation is suspended (storing of the data is inhibited). Otherwise, when the device causes the system-check indication to be suppressed, correct parity is assigned to the data and the operation is completed with result condition 1 indicated.

Basic Status Register

Each adapter provides a basic status register (BSTAT). The information provided in the BSTAT identifies certain status conditions of the adapter and attached devices, such as equipment check, whether the devices are enabled or disabled, or whether a device is generating an I/O interrupt request. The BSTAT may also indicate device-specific status conditions, which are not described in this publication.

The basic-status information, and any additional status information that may be provided by the device, is usually made available to the program by means of programmed I/O commands such as Read BSTAT. Ordinarily, the handling of I/O interruptions by the program includes the reading of the BSTAT. By means of the basic-status information, the program can determine the cause of the interruption. The basic-status information also identifies errors that occurred during the last I/O operation.

The BSTAT is also used by the device as a control register. By means of the BSTAT, the device is either enabled or disabled for generating I/O interrupt requests or initiating channel I/O burst transfers. Two programmed I/O commands, Set BSTAT Under Mask and Reset BSTAT Under Mask, allow the program to modify the BSTAT and, thus, enable or disable the device for these functions.

The BSTAT is logically 8 bits long for both byte-mode devices and halfword-mode devices. However, fewer than 8 bits may actually be installed. For any particular device, only those bits of the BSTAT needed for proper indication of status information are necessarily provided. Bits not installed are read as 0's, and an attempt by the program to modify uninstalled bits is ignored.

Two bit positions of the BSTAT are designated as *equipment check* and *interrupt request* and are common to all devices providing this type of information. A third bit designated as *enabled* is provided by all devices. These three bit positions are defined below where the meaning applies when the respective bit is a 1:

BSTAT Bit	Designation
5	Equipment Check
6	Enabled
7	Interrupt Request

Bits 5-7 are defined in the following paragraphs. The meanings of the remaining (leftmost) bits, if any, provided by the device depend on the particular device.

Programming Note: An adapter may provide a status register that contains more than eight bit positions. In this case, the basic-status byte described above is contained in the low-order eight bit positions of the status register. For example, when a 16-bit status register is provided, the equipment-check, enabled, and interrupt-request bits are located in bit positions 13, 14, and 15, respectively.

Equipment Check

When bit 5 (equipment check) is a 1, the I/O device or adapter has detected an unusual condition that is detailed by the other bit positions of the BSTAT or by additional device-dependent status bits. Equipment check may indicate (1) that a programming error, such as an invalid command was detected, (2) that an equipment malfunction occurred, or (3) that an exceptional condition affecting the normal completion of the last operation occurred. The equipment-check bit represents a summary indication of the status conditions identified by device-dependent status information.

When the channel or PCE detects the error (such as an I/O timeout check or an internal control check), the channel notifies the I/O device of the error. The device executing the programmed I/O or channel I/O operation thereby sets the equipment-check bit and may, depending on the operation, set other status bits. In addition, the program is alerted of the error by means of a system-check interruption, with the type of error indicated in the EIRV. If the error occurs because of a channel I/O operation, the device also sets the interrupt-request bit.

When the device detects an invalid command during a programmed I/O operation, an equipment check is indicated in the BSTAT. The device then notifies the program by suppressing its response, thereby causing a system-check interruption with an I/O timeout check indicated. The invalid command may be due to invalid parity on the command code, or the command code may not be assigned for the I/O device. Similarly, equipment check is indicated and a system-check interruption due to I/O timeout occurs when the device detects invalid parity on outbound data during a programmed I/O or channel I/O operation.

Errors that originate at the I/O device cause equipment check to be indicated and, depending on the particular device and type of operation, usually cause the device to also set the interrupt-request bit. An I/O interruption occurs as a result of setting the interrupt-request bit. A system-check interruption is not normally generated for such errors.

Enabled

When bit 6 (enabled) is a 1, the device is enabled for the purpose of generating an I/O interrupt request or initiating a channel I/O operation. When this bit is 0, the device is disabled; that is, it is inhibited from performing these functions.

Whether enabled or disabled, the device executes the commands described under "Programmed Input/Output Commands" in this chapter. Whether the device executes other, device-specific, programmed I/O commands when it is disabled depends on the command and the particular device.

Programming Note: The state of the enabled bit is changed only (1) under program control by means of programmed I/O commands, or (2) when an I/O system reset occurs, which causes the bit to be made 0.

Interrupt Request

When bit 7 (interrupt request) is a 1, and the enabled bit is a 1, the device is presenting an I/O interrupt request. The interrupt request is associated with the priority level to which the device is assigned, and is signaled to the program by means of the I/O interrupt request vector. If the enabled bit is 0, the I/O interrupt request is held pending at the device and is not reflected in the I/O interrupt request vector. Details about the relation of the I/O interrupt request vector to the assignment of devices to priority levels are discussed in this chapter under "Input/Output Interruptions." The interrupt-request bit is set by the device when it detects a condition that should be brought to the attention of the program. The condition may be associated (1) with a discrete event detected by the I/O device during its execution of an I/O operation, (2) with an asynchronous condition that is significant to the program, or (3) with the conclusion of a channel I/O operation.

The interrupt-request bit may also be set by the device when it encounters an error that results in the setting of the equipment-check bit. In particular, the device sets both the interrupt-request and equipment-check bits when it encounters an error during the execution of a channel I/O operation.

PIO Commands Related to the BSTAT

The following three PIO commands are executed by all devices. These commands allow the program to modify or read the contents of the BSTAT associated with the I/O device.

Reset BSTAT Under Mask

When the Reset BSTAT Under Mask command is issued, the data operand specified in the I/O instruction is used as an 8-bit mask to selectively reset corresponding bits in the BSTAT associated with the device. The 8 mask bits correspond left to right with the bit positions of the BSTAT. A mask bit of 1 causes the corresponding basic-status bit to be made 0 (reset). A mask bit of 0 indicates that the corresponding basic-status bit is to remain unchanged. Any bit positions in the mask corresponding to bit positions in the BSTAT that are not provided by the device are ignored.

Depending on the particular device, some device-specific BSTAT bits may not be resettable by the program. Such bits may be reset indirectly when other status bits are reset, or when the device detects that the condition associated with the status indication is no longer present. Thus, the bit positions of the mask corresponding to these BSTAT bits are ignored. The equipment-check and interrupt-request bits, if provided, and the enabled bit can all be reset with this command.

When the Reset BSTAT Under Mask command is issued with the INPUT/OUTPUT (halfword) instruction to a halfword-mode device, the 8-bit mask is contained in the low-order byte of the designated halfword operand (bit positions 24-31 of the designated register). What use, if any, the I/O device makes of the high-order byte of the halfword operand depends on the specific device.

Set BSTAT Under Mask

When the Set BSTAT Under Mask command is issued, the data operand specified in the I/O instruction is used as an 8-bit mask to selectively set corresponding bits in the BSTAT associated with the device. The 8 mask bits correspond left to right with the bit positions of the BSTAT. A mask bit of 1 causes the corresponding basic-status bit to be made 1 (set). A mask bit of 0 indicates that the corresponding basic-status bit is to remain unchanged. Any bit positions in the mask corresponding to bit positions in the BSTAT that are not provided by the device are ignored.

Depending on the particular device, some device-specific BSTAT bits, and the equipment-check and interrupt-request bits, if provided, may not be settable by the program. Such bits may be set indirectly when other status bits are set, or when the device otherwise detects the condition associated with the status indication. Thus, the bit positions of the mask corresponding to these BSTAT bits are ignored. The enabled bit can be set with this command.

When the Set BSTAT Under Mask command is issued with the INPUT/OUTPUT (halfword) instruction to a halfword-mode device, the 8-bit mask is contained in the low-order byte of the designated halfword operand (bit positions 24-31 of the designated register). What use, if any, the I/O device makes of the high-order byte of the halfword operand depends on the specific device.

Read BSTAT

The Read BSTAT command causes the contents of the BSTAT associated with the addressed device to be placed in the data-operand location designated in the I/O instruction. Bit positions of the BSTAT that are not provided by the device are read as 0's.

When the Read BSTAT command is issued with the INPUT/OUTPUT (halfword) instruction to a halfword-mode device, the 8-bit BSTAT is placed in the low-order byte of the designated halfword operand (bit positions 24-31 of the designated register). What information, if any, is placed in the high-order byte of the halfword operand depends on the specific device.

Input/Output Interruptions

Input/output interruptions enable the PCE to change its state in response to conditions that occur in I/O devices. These conditions can be caused by the program or by an external event at the device. I/O devices are assigned to priority levels for the purpose of generating I/O interruptions. This assignment allows a system to be configured to permit fast response to I/O interruptions from devices requiring high-priority service, relative to other attached devices.

An I/O interruption occurs when the PCE dispatches a new priority level in response to an I/O interrupt request. The I/O interrupt request is indicated to the program by means of the I/O interrupt request vector (IOIRV). The IOIRV is 8 bits long. The bit positions of the IOIRV correspond, left to right, with priority levels 0-7. The IOIRV is described in detail in Chapter 9, "PCE Control."

Priority Level Assignment

I/O devices are assigned to priority levels for the purpose of presenting I/O interrupt requests. Each device is assigned to a given priority level, and more than one device may be assigned to the same level. The assignment may be fixed at

manufacturing time or installation time, or it may be set under program control. Once the assignment is made, the device presents its interrupt requests to the assigned level.

The capability to assign an I/O device to a priority level under program control depends on the particular device and the processor model. For details concerning priority level assignment of I/O devices under program control, refer to the description manual for the specific processor model.

When two or more devices are attached to a single adapter, the priority level assignment applies to the adapter and all attached devices.

Input/Output Interrupt Requests

Conditions that initiate I/O interrupt requests are asynchronous to PCE activity, and more than one request can occur at the same time. However, only one interrupt request at a time is acted upon by the PCE. I/O interrupt requests are preserved (held pending) in the I/O device until recognized by the PCE.

The PCE usually recognizes I/O interrupt requests after executing every non-interruptible instruction. Depending on processor model, the PCE may delay this recognition for up to three instructions if it encounters an instruction string having short execution times.

Conceptually, I/O interrupt requests are recognized after each unit of operation for interruptible instructions. Depending on the processor model, a delay of up to eight units of operation prior to interrupt request recognition may occur.

An interrupt request generated by an I/O device is indicated in the BSTAT associated with the device. If the device is disabled (the enabled bit in the BSTAT is 0), the interrupt request is held pending at the device. When the device is enabled (the enabled bit is a 1), the interrupt request is also indicated in the IOIRV. When the interrupt request is associated with an enabled priority level higher in priority (lower in number) than the current priority level, an I/O interruption occurs, and the level to which the device is assigned is dispatched. Otherwise, the I/O interrupt request is held pending at the PCE until the associated priority level can be dispatched. Refer to Chapter 9, "PCE Control" for a detailed description of the PCE's priority level dispatching mechanism.

The I/O interrupt request continues to be indicated in the IOIRV so long as the device is enabled and the interrupt-request bit in the BSTAT is 1. Disabling the device or resetting the interrupt-request bit in the BSTAT (making the bit 0) removes the I/O interrupt request, provided no other device associated with the same priority level is presenting an interrupt request. When all I/O interrupt requests for the same priority level are removed, the indication in the IOIRV is removed.

Depending on processor model, an interrupt request indicated in the error interrupt request vector (EIRV) causes an interrupt request for level 0 to be also indicated in the IOIRV; however, an I/O interrupt request for level 0 may not be present. Therefore the program handling level 0 interruptions should inspect the EIRV before inspecting the IOIRV. This order of inspection will ensure that the source of an interruption for level 0 is not misinterpreted.

Programming Notes:

- I/O instructions can alter the interrupt request status by removing the current priority level interrupt request from the IOIRV or creating a higher priority interrupt request by issuing an appropriate adapter command. The effect of these instructions depends on the response time of the adapter. If the adapter responds within the I/O instruction, normally one more instruction in the current priority level is executed before the level change occurs. However, some model/adapter configurations may result in a level change immediately after the I/O instruction that alters the adapter interrupt request status. Because of the effects of intervening interrupt requests, even with slow adapters, the level change may occur before any more instructions (after the I/O instruction) in the current priority level are executed. In general, the level change due to an I/O instruction can occur immediately or after an adapter dependent number of instructions are executed.

- When an I/O interruption occurs, the interruption handling routine should (1) generate a programmed interrupt request for the current priority level, (2) read the device's status information, and then (3) remove the I/O interrupt request. Generating the programmed interrupt request allows program execution to continue after the I/O interrupt request is removed. The programmed interrupt request can be generated using the instruction SET PROGRAMMED INTERRUPT REQUEST. Removing the I/O interrupt request allows a subsequent I/O interrupt request from the associated adapter to be properly indicated. Refer to the description of the interrupt-request bit under "Basic Status Register," and the related PIO command, Reset BSTAT Under Mask, earlier in this chapter.

Multiple Interruptions for the Same Priority Level

When two or more I/O devices are assigned to the same priority level, and an I/O interruption to that level occurs, the program must determine which device is presenting the interrupt request. If more than one device is presenting an interrupt request, the priority of service to the interrupting devices is determined by the program. Facilities that assist the program in distinguishing among multiple I/O interrupt requests to the same priority level are described in the description manual for the applicable processor model.

Resetting I/O Devices

Two types of resetting can occur in the I/O system: an I/O system reset and an I/O selective reset. Both types disable an I/O device by causing the enabled bit in its BSTAT to be made 0. The specific response of each I/O device to the two kinds of reset depends on the type of device.

I/O System Reset

I/O system reset causes a reset of all attached I/O adapters and devices. This occurs when I/O reset is performed. Specific details about I/O reset are given in the description manual for the particular processor model.

I/O Selective Reset

The I/O selective reset is performed when the program executes the PIO command Reset Device. Only the devices attached to the adapter associated with the PIO address are reset. No other devices are affected.

The Reset Device command causes a reset operation to be executed at the selected I/O device. The reset state of the device depends on the particular device. For all devices, however, this command causes the enabled bit in the BSTAT associated with the device to be made 0.

This command disables the selected device. That is, when execution of the Reset Device command is completed, the device is prevented from generating an I/O interrupt request or executing a channel I/O operation. (See also the description of the enabled bit under "Basic Status Register" in this chapter.)

The operation of the I/O instruction issuing the Reset Device command includes the output of data. Whether the data is meaningful to the selected device depends on the particular device.

Channel Input/Output

The transfer of information between main storage and an I/O device is accomplished by means of channel input/output (CHIO) operations. The term *channel input/output operation* is used to denote the activity initiated at the I/O device by a PIO command that specifies a start-CHIO type operation (described later under "Channel Input/Output Operation").

A CHIO operation consists of the transfer of one or more bursts of information. The burst can consist of the transfer of a few bytes of data, a whole block of data, address information, status information, or control information used for the initiation of a new CHIO operation. The number of bytes transferred during a burst is referred to as the burst length. The burst length depends on the device and the type of operation. The length may be fixed or it may be set under program control. During the burst transfer, the I/O device monopolizes the channel and stays logically connected to it. No other device can communicate with the channel during the time a burst is transferred.

The facilities in the channel may be shared by a number of concurrently operating I/O devices. This sharing is achieved when the CHIO operations are split into short intervals of time during which a burst 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 requests from the devices. The channel controls are occupied with any one operation only for the time required to transfer a burst of information.

The system uses the facilities of the PCE for controlling CHIO operations. The sharing of common facilities between the channel and PCE causes PCE activity to be suspended during the transfer of a burst of information. This is accomplished automatically, however, and the program is not affected by the suspension of PCE activity, except for an increase in execution time.

Channel Input/Output Operation

A CHIO operation is controlled by a channel control vector (CHCV), a channel pointer (CHP), and a data count. The CHCV is the formatted control information sent by the device to the channel at the beginning of each CHIO burst transfer. The content of the CHCV is not made available to the program. Whether the format of the CHCV is apparent to the program depends on the specific device. The CHCV may be provided directly to the device by the

program, in which case the format is apparent. Alternatively, the device may generate the CHCV from other control information supplied by the program, and the program is not normally concerned with the CHCV format.

The CHCV specifies a CHP number and a CHIO command. The CHP provides the logical address of the main-storage area to be used for the burst transfer. The CHIO command is executed by the channel and specifies such operations as the reading or writing of data. The data count represents the amount of data transferred during a CHIO operation, and is maintained by the I/O device.

Functions peculiar to a device, such as rewinding a magnetic tape or positioning the access mechanism on a disk drive, are specified by means of device-specific protocols. That is, the format and meaning of control information specifying such functions, and the method used to supply the device with the information, depend on the particular device and operation. For example, device-dependent control information may appear in the programmed I/O (PIO) command code, or it may be transferred to the device as data during a CHIO or PIO operation.

Figure 8-2 illustrates the use of the CHCV, CHP, and data count during a CHIO data transfer operation. The CHIO command, which is specified within the CHCV, designates the type of operation. Refer to "Channel Control Vector" later in this chapter for a description of the CHCV format.

The program initiates a CHIO operation by issuing a PIO command to the I/O device. The device recognizes the command as a start-CHIO operation. The term "start CHIO" is the generic name for a class of device-specific commands. It is used in this publication to denote any PIO command that causes the device to initiate and execute a CHIO operation. The PIO command code designated as the start-CHIO command for a specific device is described in the SL publication for that device.

The start-CHIO command may be an immediate-type command, for which accompanying data is ignored. Conversely, the command may specify a write operation, for which the data provides control information, such as the CHIO command, CHP number, or data count, that the device uses in the execution of the CHIO operation. Control information may also be supplied to the device by means of one or more PIO instructions executed before the start-CHIO command is issued. Alternatively, the CHIO operation may consist of the writing of control information to the device followed by the reading or writing of data. In this latter case, the distinction between control information and data is made at the I/O device; the channel treats the entire operation as a data transfer.

The address of the storage area to be used for the data transfer is contained in the CHP specified in the CHCV. The address in the CHP designates the location in main storage from or into which the channel transfers the first byte of data. The address is placed in the CHP either before the program issues the start-CHIO command, or as part of the CHIO operation.

The CHP is associated with an address control vector (ACV). All main storage addresses used by the channel are treated as logical addresses. The ACV defines a logical address space, which consists of the set of consecutive logical addresses from 0 to the maximum address specified by the ACV size field. During each storage reference, the logical address is relocated by means of the dynamic address relocation facility. When dynamic address translation is specified in the ACV, the relocated address is then translated to a real address by means of the

dynamic address translation facility. On processor models that use separation protection, the translation lock is also compared with the protection key prior to allowing access to the associated translation-table entry and performing dynamic address translation. When address translation is not designated, the relocated address is used as the real address. Dynamic address relocation and translation, translation locks, protection keys, and the ACV are described in Chapter 7.

When the CHIO operation is initiated, the device is set up to issue service requests to the channel, and the channel and device assume subsequent control of the operation. The moment at which the first CHIO burst transfer begins may occur at the completion of the PIO instruction issuing the start-CHIO command, or later.

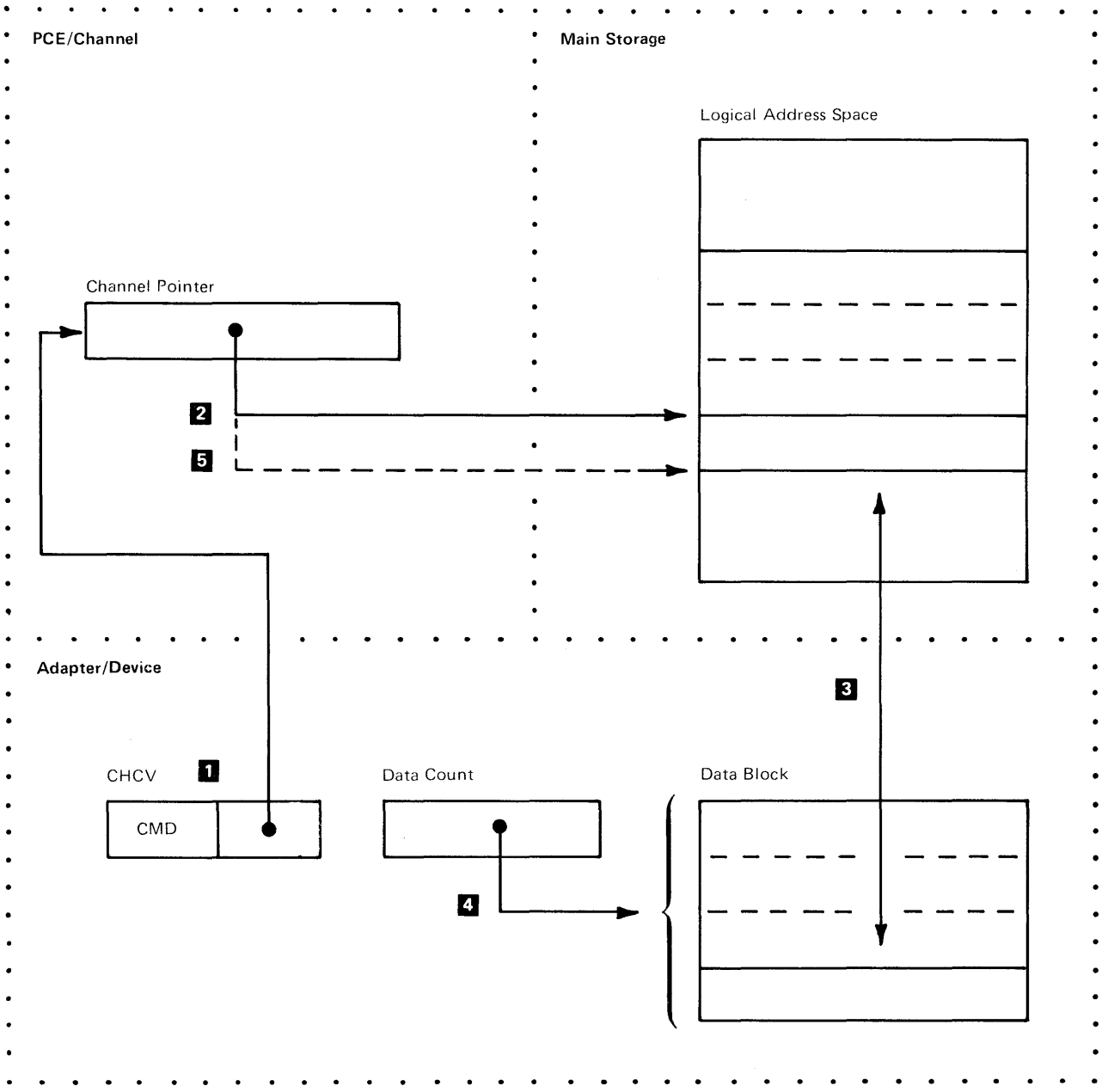
The channel executes the CHIO operation in response to service requests from the I/O device. The device requests service of the channel whenever it is ready to send or receive a burst of information. When the channel grants service to the device, the device becomes logically connected to the channel and responds by transferring the CHCV to the channel. The channel decodes the CHCV, including the command code, fetches the storage address from the CHP designated in the CHCV, and initiates the reading or writing of the burst of information. The device maintains the data count of the burst, while the channel maintains and updates the storage address as information is transferred to or from main storage.

The conclusion of a burst transfer is signaled by the device. It causes the channel to logically disconnect the device, and place the updated storage address back in the CHP. The updated address in the CHP is thus ready for use for the next burst transfer, if any.

The CHIO operation consists of one or more burst transfers. The number of bursts and the amount of information transferred during each burst depend on the device and the type of operation. The CHIO operation is concluded when the last burst transfer of the operation has been completed.

Depending on the particular device, the conclusion of a CHIO operation may be brought to the attention of the program with an I/O interruption. When the end of the operation is not signaled with an I/O interruption, or when the priority level is disabled for the I/O interruption, the conclusion may be determined by programmed interrogation of the I/O device. In either case, the device status, which contains information concerning the execution of the CHIO operation, is available to the program by execution of one or more PIO operations, such as the PIO command Read BSTAT.

Programming Note: A malfunction that affects the validity of data transferred in a CHIO operation generates a system-check interruption when the channel detects the error, and causes the operation to be terminated. A malfunction detected by the I/O device either results in termination of the CHIO operation and a system-check interruption, or is indicated after the operation is concluded, depending on the type of error. Data read during an input operation should not be used until the end of the CHIO operation has been reached and the validity of the operation has been checked. Similarly, on writing, the copy of data in main storage should not be destroyed until the program verifies that no malfunction affecting the transfer and recording of data was detected.



Notes:

1. CHCV supplied by device specifies channel pointer number and CHIO command (CMD).
2. Channel pointer designates starting location for burst transfer.
3. Data is transferred (in this case, the fourth burst of the data block).
4. Device maintains count of data transferred.
5. At completion of burst transfer, channel pointer designates starting location for next burst transfer, if any.

Figure 8-2. Channel I/O Data Transfer Operation

Execution of Channel Input/Output Operations

The execution of a CHIO operation is accomplished in one or more burst transfers. The channel executes a burst transfer at the request of the device. Each burst transfer is treated is an independent operation by the channel, which allows the channel to execute two or more concurrent CHIO operations by interleaving their burst transfers. A burst may include the transfer of data, the data address, or both. The type of information transferred and the direction of the transfer (input or output) is specified by the CHIO command. The CHIO commands are described in detail later in this chapter under "Channel Input/Output Commands."

The CHIO command is contained in the CHCV, which is transmitted by the device to the channel at the beginning of each burst. The burst transfer begins when the channel accepts the CHCV from the I/O device, and lasts until end-of-burst is signaled by the device.

Blocking of Data

Data recorded by an I/O device may be divided into physical 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.

A block of information may be transferred in one burst. Normally, however, a block is transferred in a number of bursts, each consisting of only a few bytes. One or more blocks may be transferred in one CHIO operation. The capability to transfer multiple blocks in one CHIO operation, and the manner in which it is accomplished, depends on the particular device.

For some operations, such as writing on magnetic tape, blocks are not defined, and the amount of information transferred is specified by the program. The data count, in this case, is provided to the device as control information before the data transfer. The count may be supplied to the device as part of the CHIO operation or by means of a PIO operation.

Channel Pointer

The channel pointer (CHP) is 32 bits long and is used by the channel during CHIO operations to address main storage. The CHP contains the logical address of a byte location in main storage. This logical address designates the first location referred to in the execution of a burst transfer of data between main storage and an I/O device. The address may be placed in the CHP either (1) by the program by means of register-indirect instructions (described in Chapter 6, "Register Organization"), or (2) by the I/O device as part of the burst transfer. Alternatively, a burst transfer may use the address that the channel stored back into the CHP at the end of the previous burst transfer. This is typically the case for the second and subsequent burst transfers of a CHIO operation in which the data is transferred in multiple bursts. During the burst, storage locations are referred to in ascending order of logical addresses. When the burst transfer is concluded, the channel increases the address in the CHP by an amount equal to the number of bytes transferred to or from main storage.

The system provides 64 CHPs for CHIO operations. Each CHP number is permanently assigned to one of the principal registers from sets 8-15 as follows:

CHP	Register Set
0-7	12
8-15	13
16-23	14
24-31	15
32-39	8
40-47	9
48-55	10
56-63	11

The eight registers within each set are assigned to the corresponding CHPs in ascending order. That is, CHP numbers 0-7 are respectively assigned to register numbers 0-7 in set number 12, CHP numbers 8-15 to register numbers 0-7 in set 13, and so on to CHP numbers 56-63, which are respectively assigned to register numbers 0-7 in set 11. Figure 6-4 illustrates the allocation of principal registers to channel pointers.

The CHP-number field in the CHCV designates 1 of the 64 CHPs. When a burst transfer to or from main storage begins, the channel fetches the storage address from the principal register assigned as the channel pointer designated in the CHCV.

For burst transfers that include transferring the data address from the I/O device to the CHP, the channel stores the address into the principal register assigned as the CHP specified in the CHCV. Burst transfers that write the data address from the CHP to the I/O device cause the channel to fetch the address from the principal register assigned as the CHP specified in the CHCV. Only the principal register assigned as the specified CHP is referred to for these operations.

Programming Note: The CHIO structure allows a single CHP to be used for the execution of two or more concurrent CHIO operations. However, the capability to share a CHP among concurrently executing CHIO operations should be used only when the 32-bit data address is read from the device as part of every burst transfer. This normally requires that the I/O device maintain and update the storage address during the CHIO operation, thereby ensuring that each burst transfer begins at the proper storage location.

Designation of Storage Area

The main-storage location at which a burst transfer begins is identified by the CHP designated in the CHCV, and the ACV associated with the CHP. The ACV defines the logical address space available to the channel for the transfer of a burst of data to or from main storage. The CHP contains the logical address of the first byte of data to be transferred. The extent of main storage referred to during the burst transfer is determined by the device. That is, the device controls the number of bytes transferred, and concludes the transfer after the last byte of the burst has been read or written.

Consecutive storage locations are used in ascending order of logical addresses. As information is transferred to or from main storage, the logical address from the CHP is incremented. At the conclusion of the burst transfer, the logical address in the CHP is increased by the number of bytes transferred. Thus the amount of

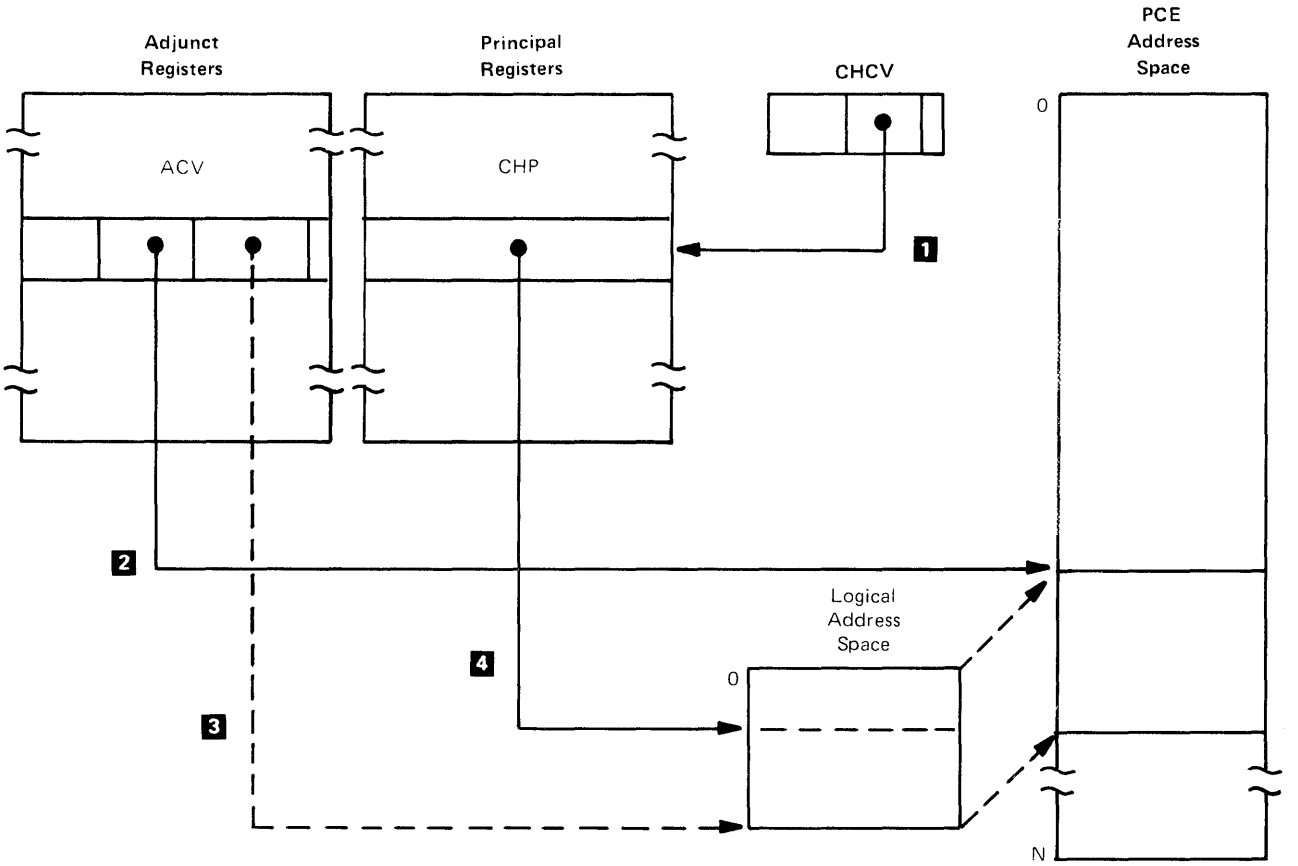
storage used during the burst transfer is reflected in the difference between the address in the CHP at the beginning of the transfer and the address stored back in the CHP at the conclusion of the transfer.

Any main-storage location available to the channel can be used in the transfer of data to or from an I/O device, provided the location is not protected against the type of reference. Protection for CHIO operations is provided when dynamic address translation is active for the operation. Protection is specified by the block-invalid and channel-store-protection bits of the access-control field in the translation-table entries. Also, depending on processor model, separation protection is specified by the translation lock and its associated translation-lock-table entry in combination with the protection key associated with the ACV. For block-invalid and channel-store protection, an access exception is detected when the channel attempts to refer to a location protected against the type of reference. For separation protection, a separation exception is detected when the channel attempts to refer to a location without a valid translation lock. In either case, the transfer is terminated and a system-check interruption is generated with channel I/O check and exception indicated.

A main-storage location is available to the channel when (1) the physical storage location is installed, (2) the corresponding logical address is within the logical address space defined for the burst transfer, and (3) the location is not protected against the type of reference. The ACV associated with the specified CHP defines the logical address space. That is, it defines the set of consecutive logical addresses, ranging from 0 to a maximum address, considered valid for the burst transfer. The maximum valid address in the logical address space is 1 less than the size of the space, which is designated by the size field in the ACV. The relationship of the CHCV, CHP, and ACV to the designation of the logical storage area is illustrated in Figure 8-3.

When the channel attempts to refer to a physical storage location not installed, a specification exception is detected. When it attempts to refer to a storage location that is access-protected or separation-protected, an access or separation exception is detected. When it attempts to refer to a location specified by a logical address outside the logical address space, an address exception is detected. In either case, the burst transfer is terminated and a system-check interruption is generated with channel I/O check and exception indicated. The CHP contains the logical address of the first byte of data referred to, if any, for the terminated burst transfer.

During an output operation, the channel may fetch data from main storage before the time the I/O device requests the data. Any number of bytes may be prefetched and buffered. A specification, access, separation, or address exception detected during prefetching of data does not affect the execution of the operation and does not cause a system-check interruption until the I/O device actually requests the data. If the burst transfer is concluded before the data is requested, the exception is not brought to the attention of the program.



Notes:

1. The CHCV designates the CHP and its corresponding ACV.
2. The origin field in the ACV defines the origin of the logical address space in the PCE address space. The origin address in the PCE address space corresponds to logical address 0.
3. The size field in the ACV defines the size of the logical address space. The maximum logical address available to the channel is 1 less than the size of the logical address space.
4. The channel-pointer content designates the starting location of the data area in the logical address space.

Figure 8-3. Designation of Logical Storage Area

Channel Input/Output Commands

The CHIO command is part of the CHCV, and designates to the channel the type of burst transfer to be performed. The six CHIO commands executed by the channel are:

Write Data
Read Data
Write Data Address
Read Data Address
Read Data Address And Write Data
Read Data Address And Read Data

Each command is described in the following paragraphs under the individual command names. Of the commands listed above, those that are implemented by a device depends on the specific device.

Write Data: The channel writes a burst of data from main storage to the I/O device. Data in storage is fetched in an ascending order of logical addresses, starting with the address contained in the designated CHP. At the conclusion of the burst transfer, which is signaled by the device, the address in the CHP is increased by an amount equal to the number of bytes written to the device.

Read Data: The channel reads a burst of data from the I/O device into main storage. Data is placed in storage in an ascending order of logical addresses, starting with the address contained in the designated CHP. At the conclusion of the burst transfer, which is signaled by the device, the address in the CHP is increased by an amount equal to the number of bytes read from the device.

Write Data Address: The channel writes 1 to 4 bytes of address information from the designated CHP to the I/O device. The information is fetched from the CHP in left-to-right order, starting at the location specified by the modifier bits in the command code. (The modifier bits are defined in this chapter under "Command Code Modifier Bits.") The operation ends when the transfer is concluded by the device.

Read Data Address: The channel reads 1 to 4 bytes of address information from the I/O device into the designated CHP. The information is placed in the CHP in left-to-right order, starting at the location specified by the modifier bits in the command code. (The modifier bits are defined in this chapter under "Command Code Modifier Bits.") The operation ends when the transfer is concluded by the device.

Read Data Address And Write Data: The channel reads 1 to 4 bytes of address information from the I/O device into the specified CHP, as described for the Read Data Address command. After information is placed in the rightmost byte location of the CHP, the channel writes a burst of data from main storage to the I/O device, as described for the Write Data command.

Read Data Address And Read Data: The channel reads 1 to 4 bytes of address information from the I/O device into the specified CHP, as described for the Read Data Address command. After information is placed in the rightmost byte location of the CHP, the channel reads a burst of data from the I/O device into main storage, as described for the Read Data command.

Channel Pointer Usage

Transfer of Data: Execution of the commands Write Data and Read Data consists of transferring a burst of data between main storage and the I/O device. The contents of the specified CHP are used as the logical address of the storage location at which the transfer begins. The number of consecutive storage locations used during the transfer is determined by the length of the burst. The burst length represents the number of bytes transferred and is controlled by the device.

The burst ends when the device signals the conclusion of the transfer, at which time the channel increases the address in the CHP by the length of the burst. If an error is detected by the channel during the transfer of the burst of data, the channel terminates the transfer and generates a system-check interruption. In this case, the address in the CHP is not increased and remains unchanged.

Transfer of Data Address: Execution of the commands Write Data Address and Read Data Address consists only of transferring address information to or from the specified CHP; no reference is made to main storage. The maximum number

of address bytes transferred depends on the starting location in the CHP, as designated by the modifier bits. That is, a maximum of 4 bytes may be transferred when the modifier bits designate the leftmost byte of the CHP, whereas only 1 byte is transferred when the rightmost byte is designated. The number of bytes transferred may be less than the maximum, depending on the device. For example, the burst may be concluded after 2 bytes are transferred to or from the two high-order byte locations of the CHP. When fewer than 4 bytes are read for the Read Data Address command, the remaining bytes in the CHP are not changed.

The conclusion of the burst transfer is signaled by the device, and it may occur before, or concurrently with, the transfer of address information to or from the rightmost byte of the CHP. An attempt to transfer information beyond the rightmost byte of the CHP causes a system-check interruption with I/O timeout check and channel I/O check indicated.

When the channel detects an error during the transfer of the data address to or from the CHP, the transfer is terminated and a system-check interruption is generated. The extent to which the contents of the CHP are changed during execution of the Read Data Address command depends on the point at which the error occurs.

Transfer of Data Address and Data: Execution of the command Read Data Address And Write Data and the command Read Data Address And Read Data consists of reading address information into the specified CHP and then transferring data to or from main storage. The channel reads 1 to 4 address bytes, depending on the modifier bits, and places the information in the CHP. After the low-order address byte is placed in the rightmost byte location of the CHP, the channel proceeds to transfer data between main storage and the I/O device in the manner described earlier for the commands Write Data and Read Data. All 32 bits of the CHP contents are used as the data address, even when fewer than 4 bytes of address information are read from the I/O device.

If the device signals the conclusion of the burst transfer before any data is transferred, no reference to main storage occurs. If the channel detects an error during the burst transfer, the transfer is terminated and a system-check interruption is generated. If the error is detected during the transfer of the data address, the extent to which the contents of the CHP are changed depends on when the error occurs. If the channel detects the error during data transfer to or from main storage, the contents of the CHP remain as modified during the reading of the address information.

Conclusion of Channel Input/Output Operations

When a CHIO operation is ended, the conclusion may be signaled by the device with an I/O interruption, or it may be determined by programmed interrogation, depending on the particular device and operation. In either case, the device generates status information that indicates conditions pertaining to the execution of the CHIO operation. This device-status information is stored in the BSTAT associated with the device. Additional device-status information, if any, is stored in device-specific status registers.

At the conclusion of a CHIO operation, the device-status information may be obtained by the program by executing one or more programmed I/O commands that read the status information. Alternatively, the device may transfer its status information into main storage as part of the CHIO operation. The BSTAT and

the associated PIO commands are described under "Basic Status Register" in this chapter. The status information provided by the device and the manner in which the information is made available to the program depend on the particular device.

Types of Conclusion

Normally, a CHIO operation lasts until the device concludes the final burst transfer. When the channel recognizes a channel exception or detects equipment malfunctioning, the channel terminates the burst transfer immediately by signaling halt to the device and logically disconnecting the device. The device is also disconnected prematurely when system reset or I/O reset is performed, or when the program issues the PIO command Reset Device.

Normal Conclusion of Data Transfer: When a CHIO operation is initiated, the I/O device is set up for data transfer. The duration of data transfer operations may be variable and is controlled by the device. Unless an error is detected, or the operation is prematurely concluded by the Reset Device command, the CHIO operation lasts until it is concluded by the device. The device-status information indicates any unusual conditions encountered during the operation and may, depending on the device and type of operation, indicate the normal ending of the CHIO operation. For devices that do not provide a normal-ending indication, the program may assume normal conclusion of the CHIO operation when no errors are indicated.

Termination Due to Channel Exception: When the channel detects a channel exception, the CHIO operation is terminated and a system-check interruption occurs with exception and channel I/O check indicated in the EIRV. The channel detects the following exceptions: operation, specification, address, separation, or access. These channel exceptions are described below.

- Bits 0-4 of the CHCV are not all 0's (operation exception for invalid operation).

- The format of the ACV associated with the burst operation is invalid (specification exception for ACV format). An ACV format is invalid (1) when bits 0-7 of the ACV are not all 0's, (2) when the size value is not defined or not available on the PCE, or (3) when the origin value exceeds the maximum address in the PCE address space.

- The channel attempts to refer to a storage location not available to the channel. A storage location is not available (1) when the logical address is greater than or equal to the value specified by the size field in the ACV associated with the burst transfer (address exception for address limit), or (2) when the physical storage location corresponding to the logical address is not installed (specification exception for real address).

- For processor models that implement separation protection, the channel attempts to refer to a storage location protected by the translation lock/protection key mechanism. The referenced storage location is logically separated from the channel I/O operation and access is not allowed when dynamic address translation is active (ACV bit 31 set to 1) and a translation lock/protection key mismatch occurs. This mismatch occurs if the value of the translation lock corresponding to the referenced storage block and the value of the protection key associated with the active ACV are not identical and neither value is zero.

- The channel attempts to refer to a storage location not accessible to the channel by access control. A storage location is not accessible when dynamic address translation is active (bit 31 of the ACV is 1) and the location is protected against the type of reference. Protection against the type of reference is specified by the access-control field in the translation-table entry corresponding to the logical address. The channel exception is detected if either (1) the channel attempts a reference of any type to the storage location and the block-invalid bit for the location is 1 (access exception for block invalid), or (2) the channel attempts to store into the location and the channel-store-protection bit is a 1 (access exception for channel store protection). Store-type references apply to the CHIO command Read Data and to the data transfer portion of the CHIO command Read Data Address And Read Data.

A channel exception causes the channel to conclude the data transfer. The conclusion is signaled to the device at the time the exception is detected. The channel sends a halt signal to the device causing the data transfer to cease. The channel then logically disconnects the device and causes a system-check interruption with exception and channel I/O check indicated in the EIRV.

When the channel signals halt to the device, the device acknowledges receipt of the halt signal by setting the equipment-check and interrupt-request bits to 1's in its BSTAT. Other device-status bits may also be set to 1's, depending on the device and type of operation. Setting the interrupt-request bit to 1 generates an I/O interruption for the priority level to which the device is assigned. The I/O interruption is indicated to the program by means of the IOIRV.

The channel exception may be recognized at the beginning of a CHIO operation or after data transfer has been initiated. When the exception is detected at the beginning of the CHIO operation, no data is transferred during the operation, and the device is signaled to terminate the operation without any reference to main storage. Whether a block of data is advanced at the device when no data is transferred depends on the particular device.

Programming Note: See the programming notes under the heading "Termination Due to Equipment Malfunction" which follows.

Termination Due to Equipment Malfunction: When equipment malfunctioning related to a CHIO operation is detected, or when invalid parity or control signals are received from the I/O device, the channel terminates the operation. The recovery procedure and the subsequent state of the device depend on the type of error and the point in the operation at which the error occurs. The program is alerted to the termination by a system-check interruption, and the EIRV indicates the type of error encountered. Except for the difference in the type of error indicated in the EIRV, the action taken by the channel is the same as described under "Termination Due to Channel Exception."

The channel causes bit position 4 of the EIRV to be set to 1, indicating a channel I/O check. One or two other bits in the EIRV are also set to 1's, indicating the

specific type of system check. The following system checks related to a channel I/O operation may be indicated in the EIRV:

EIRV Bit(s)	System Check
0	I/O Control Check
1	I/O Timeout Check
2	Storage Data Check
3	Exception
4	Channel I/O Check (always indicated for CHIO errors)
5	Internal Control Check
2 and 5	Internal Data Check

These system checks are described in detail under "System Checks" in Chapter 9.

Programming Notes:

- An interruption for a system check related to a CHIO operation occurs after the PCE completes execution of the current instruction, or the current unit of operation for interruptible instructions. The instruction address in the stored PSV for the interrupted program and the instruction address modifier in the EIRV designate the instruction that would have been executed next had the interruption not occurred. The instruction address modifier indicates whether the instruction address stored with the PSV for the interrupted program designates the location of the next instruction to be executed, or has been incremented by 2 to designate the location two bytes beyond the next instruction. The instruction address modifier bit is set to 1 if the instruction address has not been incremented by 2; otherwise it is not changed. If the address has been incremented, 2 must be subtracted from the stored instruction address before control is returned to the interrupted program.

- The detection of a system check related to a CHIO operation results in an I/O interruption as well as a system-check interruption. When the I/O device is assigned to a priority level other than 0, the system-check interruption is always handled first. This is performed unless bit 0 of the common mask is reset to 0 on processor models that can disable priority level 0 for system-check interrupt requests. Priority level 0, therefore, should not be disabled so that the channel exception or malfunction can be handled properly. When the device is assigned to priority level 0, the program handling the interruption should inspect the EIRV before inspecting the IOIRV.

Enabling and Disabling of Channel Input/Output Operations

The channel can be enabled or disabled for CHIO operations. When the channel is enabled, burst transfers can take place. When the channel is disabled, burst transfers are disallowed and the requests by the devices for initiating burst transfers are ignored. Depending on the particular device, a request to initiate a burst transfer may remain pending until the channel is enabled, or the device may remove the request before the channel is enabled.

The enabling and disabling of CHIO operations is controlled by the channel mask and the EIRV. Burst transfers may take place only when the channel is enabled by both the channel mask and the EIRV. When the channel is disabled by either one, burst transfers are disallowed.

The channel can become disabled during a CHIO operation. If the channel becomes disabled because of a system check, the CHIO operation causing the system check is terminated. When the system check is not related to the CHIO operation, or when the program disables the channel, any remaining burst transfers for the CHIO operation are held pending. The state of a CHIO operation and the associated device at the time the channel is subsequently enabled and the ability of the device to continue the CHIO operation depend on the type of operation, the particular device, and the amount of time that the operation is held pending.

Programming Note: Disabling the channel for CHIO operations does not affect execution of PIO operations by the channel.

Channel Mask

The channel mask is a 1-bit mask. When the mask is 1, the channel is enabled for CHIO operations, provided the channel is not disabled by the EIRV. When the mask is 0, the channel is disabled. The channel mask is altered under program control.

Three PCE-control instructions are provided to inspect or change the channel mask. READ CHANNEL MASK allows the program to determine the current value of the mask. SET CHANNEL MASK and RESET CHANNEL MASK provide for making the mask 1 or 0, respectively.

Error Interrupt Request Vector

The channel is disabled by the EIRV when a 1 is in any of the bit positions 0-3 or 5 of the EIRV. When these bit positions contain all 0's, and the channel mask is 1, the channel is enabled.

If execution of the PCE-control instruction WRITE ERROR INTERRUPT REQUEST VECTOR places a 1 in any of the bit positions 0-3 or 5, the channel is disabled at the completion of the instruction. When 0's are written into these bit positions, and the channel mask is 1, the channel is enabled at the completion of the instruction.

When a system check is detected by the PCE or channel, the system check is indicated with bit positions 0-5 of the EIRV. The channel becomes disabled when bits 0-3 or 5 are set to 1, and remains disabled as long as bits 0-3 or 5 of the EIRV are not all 0's. The system-check interruption is not taken at the time the system check is detected when priority level 0 is active or, for processor models that can disable priority level 0 for system-check interrupt requests, when bit 0 of the common mask is reset to 0.

Bit 4 of the EIRV is set with one or more other bits of the EIRV if the system check results from a channel I/O operation. Bit 4 alone being set to 1 does not disable the channel. See Chapter 10 for bit 4 operation in dual-PCE processors.

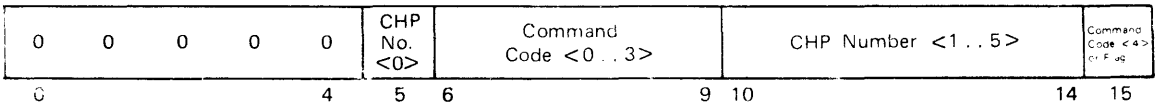
Channel Control Vector

The channel control vector (CHCV) is the formatted control information sent by the I/O device to the channel at the beginning of each CHIO burst transfer. The CHCV designates for the burst transfer the CHIO command to be executed and the channel pointer (CHP) to be used. For commands that transfer data to or from main storage, the CHP provides the logical address of the storage area to be used for the burst transfer.

When a CHIO operation consists of two or more bursts, the same copy of the CHCV may be used for each burst. Conversely, the CHIO operation may use several different CHCVs. One CHIO operation may include, for example, the transfer of the storage address to the channel pointer, control information to the I/O device, and data to main storage. At least two different CHCVs are used in this example.

Channel Control Vector Format

The CHCV is 16 bits long. Whether the format of the CHCV is made apparent to the program depends on the specific device. The CHCV may be provided to the device by the program, in which case the format of the CHCV is apparent to the program. Alternatively, the I/O device may generate the CHCV from control information it receives from the program, and the program is not normally concerned with the CHCV format. The following figure illustrates the CHCV format.



The functions of the CHCV fields are as follows:

- ***CHP Number:*** Bits 5 and 10-14 form the binary number of the channel pointer to be used for the burst transfer. Bit 5 is concatenated to the left of bits 10-14 in forming the 6-bit number. The CHP Number designates 1 of 64 CHPs available for CHIO operations.

- ***Command Code:*** The CHIO command code specifies the type of burst transfer to be performed. Bits 6-9 represent the high-order 4 bits of the 5-bit command code. Bit 15 represents the low-order bit of the command code when the operation involves a byte-mode I/O device. When the operation involves a halfword-mode I/O device, bit 15 is not used for the command code; instead, the channel assumes a 0 for the low-order bit of the code.

- ***Flag:*** When the burst transfer involves a halfword-mode I/O device, bit 15 represents the CHCV flag. The flag bit, when a 1, causes the channel to substitute 0's for the eight high-order bit positions of the CHCV in place of the corresponding bits presented by the device; only the eight low-order bits of the CHCV are inspected for data check. When the flag bit is 0, the channel uses all 16 bits supplied by the device as the CHCV.

- ***Reserved Bits:*** Bit positions 0-4 are reserved and must contain 0's. Otherwise, a system-check interruption occurs with channel I/O check and exception indicated.

Programming Notes:

- When a burst transfer is initiated by a byte-mode device, the device may transfer all 16 bits of the CHCV to the channel or only the low-order 8 bits. In the latter case, the channel assumes 0's for the high-order 8 bits. Logically, a halfword-mode device may also transfer only the low-order 8 bits of the CHCV by setting bit 15 (the CHCV flag) to 1.

- When 0's are assumed by the channel for the high-order 8 bits of the CHCV: (1) the range of CHP numbers that can be specified is limited to 0-31, (2) the CHIO commands that can be designated are limited to Write Data, Read Data, Write Data Address, and Read Data Address, and (3) only the low-order two bytes of the CHP can be accessed for Write Data Address and Read Data Address.

Command Code

The 5-bit command code specifies to the channel the type of burst transfer to be performed. Bits 0, 2, and 3 of the command code (bits 6, 8, and 9 of the CHCV) identify to the channel the type of burst transfer. Bits 1 and 4 of the code are treated as modifiers. Each command was described earlier under "Channel Input/Output Commands" in this chapter.

The command-code assignment is listed in the following table; the symbol m identifies a modifier bit.

Code Bits 0 1 2 3 4	Command
0 0 0 0 0	Write Data
0 0 1 0 0	Read Data
0 m 0 1 m	Write Data Address
0 m 1 1 m	Read Data Address
1 m 0 0 m	Read Data Address And Write Data
1 m 1 0 m	Read Data Address And Read Data

Command codes not listed in the preceding table are reserved. If the channel encounters a reserved command code, the result is unpredictable. Depending on processor model, a system-check interruption may occur.

Command Code Modifier Bits

The modifier bits pertain to burst transfers that include reading or writing the data address. From 1 to 4 bytes of address information are transferred to or from the specified channel pointer, beginning at the location designated by the modifier bits and proceeding to the right. The two modifier bits specify the beginning location in the channel pointer as follows:

Command Code Bits 1 4	Beginning Location in CHP	
	Byte-Mode Device	Halfword-Mode Device
1 0	First Byte (bits 0-7)	First Halfword (bits 0-15)
1 1	Second Byte (bits 8-15)	--
0 0	Third Byte (bits 16-23)	Second Halfword (bits 16-31)
0 1	Fourth Byte (bits 24-31)	--

Programming Note: For halfword-mode devices, bit 4 of the command code is always assumed 0 by the channel, and only the combinations 10 and 00 of the modifier bits apply. In this case, the address information transferred between the channel pointer and a halfword-mode device consists of either 2 or 4 bytes.

Chapter 9. PCE Control

This chapter describes in detail the PCE facilities that provide for: switching the status of the PCE; protecting a program from interference by another program; and, in general, enhancing the efficiency, utility, and integrity of the IBM 8100 system.

The information defining the state of the PCE and controlling its operation resides in the program status vector (PSV), the address control vector (ACV), and in other vectors of PCE control information.

By providing a supervisor mode for program execution, and a class of instructions that are valid only in the supervisor mode for changing the contents of the PSV, control vectors, permanently assigned register locations, the translation table, and the translation lock table, a means is provided for avoiding unauthorized or inadvertent change to the state and operation of the PCE.

Further protection is furnished by the dynamic-address-relocation and dynamic-address-translation facilities. By allowing programs to execute in separate address spaces, dynamic address relocation protects main storage allocated to one program from destruction or misuse by another program. The dynamic-address-translation facility includes additional protection by means of translation locks, protection keys, and access controls. For programs executing in the same address space, translation locks and protection keys allow the programs to access blocks of common storage and, at the same time, provide private blocks of storage for each program. Access control provides protection against erroneous or unauthorized storing, instruction execution, or storage references of any type by the program or channel. The operations of the relocation and translation facilities are described in Chapter 7.

Special considerations for PCE control in dual-PCE processors are described in Chapter 10.

PCE States

Excluding facilities provided for maintaining and testing equipment and programs, three PCE states are defined: wait, running, and initial.

In the wait state, no instructions are fetched or executed; in the running state, instruction fetching and execution proceed in the normal manner. A description of the wait and running states is included under "Priority Level Dispatching" in this chapter.

The initial state is the name given to the state of the PCE as it appears after initial program load (IPL) and just before the first instruction is fetched from main storage. The initial state of the PCE depends on the processor model. Refer to the specific processor description manual for a definition of the PCE's initial state.

Programming Note: In the wait state, the PCE does not make repeated references to main storage; therefore, the wait state is suitable for delaying operation until an external event occurs. References to main storage may, however, be made for channel I/O operations.

Program Modes

Four distinct modes of execution are provided that determine which instructions may be used by a program. The four modes are hierarchical and are called *application*, *input/output*, *supervisor*, and *master*. *Application* is the least privileged mode and *master* is the most privileged. The four modes are distinguished by the portion of the full set of instructions that is valid for each mode, and, for master mode, the capability to override certain storage-access protection.

In application mode, the valid instructions are those that are used for normal information processing. These instructions cannot be used to execute input/output operations, nor can they be used to modify information that controls the PCE.

In input/output mode, the valid instructions are those that can be executed in application mode, plus three input/output instructions and two PCE-control instructions. These latter five instructions are called the *input/output-privileged instructions*. A program-exception interruption occurs, with operation exception indicated, when an input/output-privileged instruction is encountered in application mode.

In supervisor mode, all instructions are valid. Those instructions that can modify PCE-control information, that can indirectly refer to any principal or adjunct register, or that can access the translation table or the translation lock table, are called *supervisor-privileged instructions*. A program-exception interruption occurs, with operation exception indicated, when a supervisor-privileged instruction is encountered in either application or input/output mode.

In master mode, as in supervisor mode, all instructions are valid. In master mode, however, all references to main storage resulting from program execution are allowed, regardless of the state of the associated access-control bits, provided the block-invalid bit is 0. An access exception is recognized when dynamic address translation is active and an attempt is made in any program mode to refer to main storage when the associated block-invalid bit is 1.

The program mode is designated by a 2-bit program-mode field in the PSV.

Programming Note: Channel I/O operations are performed independently of a PSV. Therefore, the program mode does not affect either the validity of channel I/O operations or the storage references resulting from these operations.

Program Status Vector

The program status vector (PSV) is 64 bits long and contains the information required for proper program execution. In general, the PSV is used to control instruction sequencing and general-register assignments, and to hold and indicate the status of the PCE in relation to the program currently being executed. The PSV includes the instruction address, condition indicators, register set numbers, and other control fields. The active PSV is called the *current* PSV. The status of the program is preserved for subsequent use when the current PSV is stored.

The state of the PCE is changed when a new PSV is introduced or when information is changed in the current PSV. The current PSV is stored and a new PSV is introduced as part of the interruption action performed by the PCE. The

storing of the current PSV always precedes the introduction of a new PSV. PSV information is stored in, and introduced from, permanently assigned register locations.

A number of instructions are provided to introduce new control information into certain fields of the current PSV; the old information replaced by these instructions is lost. The instruction address is updated by sequential instruction execution and replaced by successful jumps and branches. Instructions are also provided for retrieving certain information from the current PSV. These instructions place the contents of the corresponding PSV fields in the designated general register. BRANCH AND LINK places the instruction address in the designated general register.

New PSV information becomes active (that is, the information introduced into the current PSV assumes control over the PCE) at the completion of the interruption action or instruction execution that introduced the new PSV information.

Program Status Vector Format

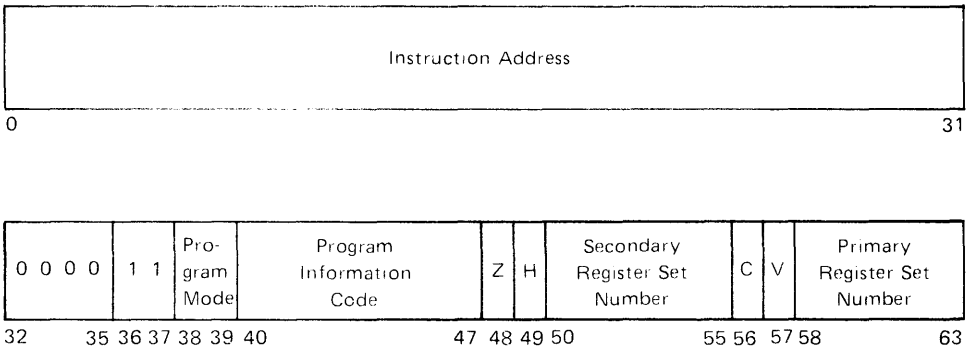


Figure 9-1. Program Status Vector Format

Figure 9-1 illustrates the PSV format.

The following is a summary of the functions of the PSV fields:

- ***Instruction Address:*** Bits 0-31 form the instruction address. This address is the logical address of the storage location containing the first halfword of the next instruction. For a detailed description on the use of bit 31 during address generation, see "Address Generation" in Chapter 3.
- ***Program Mode:*** Bits 38 and 39 identify the program mode and control which instructions may be used by the executing program. Bits 38 and 39 also control the recognition of certain access exceptions. The program mode is identified by the following values of these bits:

Bits 38-39	Program Mode
0 0	Master
0 1	Supervisor
1 1	Input/Output
1 0	Application

- ***Program Information Code:*** Bits 40-47 in the PSV stored on a program-exception interruption, or during the execution of CALL PSV, identify the cause of the switch in PSVs. When a new PSV is introduced, the contents of this field are ignored.

  When a program-exception interruption occurs, bit 40 is made a 1, bit 41 contains the instruction address modifier, and bits 42-47 contain the program-exception code. The low-order two bits of the program information code (PSV bits 46 and 47) are reserved and contain 0's. A description of the instruction address modifier and program-exception code is given in "Program Exception Conditions" in Chapter 3.

  When CALL PSV is executed, bits 40-47 in the stored PSV are set to 0. When an interruption occurs for any reason other than a program exception or execution of CALL PSV, bits 40-47 are reserved and their contents depend on processor model.

- ***Condition Indicators (Z, H, C, and V):*** Bits 48 and 49 are the two condition-indicator bits used to represent the states of result conditions 8, 4, and 2. Bits 56 and 57 are the two condition indicator bits that represent the states of result conditions 0 and 1, respectively. Each result condition has two possible states: *indicated* and *not-indicated.* The states are derived from bits 48, 49, 56, and 57 of the current PSV as follows (where x indicates that the bit is not significant in determining the state of the result condition):

PSV Bits 48	49	56	57	Result Condition Indicated
1	x	x	x	8
0	1	x	x	4
0	0	x	x	2
x	x	x	1	1
x	x	1	x	0

  Result conditions 8, 4, and 2 are derived from bits 48 and 49 such that only one of the three conditions is indicated at any one time (the other two are not-indicated); however, one of the three is always indicated. Result condition 8 is indicated whenever bit 48 is 1 and not-indicated when bit 48 is 0, without regard to bit 49. Result conditions 1 and 0 can be indicated together and with result conditions 8, or 4, or 2. Result condition 1 is indicated whenever bit 57 is 1, and result condition 0 is indicated whenever bit 56 is 1. Whenever bit 56 or 57 is 0, the corresponding result condition is not-indicated.

- **Secondary Register Set:** Bits 50-55 form the binary number of the secondary register set assigned to the program.

- **Primary Register Set:** Bits 58-63 form the binary number of the primary register set assigned to the program.

- **Reserved Bits:** Bit positions 32-37 are reserved. Bit positions 32-35 must contain 0's; otherwise, a specification exception is recognized. Bit positions 36 and 37 should contain 1's for normal system operation. Depending on processor model, certain functions that are unique to that model may be invoked when these two bit positions do not contain 1's.

When an interruption occurs, the fields of the current PSV that are stored are the instruction address, the primary and secondary register-set numbers, and the condition indicators. Information stored in the program information code is described above. The program-mode field and the reserved bits are not stored; the contents of the corresponding bit positions in the assigned register locations for the PSV remain unchanged.

Programming Notes:

- The PCE may be switched from one program mode to another only by introducing a new PSV.
- The contents of the program-information-code field (bits 40-47) in the stored PSV are defined only following a program-exception interruption or execution of CALL PSV. For all other interruptions, this field is reserved and the program should not depend on its contents.

Exceptions Associated with PSV Introduction

Exceptions associated with the information in the current PSV may be recognized when the information is introduced into the PSV, or when the next instruction is fetched.

PSV Format Exceptions

When a 1 is introduced into reserved bit positions 32-35 of the PSV, a program-exception interruption for specification exception occurs after the PSV becomes active. The newly introduced PSV is stored unmodified, with the exception of the program information code, which identifies the cause of the interruption, and the possible exception of the instruction address, which may have been incremented by 2. The instruction address modifier (PSV bit 41) is reset to 0 if the instruction address has been incremented; otherwise it is set to 1.

An instruction-fetch reference may be attempted before the program-exception interruption. Therefore, it is unpredictable whether the stored PSV indicates the specification exception or an exception related to the attempted instruction fetch.

Other Exceptions

If an instruction-fetch reference is attempted before the program-exception interruption, an access, address, separation, or specification exception associated with the storage location of the fetched instruction may be recognized. It is then unpredictable whether the stored PSV indicates the specification exception or an exception related to the attempted instruction fetch. The instruction address stored because of the resulting program-exception interruption may be incremented by 2. The instruction address modifier (PSV bit 41) is reset to 0 if the instruction address has been incremented; otherwise it is set to 1. The other fields of the current PSV, except for the program information code, are stored unchanged.

Depending on processor model and the value in the instruction address field, when an address exception is detected, a specification exception may be indicated instead (see "Address Exception" in Chapter 3).

Address Control Vector

The address control vector (ACV) is 32 bits long and contains the information required for dynamic address relocation and for activating dynamic address translation. Dynamic address relocation, dynamic address translation, and the format and function of the ACV are discussed in Chapter 7. The active ACV is called the current ACV.

A new ACV is introduced at the same time a new PSV is introduced. Unlike the switching of PSV information, however, the current ACV is not stored because the program cannot alter its contents. The new ACV becomes active at the end of the interruption action that caused it to be introduced and remains unchanged until another PSV/ACV pair is introduced.

An ACV also participates in channel input/output operations. Chapter 8 discusses the relation of an ACV to channel I/O operations.

Exceptions Associated with ACV Introduction

Exceptions associated with the information in the current ACV may be recognized when the information is introduced into the ACV or when the next instruction is fetched.

ACV Format Exceptions

A program-exception interruption for a specification exception occurs after the ACV becomes active (1) when the origin field in the ACV designates an address that exceeds the maximum address in the PCE address space, (2) when the size field contains a value that is undefined or not provided by the PCE, or (3) when a 1 is contained in a reserved bit position (bit positions 0-7). The action taken by the PCE is the same as the action previously described for recognition of a PSV format exception.

Other Exceptions

When dynamic address translation is not invoked (bit 31 of the ACV is 0) and the origin field in the ACV designates the address of a physical main storage location that is not installed, a specification exception is recognized when the next instruction is fetched. The action taken by the PCE is the same as the action previously described for recognition of other exceptions associated with the PSV.

Floating-Point Status Vector

The floating-point status vector (FSV) is 24 bits long and contains the information required for proper execution of floating-point instructions. The FSV includes a register set number and other control and status information. In general, the FSV is used to (1) control floating-point register usage, (2) control the precision of floating-point operations, (3) control certain program-exception interruptions, and (4) hold and indicate floating-point errors associated with the program.

The floating-point feature provides eight FSVs, one associated with each priority level. A new FSV is made active as part of the interruption action performed by the PCE when a new priority level is dispatched. The new FSV becomes active at the completion of the interruption action. The active FSV is called the current FSV.

The program should initialize all FSVs by means of the floating-point instruction WRITE FLOATING-POINT STATUS VECTOR before executing any other floating-point instruction.

Floating-Point Status Vector Format

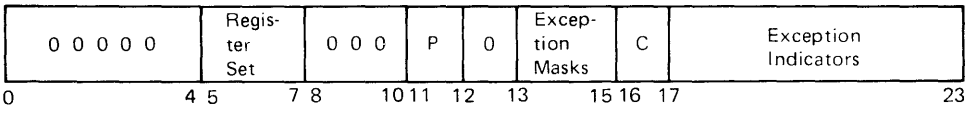


Figure 9-2. Floating-Point Status Vector Format

Figure 9-2 shows the FSV format; the following summarizes its meaning:

- ***Register Set:*** Bits 5-7 form the binary number of the floating-point register set available to programs executed at the priority level associated with the FSV.
- ***Precision Mode (P):*** Bit 11 controls the precision mode in which floating-point instructions are executed. Floating-Point instructions are executed in short-precision mode when the bit is 0 and in long-precision mode when the bit is 1.
- ***Exception Masks:*** Bits 13-15 are the three program-exception mask bits. Each bit is associated with a program exception recognized during execution of a floating-point instruction, as follows:

Exception Mask Bit	Program Exception
13	Significance
14	Exponent Overflow
15	Exponent Underflow

When a mask bit is 0, the exception results in a program-exception interruption. When a mask bit is 1, no program-exception interruption occurs. The significance and exponent-underflow masks (bits 13 and 15) also determine the manner in which floating-point addition, subtraction, division, and multiplication are completed.

- ***Floating-Point Check (C):*** Bit 16 indicates an equipment check associated with the floating-point feature. This bit is made a 1 and a system-check interruption occurs when equipment malfunctioning is detected during the execution of a floating-point instruction. The floating-point check bit is set in the FSV associated with the current priority level, and the floating-point instruction is terminated.
- ***Exception Indicators:*** Bits 17-23 identify program exceptions recognized during execution of floating-point instructions and detected by the floating-point feature. Each bit is associated with a program exception as follows:

Indicator Bit	Program Exception
17	Floating-Point-Operation Exception
18	Floating-Point-Privileged-Operation Exception
19	Floating-Point-Specification Exception
20	Floating-Point-Divide Exception
21	Significance Exception
22	Exponent-Overflow Exception
23	Exponent-Underflow Exception

The indicator bit is made a 1 and a program-exception interruption occurs when the associated exception is detected. If, however, an exception-mask bit is 1 at the time the associated exception occurs, the indicator bit is not made a 1 and a program-exception interruption does not take place.

- ***Reserved Bits:*** Bit positions 0-4 are reserved and must be 0. A floating-point-specification exception is recognized when an attempt is made to write a 1 into any of these bit positions. Bit positions 8-10 and 12 are also reserved, but are not checked for 0's when they are written. When an interruption occurs that is not caused by a floating-point check or program exception, the contents of bit positions 8-10 and 12 are unpredictable.

Programming Note: Bit positions 0-4, 8-10, and 12, which are reserved, should be written as 0's.

Referring to the FSV

Instructions are provided to read or modify all of the FSV or only a part of it. The instructions that read or modify the entire FSV can refer to the FSV associated with any priority level, including the current level. The instruction that can modify the entire FSV is supervisor-privileged. All floating-point instructions are described in Chapter 5.

Exceptions Related to the FSV

An exception related to the FSV is recognized at the time the FSV is written. A program-exception interruption for a floating-point-specification exception occurs when the storage operand of the WRITE FLOATING-POINT STATUS VECTOR instruction contains a 1 in bit positions 0-4 of the FSV. The exception is identified in the FSV associated with the current priority level, even if an FSV for a different priority level was designated in the storage operand. Execution of the WRITE FLOATING-POINT STATUS VECTOR instruction is suppressed.

Priority Levels

The PCE gives control to programs in response to requests for program execution. The action performed by the PCE to determine which program is to be given control is called *dispatching*. The PCE performs dispatching functions in response to requests from three sources: requests created by a program, requests signaled by I/O devices, and requests generated by the PCE as a result of detecting certain errors. To permit fast response to requests of high priority, eight priority levels are provided. A request for one of the priority levels must normally be present in order for program execution to occur.

The eight priority levels provided are numbered 0-7. Level 0 is defined as the highest priority, level 1 as the next highest priority, and so on to level 7, which is defined as the lowest priority. The priority level associated with the currently executing program (current PSV) is called the *current priority level.*

The PSV and ACV define the state of the associated program and its relationship to the PCE. A new ACV is introduced at the same time a new PSV is introduced and, thus, each PSV is associated with an ACV. A program is assigned an execution priority by means of its PSV/ACV pair. A unique set of permanently assigned register locations is associated with each priority level. These register locations are used to hold the PSV and ACV information that identifies a program to the PCE. The relative priority of a program is thus determined by the register locations in which its PSV/ACV pair is held. The assignment of register locations to hold PSV and ACV information is described in this chapter under "Interruption Action" and in Chapter 6.

Programming Notes:

- The priority structure and associated PSV and ACV register locations allow the definition of a distinct execution environment for each program. These facilities also allow a single copy of a program to be associated with two or more priority levels. When a new PSV/ACV pair is introduced, the state of the associated program is defined for the PCE. The program state includes the definition of the logical address space, general registers, and program status and control information used during program execution.

- A supervisory program can establish a program's states and relative priority when it initializes the PSV and ACV register locations. The state of a program is preserved automatically when program execution is interrupted and is restored when the PSV/ACV pair is again introduced. This capability permits fast response to requests for program execution.

- The PSV and ACV information stored in register locations may be inspected or modified using the register-indirect instructions. These instructions are described in Chapter 6.

Dual PSV/ACV Facility

The dual PSV/ACV facility provides a means for a PCE (each PCE in dual-PCE processors) to associate two programs with each priority level. Each program is defined by its associated PSV/ACV pair.

Primary and Secondary PSV/ACV Pairs

Two PSV/ACV pairs are associated with each priority level; one is designated the *primary* PSV/ACV pair and the other is the *secondary* pair. The format of a secondary PSV is identical to the format of a primary PSV. Primary and secondary ACVs also have identical formats. The primary and secondary PSV/ACV pairs are identified by the particular register locations used to hold the PSV and ACV. The primary PSVs are held in principal register sets 0 and 1; secondary PSVs are held in principal register sets 4, and 5. The ACVs paired with the PSVs are held in the corresponding adjunct register sets (0, 1, 4, and 5).

The instruction CALL PSV is provided to introduce the opposite (dual) PSV/ACV pair for the current priority level. If CALL PSV is executed when a primary PSV is active, the current (primary) PSV is stored and the secondary PSV/ACV pair for the current priority level is introduced. Similarly, if CALL PSV is executed when a secondary PSV is active, the current (secondary) PSV is stored and the primary PSV/ACV pair for the current priority level is introduced. CALL PSV is described in Chapter 4.

Exceptions resulting from execution of a program cause a program-exception interruption. (Program exceptions are described in Chapter 3, "Program Execution.") A program exception encountered while a secondary PSV is active causes the current (secondary) PSV to be stored and the primary PSV/ACV pair for the current priority level to be introduced. A program exception encountered while a primary PSV is active results in a request for program execution at priority level 0.

Programming Note: The dual PSV/ACV facility provides a mechanism that allows both a supervisory and an application program to execute at the same priority level. The primary PSV/ACV pair is normally used for the supervisory program; the secondary PSV/ACV pair may be used for any program.

Program Activation Vector

A program activation vector (PAV) is an 8-bit control vector that indicates which PSV/ACV pair (primary or secondary) is to be introduced when a new priority level is given control by the PCE. The PAV is associated with the eight priority levels on a bit basis; that is, bit 0 of the PAV is associated with level 0, bit 1 with level 1, and so on to bit 7, which is associated with level 7. If the bit position corresponding to the new priority level contains a 0, the primary PSV/ACV pair for that level is introduced; if the bit position contains a 1, the secondary PSV/ACV pair is introduced.

When the opposite (dual) PSV/ACV pair for the current level is introduced (by CALL PSV or by a program-exception interruption while the secondary PSV is active), the PAV is updated to indicate which PSV is made active. If the primary PSV is made active, the bit position corresponding to the current level is set to 0. If the secondary PSV is made active, the bit position corresponding to the current level is set to 1.

The PCE-control instructions WRITE PROGRAM ACTIVATION VECTOR and READ PROGRAM ACTIVATION VECTOR allow the contents of the PAV to be inspected or modified. For some processor models, modification of the state of the PAV bit associated with the current priority level is not allowed by hardware. When an interruption occurs, the current PSV is always stored in the register locations from which it was loaded regardless of the state of the PAV bit associated with the current priority level.

Programming Note: For those processor models that allow modification of the PAV bit associated with the current level, the instruction WRITE PROGRAM ACTIVATION VECTOR should not be used to change this bit. If the bit associated with the current level is changed when the PAV is written, program execution at the current level may be prematurely concluded. This will occur, for example, if a higher priority level is given control because of an interruption before the program removes the interrupt request for the current level. When program execution at the current level is resumed, the opposite PSV/ACV pair will be introduced. Further, if priority level 0 is given control because of a system check, the PAV provides a misleading indication by identifying the opposite PSV as being active when the interruption occurred.

Interruptions

An interruption is defined as the action performed by the PCE when control is taken from one PSV/ACV pair and given to another pair. The program associated with the PSV/ACV pair from which control is taken, is called the interrupted program. An interruption occurs when the PCE's dispatching mechanism determines that a new PSV/ACV pair is to be introduced, whether at the current priority level or at a different level. The interruption action is performed automatically by the PCE.

An interruption always involves storing the current PSV in its assigned register locations and introducing a new PSV/ACV pair. Processing resumes as specified by the new PSV/ACV pair. The stored PSV holds all necessary PCE status information relative to the program.

When program execution is interrupted because of a request that is not associated with an error, the stored PSV contains the address of the instruction that would have been executed next had the interruption not occurred, thus permitting automatic resumption of the interrupted program. When an interruption occurs as a result of an error, specifically a program exception or a system check, information in the stored PSV or in control vectors permits identification of the error and the instruction last executed. Refer to "Interruption Information" in this chapter for further description of the information made available following an interruption.

Interrupt Requests

The PCE's dispatching mechanism uses requests for program execution to determine which program is to be given control. When program execution at a higher priority level is requested, or when program execution at a lower priority level is requested and the request for the current level is removed, the dispatching process results in an interruption of the current program. For this reason, the requests for program execution are called *interrupt requests.* Interrupt requests always designate a specific priority level. An interrupt request, thus, represents a request for execution of the program defined by a PSV/ACV pair associated with the designated priority level. When interrupt requests are present for two or more enabled priority levels, program execution occurs on the highest priority of these levels.

There are three sources of interrupt requests: I/O devices, programs, and system checks. These sources of interrupt requests are identified by the control vectors in which requests from each source are held. Interrupt requests from I/O devices are held in the I/O interrupt request vector. Requests created by programs are held in the programmed interrupt request vector. System checks are errors detected by the PCE, the channel, or, when installed, the floating-point feature. When a system check is detected, an interrupt request is recorded in the error interrupt request vector. The error interrupt request vector also serves to identify the error.

An interrupt request is generated when a bit position of an interrupt request vector is set to 1. The interrupt request is removed when the bit position is set to 0.

I/O devices and executing programs can create requests for program execution at any of the eight priority levels. System checks always result in requests for program execution at priority level 0. The three types of interrupt requests function independently and may be present at one time in any combination. I/O and programmed interrupt requests for the same priority level can be present simultaneously on any level, whereas level 0 can have present any combination of all three types of interrupt requests. Further, two or more priority levels can, and often do, have interrupt requests present simultaneously.

Instructions are provided for inspecting or modifying the interrupt request vectors. These instructions are described under "Instructions for PCE Control" in this chapter.

Programming Notes:

- Although program execution at priority level 0 can be invoked by a request from any of the interrupt-request sources, this level is normally used for programs that handle system checks.

- Depending on the configuration of the system, a given priority level may be used only for a program invoked by interrupt requests from a single source, or the priority level may be used for a program that processes requests from more than one source. When the program is invoked by requests from a single interrupt-request source, the identity of the source is known implicitly. When the program is invoked by interrupt requests from several sources, it must explicitly examine the applicable sources to determine which one

generated the interrupt request. When more than one source generates an interrupt request at the same time and for the same level, the sequence in which the program tests the sources determines their relative priority.

- A request for program execution at a particular priority level normally must be present for the duration of program execution at that level. When the interrupt request that is sustaining program execution at a priority level is removed, program execution at that level is concluded at the completion of the instruction that removes the interrupt request. The requirement for an interrupt request to be present so that program execution can take place does not apply when the master mask is 0. Refer to "Enabling and Disabling" in this chapter for a description of the master mask.

- The PCE can enter the wait state with an interrupt request present when the associated priority level is disabled.

I/O Interrupt Request Vector

The I/O interrupt request vector (IOIRV) is an 8-bit control vector that holds interrupt requests generated by I/O devices. The control vector is associated with the eight priority levels on a bit basis; that is, bit 0 (the leftmost bit) of the control vector is associated with level 0, bit 1 with level 1, and so on to bit 7, which is associated with level 7. The presence or absence of an I/O interrupt request for a priority level is indicated by a 1 or 0, respectively, in the bit position associated with the priority level.

READ I/O INTERRUPT REQUEST VECTOR permits inspection of the IOIRV. I/O interrupt requests are generated, maintained, and removed under control of I/O devices. Therefore, modification of the IOIRV by the program is performed indirectly with an I/O instruction. Chapter 8 describes how the program can determine which I/O device is presenting an interrupt request when more than one device is assigned to the same priority level, and how the IOIRV may be modified indirectly.

Programming Note: Typically, I/O interrupt requests are generated asynchronously with respect to program execution because of events occurring at the I/O device. They are removed by the program by means of Input/Output instructions. An I/O interrupt request can be generated when the PCE is in the running state or in the wait state.

Programmed Interrupt Request Vector

The programmed interrupt request vector (PIRV) is an 8-bit control vector that holds interrupt requests created by programs. The association of the PIRV with the eight priority levels, and the indication of the presence or absence of a programmed interrupt request, are the same as for the IOIRV.

Five instructions are provided for inspecting or modifying the PIRV. READ PROGRAMMED INTERRUPT REQUEST VECTOR is used to inspect the control vector. OR WITH PROGRAMMED INTERRUPT REQUEST VECTOR and AND WITH PROGRAMMED INTERRUPT REQUEST VECTOR provide for generating and removing, respectively, the interrupt requests for any of the

priority levels. SET PROGRAMMED INTERRUPT REQUEST and RESET PROGRAMMED INTERRUPT REQUEST permit the program to generate or remove the programmed interrupt request for the current priority level.

Programming Note: The PIRV provides the primary mechanism for sustaining program execution at a priority level, particularly for programs that respond to interrupt requests from I/O devices and must remove the request from the IOIRV. A programmed interrupt request can be generated only when the PCE is in the running state.

Error Interrupt Request Vector

The error interrupt request vector (EIRV) is an 8-bit control vector in which interrupt requests are recorded when system checks are detected. Unlike the IOIRV and PIRV, the EIRV is associated only with priority level 0. Bit positions 0-5 of the EIRV are used to hold the interrupt request and identify the system check. When any of these bit positions is set to 1, a request for program execution at priority level 0 is created. If all of the bit positions 0-5 of the EIRV contain 0's, no interrupt request present is related to a system check. Refer to Chapter 10 for system check in dual-PCE processors.

Bit position 6 of the EIRV contains the instruction address modifier bit. This bit position is not used as an error indicator and, when set to 1, does not create an interrupt request. Bit position 7 is reserved. The section "Interruption Information" in this chapter describes the format and contents of the EIRV.

Two instructions are provided for inspecting or modifying the EIRV. READ ERROR INTERRUPT REQUEST VECTOR permits the program to examine the EIRV, while WRITE ERROR INTERRUPT REQUEST VECTOR allows the program to modify its contents. The write instruction must be used to remove an error interrupt request; the PCE does not automatically clear the EIRV. The write instruction can also be used to generate an interrupt request by introducing a 1 into any of bit positions 0-5 of the EIRV.

When an error interrupt request is present, channel-I/O burst transfers are suspended and are not resumed until all of bit positions 0-3 and 5 of the EIRV are made 0. Detailed information on suspension of channel-I/O burst transfers is provided in Chapter 8.

Programming Note: An error interrupt request can be generated when the PCE is either in the running state or in the wait state.

Enabling and Disabling

The priority levels can be enabled or disabled for I/O interrupt requests and programmed interrupt requests. When a priority level is enabled, a PSV/ACV pair at that priority level can be given control in response to one of these interrupt requests. When a priority level is disabled, I/O interrupt requests and programmed interrupt requests for that level remain pending until the level is enabled.

Whether priority levels are enabled or disabled is indicated and controlled by bits in two masks: the master mask and the common mask. The two masks provide a hierarchy of control. The 1-bit master mask controls priority levels 1-7. The 8-bit common mask provides individual control for each of the eight priority levels.

Depending on processor model, bit 0 of the common mask can disable priority level 0 for system-check interrupt requests indicated in the EIRV, in addition to I/O interrupt requests and programmed interrupt requests. Because of possible degradation of system integrity when system-check interrupt requests are held pending, the disabling of priority level 0 for system-check interruptions is not recommended. See "Common Mask" in this chapter.

Programming Notes:

- The mask bits are not changed as part of the priority level dispatching process.
- The capability of disabling a priority level is intended primarily for disabling I/O interrupt requests.
- The operation of the PCE-control instruction DISPATCH NEW LEVEL is independent of the states of the master mask and common mask. The states of the master mask and common mask have no effect on the introduction of the dual PSV/ACV pair for the current level by CALL PSV or by a program-exception interruption.

Master Mask

The master mask is a 1-bit mask that determines whether a PSV/ACV pair at priority levels 1-7 can be given control in response to an interrupt request.

When the master mask is a 1, the priority levels are individually enabled or disabled by the bits of the common mask. When the master mask is 0, levels 1-7 are all disabled except for the current level, which remains enabled. No priority level other than the current level and level 0 is eligible for selection by the PCE's dispatching algorithm. Program execution on the current level continues even if the interrupt request for the current level is removed or the current level is disabled by the common mask. The wait state cannot be entered when the master mask is 0. The master mask does not disable priority level 0. See "Common Mask".

Three instructions are provided to inspect or change the master mask. READ MASTER MASK allows the program to determine the current value of the mask. SET MASTER MASK and RESET MASTER MASK provide for making the mask 1 or 0, respectively.

When RESET MASTER MASK is executed, the priority levels are disabled at the completion of the instruction. When a priority level other than level 0 is active at the time the instruction is executed, program execution continues on the current level so long as the mask remains 0 and no interrupt requests for priority level 0 are generated. If an interrupt request for priority level 0 is generated, causing level 0 to be dispatched, program execution continues at priority level 0 as long as the master mask remains 0. Similarly, if level 0 is active at the time RESET MASTER MASK is executed, execution continues at level 0 as long as the master mask is 0. When SET MASTER MASK is executed, the priority levels become individually enabled or disabled by the common mask at the completion of the instruction.

In dual-PCE processors, the master masks, one in each PCE, are interlocked so that simultaneous processing does not occur in both PCEs when both master masks are disabled. For a detailed description of the master mask in dual-PCE processors, see Chapter 10.

Programming Notes:

- The master mask is intended primarily for use in situations in which the program must not be exposed to I/O or program interruptions. For example, manipulation of a queue shared among two or more programs that execute at different priority levels can be performed by making the master mask 0 before operating on the queue, and restoring the mask to 1 when finished. Thus, the queue cannot be altered by a second program as a result of an I/O interruption during the time it is being operated upon by the first program. Note that this example does not apply when the queue is shared by a program executed at priority level 0.

- The master mask can also be used to continue program execution after the interrupt request for the current priority level is removed. This situation may arise, for example, when handling a system check. If the master mask is made a 0 by the program executing at priority level 0 and the EIRV is then cleared, program execution on level 0 will continue while allowing a second error, if any, to be properly indicated in the EIRV. After the error has been processed, the program can make the master mask a 1, which concludes program execution on level 0. If, however, a second error has been indicated in the EIRV, program execution on level 0 will continue, allowing the second error to be processed.

- If a system-check interruption occurs while the master mask is 0, the instruction DISPATCH NEW LEVEL must be executed in order to return control from level 0 without altering the state of the master mask.

Common Mask

The common mask is an 8-bit mask that provides selective control over the priority levels within a PCE for I/O interrupt requests indicated in the IOIRV and programmed interrupt requests indicated in the PIRV. The mask bits (0-7) correspond with priority levels 0-7. The common mask provides control over levels 1-7 only when the master mask is a 1, and over level 0 at any time.

When a common-mask bit is 1, the associated priority level is enabled for program execution in response to I/O and programmed interrupt requests. When the mask bit is 0, the associated priority level is disabled and program execution in response to interrupt requests for that level cannot occur.

Depending on processor model, bit 0 of the common mask can disable priority level 0 for system-check interrupt requests indicated in the EIRV, in addition to I/O interrupt requests and programmed interrupt requests.

Two instructions are provided to inspect or change the common mask. READ COMMON MASK retrieves the contents of the mask. WRITE COMMON MASK allows the program to change the mask bits.

When the master mask is a 1, and a common-mask bit is changed from 1 to 0 or from 0 to 1, the associated priority level is considered disabled or enabled, respectively, at the completion of the instruction. When the common-mask bit for the current level is made 0 and the master mask is a 1, program execution at the current level is concluded with the completion of the instruction. Refer to "Master Mask" in this chapter for more details about the interdependencies of the common mask and master mask.

For processor models that can disable priority level 0 for error interrupt requests, disabling priority level 0 causes error interrupt requests to be held pending. When an error is detected during execution of an instruction and level 0 is disabled, the instruction is terminated and the PCE continues to fetch and execute instructions with unpredictable results. Therefore, because of the possible degradation of system integrity, the program should not disable level 0 during normal system operation.

Programming Note: If the execution of WRITE COMMON MASK writes all 0's into the common mask and the master mask is a 1, or SET MASTER MASK is executed when the common mask is all 0's, the PCE enters an uninterruptible wait state, except for possible system checks from pending I/O operations. Reset must be manually initiated to leave the uninterruptible wait state.

Priority Level Dispatching

The action performed by the PCE to select a priority level for program execution is called *priority level dispatching*. Conceptually, the PCE selects after every operation the priority level at which the next operation is to be performed. The selection is made after one operation is performed and before a subsequent operation is started. The entire execution of a non-interruptible instruction is an operation. For interruptible instructions, an operation may consist in partial execution of the instruction. The priority level at which the next operation is performed is determined by the interrupt requests present and the priority levels enabled. The instruction DISPATCH NEW LEVEL (described in this chapter under "Instructions for PCE Control") allows the program to select the priority level at which the next operation is to be performed.

A new priority level is selected in the following cases:

- When an interrupt request exists for a level higher in priority than the current level, and the higher priority level is enabled.
- When all interrupt requests for the current priority level are removed, or the current level is disabled, and another interrupt request is present for an enabled priority level.

The wait state is entered at the completion of the current instruction when all interrupt requests for the current priority level are removed, or the current level is disabled, and no other interrupt request is present for an enabled priority level.

The PCE is interruptible in the wait state, provided the priority level designated by an interrupt request is enabled. To leave the wait state without manual intervention, a priority level must be enabled for an interrupt request.

The PCE is in the wait state when no interrupt request is present for any enabled priority level and the master mask is 1. When a request is present for an enabled priority level or the master mask is 0, the PCE is in the running state.

The current PSV and FSV are not stored when the wait state is entered. When the wait state is ended because of an interrupt request, priority level dispatching and the associated interruption action are resumed as if the interrupt request were present at the completion of the last instruction. If a new PSV/ACV pair is not introduced when the wait state is ended (execution resumes at the level that was active when the wait state was entered), any prefetched instructions are not discarded.

Current and Last Priority Levels

The level numbers of the current priority level (CPL) and the last priority level (LPL) are automatically maintained by the PCE. The CPL number is the 3-bit binary number of the active level. The LPL number is the 3-bit binary number of the level active immediately before switching to the current PSV. The CPL and LPL numbers are maintained in two control vectors.

As part of the interruption action associated with giving control to a program at a different priority level, the contents of the LPL vector are replaced with the CPL number, and then the contents of the CPL vector are replaced with the number of the new level. If the opposite PSV/ACV pair at the current priority level is introduced, the contents of the LPL vector are replaced with the number of the current level.

The current and last priority-level numbers are not changed when the wait state is entered. Further, they are not changed when the running state is reentered by giving control to the same priority level that was active when the wait state was entered.

An instruction is provided for reading the contents of the CPL and LPL vectors. READ CURRENT AND LAST LEVELS loads both level numbers into a designated general register.

Programming Note: When one copy of a program is executed at two or more priority levels, the program can determine the current level number by executing READ CURRENT AND LAST LEVELS. The LPL number is provided primarily to allow the error-handling program executing at priority level 0 to determine which level was active at the time a system-check interruption occurred. Refer to the discussion in this chapter under "Interruption Information" for further details.

Summary of the Priority Level Dispatching Process

This summary describes the action performed by the PCE to select the next priority level for program execution. Conceptually, this action is repeated after execution of every instruction, or each unit of operation for the interruptible instructions. Determining which priority levels are eligible for selection is performed by combining all interrupt requests and excluding those associated with disabled priority levels. A PSV/ACV pair associated with the highest eligible priority level is then given control.

Figure 9-3 summarizes the steps involved in determining the priority levels that can be selected. The illustration and description present the selection process as if the steps were performed sequentially. Two or more steps may actually be performed in parallel and not necessarily in the sequence presented.

Step 1:
W<0..7> <— IOIRV<0..7> v PIRV<0..7>

Step 2: (Note)
W<0..7> <— W<0..7> • CM<0..7>

Step 3: (Note)
W<0> <—
EIRV<0> v EIRV<1> v EIRV<2> v
EIRV<3> v EIRV<4> v EIRV<5> v W<0>

Step 4:
W<1..7> <— W<1..7>•
(MM | | MM | | MM | | MM | | MM | | MM | | MM)

Step 5:
W<CPL> <— W<CPL> v !MM

Where:

<—	"is replaced by"
v	"logically ORed bit by bit with"
•	"logically ANDed bit by bit with"
| |	"is concatenated with"
¬	"the logical inverse of"
a<m>	Bit position m of vector or mask a
a<m..n>	Bit positions m through n of vector or mask a
CM	The 8-bit common mask
CPL	The current priority-level number
EIRV	The 8-bit error interrupt request vector
IOIRV	The 8-bit I/O interrupt request vector
MM	The 1-bit master mask
PIRV	The 8-bit programmed interrupt request vector
W	An 8-bit working vector denoting the step-by-step result

Step 1 determines the presence of I/O and programmed interrupt requests for priority levels 0-7.

Step 2 excludes from possible selection priority levels disabled by the common mask when one or more mask bits are 0.

Step 3 determines the additional presence of system-check interrupt requests for priority level 0.

Note: *For processor models that can disable priority level 0 for system-check interrupt requests by resetting bit 0 of the common mask to 0, Steps 2 and 3 are conceptually reversed. That is, the additional presence of system-check interrupt requests for priority level 0 is determined in Step 3 before excluding the priority levels from possible selection that are disabled by the common mask when one or more mask bits are reset to 0 as performed in Step 2.*

Step 4 excludes from possible selection all priority levels disabled by the master mask when the mask is 0.

Step 5 enables the current priority level and generates a pseudo interrupt request for the current level when the master mask is 0.

Figure 9-3. Determining the Dispatchable Priority Levels

At the completion of Step 5, the 8-bit result vector (W) designates, with 1's in the respective bit positions, all enabled priority levels for which interrupt requests are present. The bit positions of the result vector correspond, left to right, with priority levels 0-7.

The selection of the the highest-priority level is determined by the leftmost bit position of the result vector (W) in which a 1 appears. If the selected level is the same as the current level, program execution continues at the current level with the execution of the next operation. If the selected level and current level are different, program execution at the current level is interrupted, and the program at the selected level is given control.

Programming Note: The wait state is entered when the result vector (W) is all 0's; that is, when the master mask is 1 and (1) all priority levels for which interrupt requests are present are disabled by the common mask, or (2) no interrupt requests are present.

Interruption Action

An interruption consists of storing the current PSV, updating certain control information, and introducing a new PSV/ACV pair. Permanently assigned registers are used as the locations in which the current PSV is stored and from which the new PSV and ACV are loaded. The current PSV is always stored back into the same register locations from which it was loaded. The current ACV is not stored because its contents are not changed. This interruption action is performed automatically by the PCE; no action by the program is necessary to store PSV information or to introduce a new PSV/ACV pair.

The PSV is held in a permanently assigned even and odd numbered pair of consecutive registers in a principal register set. Corresponding to the principal register locations for the PSV is a register location in an adjunct register set that is assigned to hold the ACV. The corresponding PSV and ACV register locations are considered to be paired together; that is, a PSV/ACV pair is always introduced from corresponding register locations. See Chapter 6 for a description of the register locations permanently assigned to hold PSV and ACV information and of the instructions that can be used to refer to these register locations.

Control Given to a Program at a New Priority Level

The active program is interrupted when the PCE's dispatching algorithm determines that a program is to be executed on a priority level that is different from the current level. (Refer to "Priority-Level Dispatching" in this chapter). The interruption action consists of (1) storing the current PSV, (2) replacing the contents of the LPL vector with the CPL number, (3) replacing the contents of the CPL vector with the new priority level number, and (4) loading the new PSV and ACV. When the floating-point feature is installed, the interruption action also includes switching from the current FSV to the new FSV.

The new PSV and ACV are loaded from the register locations that are associated with the new priority level. The dual PSV/ACV facility associates with each priority level, register locations that hold two PSV/ACV pairs. The PAV controls which of the two pairs is introduced when a program at a new priority level is given control. If the PAV bit position corresponding to the new priority level contains a 0, the primary PSV/ACV pair for that level is introduced; if the bit position contains a 1, the secondary PSV/ACV pair is introduced.

When the floating-point feature is installed, the current FSV becomes inactive and the FSV associated with the new priority level is made the current FSV. The FSVs associated with the eight priority levels are accessible with the instructions READ FLOATING-POINT STATUS VECTOR and WRITE FLOATING-POINT STATUS VECTOR.

Control Given to a New Program at the Current Level

The active program is interrupted and control is given to a new program at the current level in two cases: (1) when the instruction CALL PSV is executed, and (2) when a program exception occurs while a secondary PSV is active. If CALL PSV is executed when a primary PSV is active, the current PSV is stored in the register locations assigned to the primary PSV for the current level, and the secondary PSV/ACV pair for the current level is introduced. Similarly, if CALL PSV is executed when a secondary PSV is active, the current PSV is stored in the secondary-PSV register locations for the current level and the primary PSV/ACV pair for that level is introduced. A program exception that occurs while a secondary PSV is active causes the current PSV to be stored in the register locations assigned to the secondary PSV and the primary PSV/ACV pair for the current level to be introduced.

The interruption action consists of (1) storing the current PSV, (2) loading the new PSV/ACV pair, (3) updating the program activation vector to indicate which of the two PSV/ACV pairs is made active, and (4) replacing the contents of the LPL vector with the number of the current level. The new PSV and ACV are loaded from the register locations that are assigned to the inactive PSV/ACV pair for the current priority level; that is, the PSV/ACV pair loaded is opposite from the pair that is active at the time the interruption occurs. The program activation vector is updated to indicate the PSV/ACV pair that is made active. If the primary pair is made active, the bit position corresponding to the current level is set to 0. If the secondary pair is made active, the bit position corresponding to the current level is set to 1.

Point of Interruption

Conceptually, an interruption is permitted between operations; that is, an interruption can occur after one operation is performed and before a subsequent operation is started, where an *operation* is defined as the execution of an instruction. However, the MOVE and COMPARE LOGICAL operations are interruptible; that is, an interruption is allowed after partial execution of these instructions. Whenever this publication refers to points of interruption, including those that occur within the execution of interruptible instructions, the term *unit of operation* is used. The use of this term considers that the entire execution of a noninterruptible instruction consists, in effect, of one unit of operation.

With the exception of I/O interrupt requests, interruptions may occur after execution of each noninterruptible instruction. Depending on processor model, I/O interrupt requests may be delayed for up to three noninterruptible instructions if the PCE encounters an instruction string having short execution times.

Execution of an interruptible instruction is considered to consist of a number of units of operation, and an interruption is permitted between units of operation. Depending on processor model, up to eight units of operation may be executed between points in the operation at which an interruption is allowed. In this case, the number of units of operation executed without allowing interruptions is

predetermined. After each predetermined number of units of operation, the operand addresses and count values are updated to correspond to the amount of data processed. The specific predetermined number of units of operation is fixed, except for the first and last execution groups.

When a program-exception interruption occurs, the instruction-address and instruction-address-modifier fields in the stored PSV designate the point of interruption. When the instruction address modifier is 0, the instruction-address field contains a value 2 greater than the logical address of the first halfword of the instruction in which the program exception occurred. When the instruction address modifier is 1, the instruction address field contains the logical address of the first halfword of the instruction in which the program exception occurred.

Types of Ending

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

When execution of an instruction is *completed*, results are provided as called for in the definition of the instruction. If an interruption occurs after an instruction is completed without the detection of a program-exception condition, the instruction address in the stored PSV designates the next instruction to be executed. If an instruction is completed following the detection of a program-exception condition that results in a program-exception interruption, the instruction address and instruction address modifier in the stored PSV designate the completed instruction.

When execution of an instruction is *suppressed*, the instruction is effectively not executed. The contents of any result fields, including the condition indicators, are not changed. The instruction address and instruction address modifier in the PSV stored because of an interruption designate the suppressed instruction. Instruction execution is suppressed only for program-exception interruptions.

When execution of an instruction is *suspended*, the contents of any fields due to be changed by the instruction may be partially updated. The operation may have replaced all, part, or none of the contents of the designated operand locations, and it may have changed the condition indicators, an address value, or a count value, if such change was called for by the instruction. The instruction may be retried without software adjustment of register values, assuming the cause of suspension is removed.

When execution of an instruction is *terminated*, the contents of any fields due to be changed by the instruction are unpredictable. The operation may have replaced all, part, or none of the contents of the designated result fields and may have changed the condition indicators if such change was called for by the instruction. If an instruction is terminated following the detection of a program-exception condition that results in a program-exception interruption, the instruction address and instruction address modifier in the stored PSV designate the terminated instruction. If the interruption occurs because of a system check, the instruction address in the stored PSV and the instruction address modifier in the EIRV designate the terminated instruction.

Figure 9-4 includes a summary of the types of endings for the various types of interruptions.

Interruption Type	Instruction Ending	Instruction Address	Program Information Code	Other Information
CALL PSV	Completed	Next instruction	00000000	Program conventions
Programmed	Completed	Next instruction	1	PIRV 2 and program conventions
I/O	Completed	Next instruction	1	IOIRV 2 and BSTAT 3
Program exception (secondary PSV)	Completed, suppressed, suspended, or terminated	Failing instruction 4	1mxxxx00	CPL identifies current level
Program exception (primary PSV)	Completed, suppressed, suspended, or terminated	Failing instruction 4	1mxxxx00	EIRV=000100m0, LPL identifies interrupted level
Channel exception	Completed	Next instruction 4	1	EIRV=000110m0, LPL identifies interrupted level, PAV identifies PSV on level 6
System check (except program and channel exception)	Completed, suspended 7, or terminated.	Next or failing instruction 4 5	1	EIRV=zzz0zzm0. LPL identifies interrupted level, PAV identifies PSV on level 6

Where:

1 The program-information-code field is reserved for these interruption types; its contents depend on processor model.

2 The PIRV or IOIRV bit position that corresponds to the priority level given control contains a 1.

3 The basic status register (BSTAT) associated with an I/O device indicates whether that device generated the I/O interruption. Chapter 8 describes the BSTAT.

4 The instruction address designates the instruction, or 2 bytes beyond it, as indicated by the instruction address modifier. Note that for program exception (primary PSV), the instruction address modifier is made available to the program in both the program information code and the EIRV.

5 When bit position 4 of the EIRV contains a 1, indicating a channel I/O check, the instruction address is associated with the next sequential instruction to be executed. Otherwise, the instruction address is associated with the failing instruction.

6 The PAV correctly identifies which PSV/ACV pair (primary or secondary) was active at a priority level if the interrupted program did not change the state of the PAV bit associated with its priority level.

7 Detection of a parity error on a PIO read operation causes the instruction to be suspended (storing of the data is inhibited).

m Bit position containing instruction address modifier.

x Bit positions containing program exception code. Chapter 3 identifies the program exception codes.

z Bit positions in EIRV indicating cause of system check.

Figure 9-4. Summary of Interruption Information

Programming Note: Normally, instruction execution is terminated only as a result of equipment malfunctioning. Program exceptions that result in an interruption generally cause the unit of operation to be suppressed, suspended, or completed. In some cases, however, the instruction may be terminated. The specific cases are noted under "Program Exception Conditions" in Chapter 3 and in the individual instruction descriptions.

Execution of Interruptible Instructions

Execution of an interruptible instruction (MOVE or COMPARE LOGICAL operations) is completed when all units of operation associated with the instruction are completed. When execution is completed, the instruction address in the PSV designates the next instruction to be executed. When an interruption occurs after completion of a unit of operation, all prior units of operation are completed.

On completion of a unit of operation other than the last one, the instruction address in the PSV stored because of an I/O interruption designates the interrupted instruction. For a system-check interruption because of a channel I/O check, the instruction address in the stored PSV and the instruction address modifier in the EIRV designate the interrupted instruction. The address and count values are adjusted such that execution of the interrupted instruction can be resumed from the point of interruption when the PSV stored because of the interruption is made the current PSV.

When a unit of operation of an interruptible instruction is terminated, the contents, in general, of any fields due to be changed by the instruction are unpredictable. When termination occurs because of a system-check interruption, the instruction address in the stored PSV and the instruction address modifier in the EIRV designate the interrupted instruction.

For termination because of a program-exception interruption, the instruction address and instruction address modifier in the stored PSV designate the interrupted instruction. The address and count values are adjusted to correspond to the unit of operation last completed before the interruption occurred. One or more units of operation may have terminated before the interruption. The number terminated depends on the processor model and the point in the operation at which the exception is detected.

Programming Note: The effect of resuming an interruptible instruction from the point of interruption after completion or suspension of a unit of operation is the same as if the instruction were executed without interruption.

Interruption Information

The PSV stored as part of the interruption action 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 interruptions because of program exceptions and system checks, an instruction-address-modifier bit is also stored that permits the program to identify the instruction being executed when the interruption occurred, and, when appropriate, resume execution of the interrupted program.

Depending on processor model, when an access or separation exception causes a program-exception interruption, the block index of the PCE address in error is placed in the EBI register associated with the active ACV.

Source Identification

To identify the source of an interruption, seven types of interruptions are defined: CALL PSV, programmed, I/O, program exception (secondary PSV), program exception (primary PSV), channel exception, and system checks other than program exception (primary PSV) and channel exception. For most of these types, the cause is defined by additional information made available to the program. The specific information and its location depends on the interruption type. Figure 9-4 gives the location of specific information for each type of interruption.

When an interruption occurs because of a floating-point exception or a floating-point check, information identifying the floating-point exception or check is provided in bit positions 16-23 of the FSV that was active at the time the interruption occurred. Otherwise, these bit positions in the FSV are not changed as a result of an interruption.

Depending on processor model, certain high-order bits of the instruction address that are not needed to represent the maximum address in the PCE address space may not be stored with the current PSV when an interruption occurs. The contents of bit positions in the PSV register locations corresponding to the instruction-address bits that are not stored either remain unchanged or are set to 0, depending on the particular bit position.

Programming Note: Introducing a PSV with an instruction address that is greater than the maximum address in the PCE address space causes a program-exception interruption. Depending on processor model, a high-order bit position of the instruction address, which contained a 1, may be set to 0 when the PSV is stored. Thus, the stored instruction address may have been modified such that it is less than the maximum address in the PCE address space, making it appear to be valid.

System-Check Interruption

The system-check interruption provides a means for reporting to the program an error associated with the operation of system equipment, or with the execution of a supervisory program. The group of system checks includes those associated with machine errors, I/O errors, channel exceptions (usually caused by a programming error), and the program exceptions detected while a primary PSV is active (usually indicating an error in a supervisory program). System checks result in an interrupt request for program execution at priority level 0 and are indicated in the error interrupt request vector (EIRV). The information in the EIRV may be used to determine the cause of the system-check interruption.

Error Interrupt Request Vector Format

Figure 9-5 shows the 8-bit EIRV format; the following describes the bit meanings. Each system check is described under "System Checks" in this chapter.

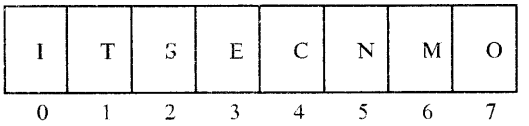


Figure 9-5. Format of Error Interrupt Request Vector

Bit(s)	Meaning
0	I/O Control Check (I)
1	I/O Timeout Check (T)
2	Storage Data Check (S)
3	Exception (E)
4	Channel I/O Check (C)
5	Internal Control Check (N)
2 and 5	Internal Data Check (S and N)
6	Instruction Address Modifier (M)
7	Reserved

Bits 0-5 provide a summary indication of the system check.

Bit 6 is the instruction address modifier. The bit is inspected by the program to determine whether the instruction-address field in the stored PSV contains the address of the first halfword of the instruction, or 2 bytes beyond it. The instruction designated by the instruction address and the instruction address modifier is either (1) the failing instruction associated with the system check, or (2) the next instruction to be executed when the system-check interruption is caused by a channel I/O operation.

Bit 6 is set to 1 when the instruction-address field contains the address of the first halfword of the instruction. When the instruction-address field contains a value 2 greater than the address of the first halfword of the instruction, bit 6 is left unchanged. Because the PCE does not change the state of bit 6 for the latter case, the program must set bit 6 to 0 before permitting a subsequent system-check interruption to occur. This ensures that bit 6 will be 1 or 0, respectively, for the two cases, and thus allows the program handling the system-check interruption to determine which of the two instruction-address values was stored.

The value of bit 6 is significant when an interruption occurs because of any system check. That is, the value of bit 6 does not depend on the type of error indicated in bit positions 0-5 of the EIRV.

Bit 7 is reserved; it is read as 0 and should be written as 0.

Programming Note: When a program is given control on priority level 0, it cannot be interrupted because of a system check. Therefore, the level-0 program should retrieve the contents of the EIRV and then clear it as soon as possible, so that a subsequent error, if any, can be properly indicated in the control vector. This procedure can be performed by executing SET PROGRAMMED INTERRUPT REQUEST before clearing the EIRV, thus holding program execution on level 0.

System Checks

Equipment malfunctions and other errors that cause system-check interruptions are referred to as system checks. System checks are indicated to the program in the EIRV. When a system check is detected, one or more bits in the EIRV are set to 1, depending on the specific system check; the remaining bits in the EIRV are left unchanged.

Four major classes of system checks are defined and are distinguished by the bits of the EIRV that are used to indicate the system check. The four classes are: input/output checks, machine checks, certain program exceptions, and channel exceptions.

Input/output checks are indicated with EIRV bits 0 and 1, separately. These system checks are usually caused by an I/O device. They include invalid parity and invalid control-signal responses detected by the channel during an I/O operation. Bit 4 of the EIRV is also set to 1 if the system check results from a channel I/O operation. Depending on processor model, EIRV bit 5 may also be set to 1 for certain system checks detected during a channel I/O operation. If the system check results from a programmed I/O operation, bit 4 is left *unchanged.*

Machine checks are indicated with EIRV bits 2 and 5, separately or together. These checks include errors detected on information fetched from storage, from a principal or adjunct register, from the translation table, or from the translation lock table. Machine checks also include errors detected internally on PCE, channel, or floating-point-feature control logic. Machine checks detected as a result of channel I/O operations are indicated by also setting EIRV bit 4 to 1.

Program exceptions are indicated with EIRV bit 3 when they are detected while a primary PSV is active. Program exceptions detected while a secondary PSV is active are not treated as system checks, and do not cause a system-check interruption. Refer to Chapter 3 for a detailed description of program exceptions.

Channel exceptions are indicated with EIRV bits 3 and 4, together. Thus, bit 4 being 1 differentiates a channel exception from a program exception. This class represents system checks detected by the channel during channel I/O operations, usually resulting from a programming error. Channel exceptions include such programming errors as invalid storage address and ACV format.

Each system check is described in detail in this section under the appropriate heading, such as "I/O Control Check."

Programming Notes:

- When handling a system-check interruption, the program executing on priority level 0 should read the contents of the EIRV and then clear it as soon as possible. This action will allow a subsequent system check, detected while level 0 is still active, to be indicated properly. Clearing the EIRV also enables the channel for channel I/O burst transfers, provided the channel mask is 1.

- If two or more system checks are detected before the program can retrieve the contents of the EIRV, the resulting indication may be ambiguous. For example, if a storage data check and an internal control check are detected, the resulting content of the EIRV indicates an internal data check. This situation can arise (1) when multiple system checks are detected during one operation, (2) when a system-check interruption occurs and a subsequent system check is detected before the program clears the EIRV, or (3) for processor models that can disable priority level 0 for system-check interrupt requests, bit 0 of the common mask is reset to 0, thus allowing the indications of successive system checks to be accumulated.

  Since PCE operation may be unpredictable when a system-check interruption is not taken, the priority level 0 program should minimize, as much as it reasonably can, the possibility of encountering a system check. In particular, the program assigned to the primary PSV for priority level 0 should be thoroughly tested so as to preclude a program exception, and the program assigned to either PSV for priority level 0 should use only those facilities of the PCE necessary to perform its function.

Because of the possible loss of PCE integrity when a system-check interruption cannot be taken, the program on processor models that can disable priority level 0 for system-check interrupt requests should not disable priority level 0 for any reason during normal system operation.

Note: *The following detailed description of system checks applies when detection of the system check causes an interruption; that is, the description applies when the current priority level is other than 0 at the time the system check is detected. This assumption is also made throughout this manual wherever the description refers to a system check. If priority level 0 is active, or for processor models that can disable priority level 0 for system-check interrupt requests and priority level 0 is disabled, detection of a system check does not cause an interruption, and the PCE proceeds to fetch and attempt to execute what it fetched as the next instruction. The results of this action are unpredictable.*

- Whether or not the interruption occurs, the system check is indicated in the EIRV. However, if the interruption does not occur, and the system check is related to the PCE's processing of an instruction or interruption, the manner in which the instruction execution or interruption action is ended is unpredictable. If the system check is related to the execution of a channel I/O operation, the operation is terminated. In either case, the system-check interruption remains pending until the program executing on priority level 0 services the interrupt request, or for processor models that can disable priority level 0 for system-check interrupt requests, until the interruption occurs when priority level 0 is subsequently enabled.

- If a program exception occurs while a primary PSV is active and the system-check interruption is not taken, the operation is completed with unpredictable results.

I/O Control Check: Bit 0 of the EIRV indicates that a data check (invalid parity) was detected on information transferred from an I/O device to the channel. Bit 4 and, depending on processor model, bit 5 of the EIRV are also set to 1 if the error occurs during a channel I/O operation. Bits 4 and 5 are left unchanged if the error occurred during a programmed I/O operation.

I/O control check is indicated when the channel detects invalid parity on inbound information, and the I/O device signals that invalid parity is to be treated as a system check. This information includes the data read during a programmed or channel I/O operation, as well as the address information, if any, read as part of a channel I/O operation. Invalid parity detected on channel-control-vector information, read as part of a channel I/O operation, is always treated as a system check.

Whenever I/O control check is indicated because of invalid parity on inbound information, the programmed I/O operation is suspended (storing of the data is inhibited) or the channel I/O operation is terminated.

When the I/O device signals that invalid parity is not to be treated as a system check, the information is stored with correct parity, and the operation proceeds to normal completion. For programmed I/O operations, result condition 1 is indicated to the program in place of the I/O-control-check indication. For channel I/O operations, the information transfer proceeds with no error indication.

I/O Timeout Check: Bit 1 of the EIRV indicates either that an expected control-signal response from a device was not received by the channel within the allowable time interval, or that a control-signal response from a device was held active beyond the allowable time interval or that an incorrect control-signal response from a device was detected. Bit 4 and, depending on processor model, bit 5 of the EIRV are also set to 1 if any of the conditions occur during a channel I/O operation. Bits 4 and 5 are left unchanged if the timeout occurs during a programmed I/O operation.

The actual cause of the I/O time-out check can originate in the PCE, channel, or adapter from such abnormal conditions as invalid control-signal sequences, invalid command codes, invalid parity on outbound data transfers, or hardware malfunction. For example, the absence of an expected control-signal response is detected when an I/O instruction is executed specifying a PIO address that is not recognized by any attached device. The PIO address may not be assigned to any attached device, or the device to which the PIO address is assigned may be powered off.

Certain incorrect responses received either in combination with or in place of an expected response may not cause a system-check interruption. An example of the latter case is the instruction INPUT/OUTPUT (halfword) which is specified as compatible with halfword-mode devices only. However, when the instruction is executed designating the PIO address of a byte-mode device, the operation may be completed with no system-check indication, in which case the results are unpredictable. Whether the system check is indicated depends on the specific byte-mode device.

Detection of an I/O timeout check causes the programmed or channel I/O operation to be terminated.

Storage Data Check: Bit 2 of the EIRV indicates that a data check was detected in the information fetched from or the information being stored in main storage. Such information is considered invalid. The malfunction causing the information to become invalid may be associated with the storage location from which the information was fetched, or it may be located on the path to or from storage. Bit 4 of the EIRV is also set to 1 if the data check results from a channel I/O operation.

Except for the following cases, detection of a storage data check causes the current operation to be terminated. If the storage data check results from an instruction-fetch reference, execution of the fetched instruction is suppressed. If the storage data check results from a channel I/O operation, the invalid information is not transmitted to the device.

Depending on processor model:

- The prefetching of invalid information from storage may result in a system-check interruption before the information is actually needed. Further, the interruption may occur even if the invalid information would not have been used (such as instruction prefetch where the prefetched instruction is discarded).

- In addition to bit 2, bit 5 of the EIRV may also be set to 1 for certain storage data check conditions. Additionally, when executing a floating-point FS-instruction, detection of a storage data check may set floating-point status vector bit 16 (Floating-Point Check) to 1.

Exception: Bit 3 of the EIRV indicates that a program exception was detected while a primary PSV was active, or that a channel exception was detected during a channel I/O operation. Bit 4 of the EIRV is also set to 1 for the channel exception.

Detection of a program exception causes the current operation to be to be suspended, suppressed, completed, or terminated, depending on the specific exception. The program information code in the stored PSV indicates the cause of the program exception. Refer to Chapter 3 for a detailed description of program exceptions.

Detection of a channel exception causes the channel I/O operation to be terminated. Channel exceptions are described under "Termination Due to Channel Exception" in Chapter 8.

Channel I/O Check: Bit 4 of the EIRV is set to 1 in conjunction with bits 0-3 and 5 of the EIRV when the indicated system check results from a channel I/O operation. When the indicated system check is not associated with a channel I/O operation, that is, when it results from instruction fetch or execution, or from interruption action, bit 4 is left unchanged.

Internal Control Check: Bit 5 of the EIRV indicates that an equipment malfunction has occurred in the PCE, channel, or, if installed, the floating-point feature. Malfunctioning includes errors on information being processed within the PCE, channel, or floating-point feature, as well as errors associated with the control logic of the PCE, channel, or floating-point feature. If the malfunction occurs in the floating-point feature, a floating-point check is additionally indicated in the floating-point status vector. Bit 4 of the EIRV is also set to 1 when the malfunction results from a channel I/O operation.

Detection of the internal control check causes the current operation to be terminated. The state of the PCE, channel, or floating-point feature after the internal control check is detected depends on the source of the fault and the point in the operation at which the fault is detected.

Internal Data Check: Bits 2 and 5 of the EIRV, together, indicate that a data check was detected in the information fetched from a principal or adjunct register, or from the translation table or translation lock table. Such information is considered invalid. Bit 4 of the EIRV is also set to 1 if the internal data check results from a channel I/O operation.

Except as stated below, detection of an internal data check causes the current operation to be terminated.

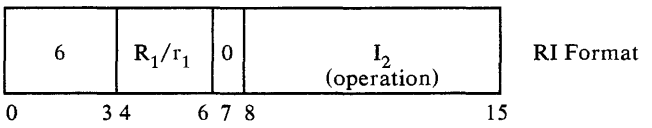
If invalid information is fetched from the translation table or translation lock table as part of the dynamic-address-translation process, the associated storage reference may be attempted. For an instruction-fetch reference, execution of the fetched instruction is suppressed. If invalid information is fetched from a general register and the invalid information represents a storage address, the storage reference using the invalid address may be attempted. For an instruction-fetch

reference, execution of the fetched instruction is suppressed. If invalid information is fetched from a principal or adjunct register as part of a newly introduced PSV or ACV, the interruption action that caused the new PSV and ACV to be introduced is completed, and the instruction fetch associated with the invalid PSV or ACV may be attempted. However, execution of the instruction is suppressed.

Depending on processor model, the prefetching of invalid information from the translation table or the translation lock table (during the dynamic-address-translation process), or from a register, may result in a system-check interruption before the information is actually needed. Further, the interruption may occur even if the information is not actually required (such as instruction prefetch when the prefetched instruction is discarded). When dynamic address translation is not active and invalid information is fetched from the translation table or the translation lock table during execution of the LAT or LATL instruction, only EIRV bit 5 may be set without bit 2.

Instructions for PCE Control

A class of instructions—the PCE-control instructions—is provided to access and modify information in control vectors, as well as to perform other operations necessary for PCE control. All PCE-control instructions are in the RI format, as shown in the following figure:



The 8-bit I_2 field is used as an extension to the operation code to designate the operation to be performed and, where applicable, the control vector used for the operation. If the I_2 field designates an undefined value, an operation exception is indicated.

This class of instructions is also called the control-immediate instructions. The mnemonic KI is used to designate the operation for all instructions in this class. The mnemonic KI and the RI format are also used for the instruction CALL PSV. This instruction is available for general use, however, and is not considered a PCE-control instruction. CALL PSV is described in Chapter 4.

The R_1/r_1 field designates a general-register byte operand or word operand for those operations that transfer information between a control vector and a general register. The operand byte is located in bit positions 16-23 (the upper byte-operand location) of a general register in the primary register set. The word operand is also located in a general register in the primary register set. For operations that do not use a register operand, the R_1/r_1 field is reserved and should contain 0's.

PCE-control instructions used to alter information in control vectors are supervisor-privileged. Thus, instructions that set, reset, write, or perform logical operations on information in control vectors cannot be executed in input/output or application mode. There are two exceptions to this: the instructions SET PROGRAMMED INTERRUPT REQUEST and RESET PROGRAMMED INTERRUPT REQUEST. These two instructions are input/output-privileged and are invalid only in application mode. Operations that read information in control vectors are valid in all program modes.

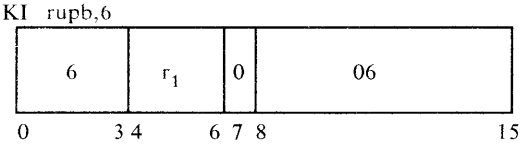
Bit position 7 of the instruction is used both as the low-order bit of the 4-bit R_1/r_1 field and as an extension of the operation code. In the latter case, the bit distinguishes this instruction from the I/O BYTE (immediate) instruction.

The descriptions of the PCE-control instructions and their mnemonics, formats, and operation codes follow.

Note: *The procedure for describing the individual instructions and the symbols used in the instruction formats and the expressions of operations are defined under "Instruction Descriptions" in Chapter 4. The assembler language notation used in the instruction descriptions is explained in Appendix B. The second operand specification in the assembler language statement for each instruction is shown as a decimal number that identifies the PCE-control operation. In the instruction format, the corresponding I_2 field is shown in hexadecimal representation.*

AND WITH PROGRAMMED INTERRUPT REQUEST VECTOR

KI rupb,6



RI Format

Operation

PIRV <— PIRV • (r_1)

Description

The AND of the 8-bit programmed interrupt request vector (PIRV) and the byte at the first-operand location is placed in the programmed interrupt request vector.

The operands are treated as unstructured logical quantities, and the connective AND is applied bit by bit. A bit position in the result is made a 1 if the corresponding bit positions in both operands contain a 1; otherwise, the result bit is made 0.

The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions

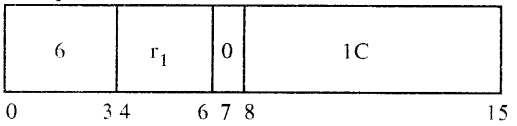
The conditions remain unchanged.

Program Exceptions (Suppression)

Operation (supervisor-privileged operation)

DISPATCH NEW LEVEL

KI rupb,28



RI Format

Operation

If CPL ≠ (r_1<5..7>)
  Then Store current PSV
    LPL <— CPL
    CPL <— (r_1<5..7>)
    Load new PSV/ACV pair indicated by PAV<CPL>
  Else NSI

Description
If the value in the three low-order bit positions at the first-operand location is not equal to the current priority level number, the new priority level designated by the first operand is dispatched; otherwise, no operation is performed.

The current PSV is stored into the permanently-assigned register locations from which it was originally loaded. The last priority level (LPL) vector is then set to the current priority level (CPL) number, and the CPL vector is set to the value of the first operand. Subsequently, a PSV and ACV are loaded from the register locations assigned to the priority level indicated by the new value of the CPL number.

The bit position in the program activation vector associated with the new priority level determines whether the primary or secondary PSV/ACV pair is loaded. If the PAV bit is 0, the primary pair is loaded; if the PAV bit is a 1, the secondary pair is loaded.

The priority-level-dispatching mechanism, including interrupt-request and enabling-disabling information, is bypassed until one instruction, or part of one interruptible instruction, is executed on the new priority level (see "Point of Interruption" in this chapter). Priority level dispatching is resumed thereafter.

If a system-check interruption occurs during this operation because of a channel I/O check, the first instruction to be executed on the designated new priority level may be suppressed without indication to the program. Whether or not the first instruction on the new level is suppressed is unpredictable.

An instruction may not execute on the new priority level if a program-exception condition is encountered. Additionally, if a secondary level is designated, a program exception may cause the primary PSV/ACV pair for the designated level to be activated. Whether or not an instruction is executed on the primary level before priority level dispatching resumes is unpredictable.

Bits 5-7 of the first-operand byte contain the designated priority-level number. Bits 0-3 of the operand are ignored. Bit 4 of the operand is reserved and should be 0. Depending on processor model, a specification exception may occur when bit 4 is not 0.

The first operand is located in bit positions 16-23 of the designated primary general register.

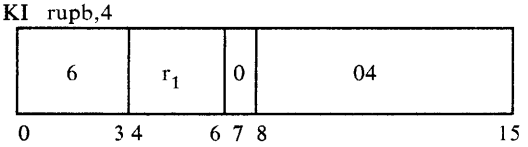
Result Conditions
The condition indicators in the stored PSV remain unchanged.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)
Specification (operand: depending on processor model, bit 4 not 0)

Programming Note
If a system-check interruption occurs while the master mask is 0, DISPATCH NEW LEVEL must be used in order to exit from priority level 0 without altering the master mask.

OR WITH PROGRAMMED INTERRUPT REQUEST VECTOR

KI rupb,4



RI Format

Operation
PIRV <— PIRV v (r_1)

Description
The OR of the 8-bit programmed interrupt request vector (PIRV) and the byte at the first-operand location is placed in the programmed interrupt request vector.

The operands are treated as unstructured logical quantities, and the connective OR is applied bit by bit. A bit position in the result is made a 1 if the corresponding bit position in one or both operands contains a 1; otherwise, the result bit is made 0.

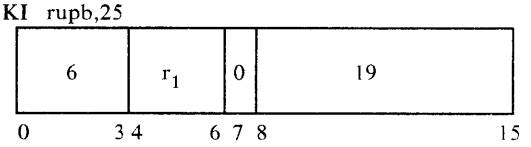
The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)

READ CHANNEL MASK

KI rupb,25



RI Format

Operation
(r_1) <— 0000000 | | CHM

Description
The 1-bit channel mask (CHM) is placed unchanged in the byte at the first-operand location.

The channel-mask bit is placed in bit position 7 of the first-operand byte; bit positions 0-6 are reserved and set to 0's.

The first operand is located in bit positions 16-23 of the designated primary general register.

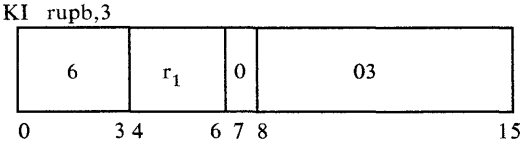
Result Conditions
The conditions remain unchanged.

Program Exceptions
None

Programming Note
The channel mask controls whether channel I/O burst transfers are enabled or disabled (see Chapter 8, "Input/Output Operations").

READ COMMON MASK

KI rupb,3



RI Format

Operation
(r_1) <— CM

Description
The 8-bit common mask (CM) is placed unchanged in the byte at the first-operand location.

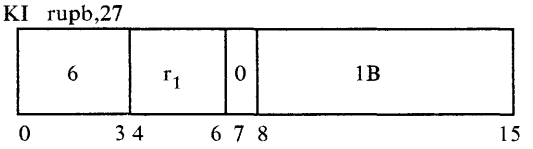
The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

READ CONDITION INDICATORS

KI rupb,27



RI Format

Operation
(r_1) <— 0000 | | Current-PSV<48,49,56,57>

Description
The condition indicators in the current PSV are placed unchanged in the four low-order bit positions of the byte at the first-operand location.

Condition indicators in bit positions 48 and 49 of the current PSV are placed in bit positions 4 and 5 of the first-operand byte; the indicators in bit positions 56 and 57 of the current PSV are placed in bit positions 6 and 7. The four high-order bit positions of the first operand are set to 0's.

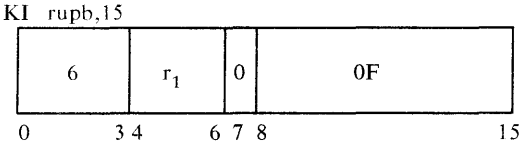
The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

READ CURRENT AND LAST LEVELS

KI rupb,15



RI Format

Operation

(r_1) <— 0| |CPL| |0| |LPL

Description

The current priority level (CPL) and last priority level (LPL) numbers are placed unchanged in the byte at the first-operand location.

The 3-bit CPL number is placed in bit positions 1-3 of the first-operand byte; the LPL number is placed in bit positions 5-7. Bit positions 0 and 4 of the operand are reserved and made 0.

The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions

The conditions remain unchanged.

Program Exceptions

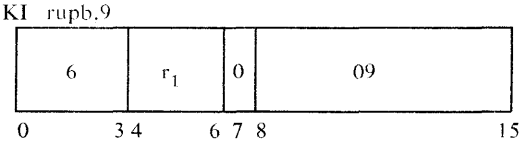
None

Programming Note

The instruction may be used by a level-0 program to determine which priority level was active when a system-check interruption occurred.

READ ERROR INTERRUPT REQUEST VECTOR

KI rupb,9



RI Format

Operation

(r_1) <— EIRV

Description

The 8-bit error interrupt request vector (EIRV) is placed unchanged in the byte at the first-operand location.

The first operand is located in bit positions 16-23 of the designated primary general register.

Bit position 7 of the error interrupt request vector is reserved and is read as 0.

Result Conditions

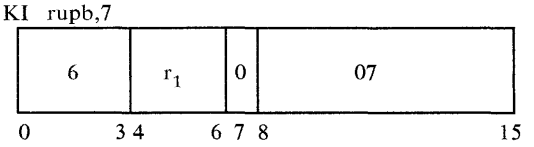
The conditions remain unchanged.

Program Exceptions

None

READ I/O INTERRUPT REQUEST VECTOR

KI rupb,7



RI Format

Operation
(r_1) <— IOIRV

Description
The 8-bit I/O interrupt request vector (IOIRV) is placed unchanged in the byte at the first-operand location.

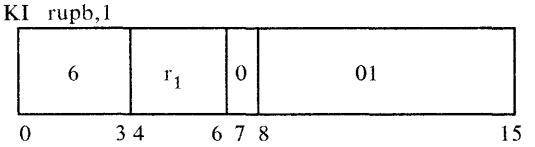
The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

READ MASTER MASK

KI rupb,1



RI Format

Operation
(r_1) <— 0000000 | | MM

Description
The 1-bit master mask (MM) is placed unchanged in the byte at the first-operand location.

The master-mask bit is placed in bit position 7 of the first-operand byte; bit positions 0-6 are reserved and set to 0's.

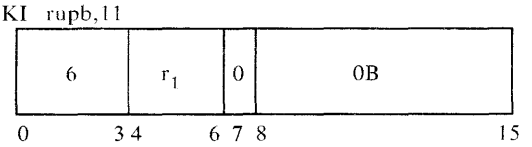
The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

READ PRIMARY REGISTER SET NUMBER

KI rupb,11



RI Format

Operation

(r_1) <— 00| | Current-PSV<58..63>

Description

The 6 bits of the primary-register-set field in the current PSV are placed unchanged in the low-order bit positions of the byte at the first-operand location.

Bit positions 58-63 of the current PSV are placed in bit positions 2-7 of the first-operand byte. Bit positions 0 and 1 of the first operand are reserved and set to 0's.

The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions

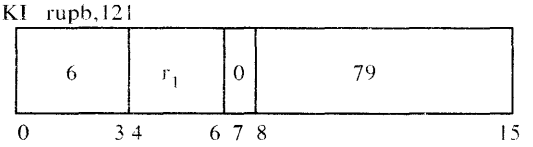
The conditions remain unchanged.

Program Exceptions

None

READ PROGRAM ACTIVATION VECTOR

KI rupb,121



RI Format

Operation

(r_1) <— PAV

Description

The 8-bit program activation vector (PAV) is placed unchanged in the byte at the first-operand location.

The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions

The conditions remain unchanged.

Program Exceptions

None

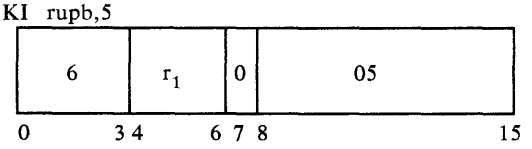
Programming Note

With one exception, this instruction may be used by a level-0 program to determine which of the two PSV/ACV pairs (primary or secondary), associated with the last (interrupted) priority level, was active when a system-check interruption occurred. The exception is when the interrupted program has altered

the PAV bit corresponding to its priority level. In this case the PAV will provide a misleading indication by identifying the opposite PSV as being active when the interruption occurred. (See the programming note under "WRITE PROGRAM ACTIVATION VECTOR.") If the system-check interruption is caused by a program exception, it is not necessary to read the PAV since, by definition, the primary PSV/ACV pair was active.

READ PROGRAMMED INTERRUPT REQUEST VECTOR

KI rupb,5



RI Format

Operation
(r_1) <— PIRV

Description
The 8-bit programmed interrupt request vector (PIRV) is placed unchanged in the byte at the first-operand location.

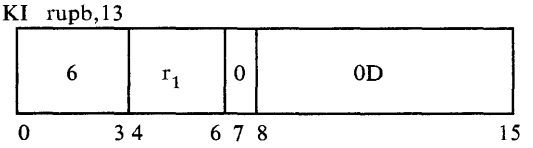
The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

READ SECONDARY REGISTER SET NUMBER

KI rupb,13



RI Format

Operation
(r_1) <— 00| | Current-PSV<50..55>

Description
The 6 bits of the secondary-register-set field in the current PSV are placed unchanged in the low-order bit positions of the byte at the first-operand location.

Bit positions 50-55 of the current PSV are placed in bit positions 2-7 of the first-operand byte. Bit positions 0 and 1 of the first operand are reserved and set to 0's.

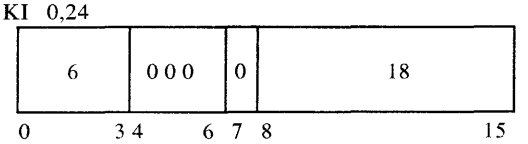
The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

RESET CHANNEL MASK

KI 0,24



RI Format

Operation
CHM <— 0

Description
The 1-bit channel mask (CHM) is made 0 (reset).

Bit positions 4-6 of the instruction are reserved and should contain 0's.

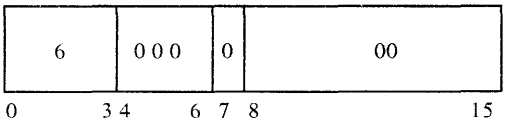
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)

Programming Note
The channel mask controls whether channel I/O burst transfers are enabled or disabled (see Chapter 8, "Input/Output Operations").

RESET MASTER MASK

KI 0,0



RI Format

Operation
MM <— 0

Description
The 1-bit master mask (MM) is made 0 (reset).

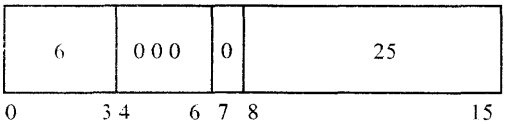
Bit positions 4-6 of the instruction are reserved and should contain 0's.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)

RESET PROGRAMMED INTERRUPT REQUEST

KI 0,37



RI Format

Operation
PIRV<CPL> <— 0

Description
The bit position, corresponding to the number of the current priority level (CPL), in the 8-bit programmed interrupt request vector (PIRV) is made 0 (reset).

Bit positions 4-6 of the instruction are reserved and should contain 0's.

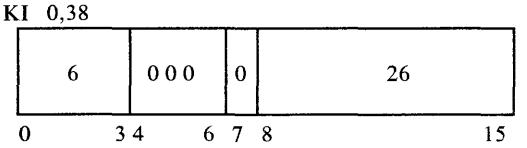
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (I/O-privileged operation)

Programming Note
This instruction may be used to remove the programmed interrupt request for the current priority level without first determining which level is active. This function is useful, for example, in a common exit routine that is executed on two or more priority levels.

SET CHANNEL MASK

KI 0,38



Operation
CHM <— 1

Description
The 1-bit channel mask (CHM) is made a 1 (set).

Bit positions 4-6 of the instruction are reserved and should contain 0's.

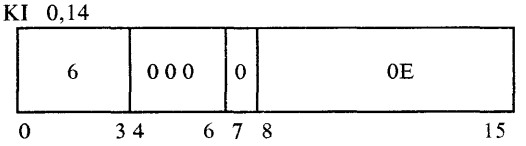
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)

Programming Note
The channel mask controls whether channel I/O burst transfers are enabled or disabled (see Chapter 8, "Input/Output Operations").

SET MASTER MASK

KI 0,14



Operation
MM <— 1

Description
The 1-bit master mask (MM) is made a 1 (set).

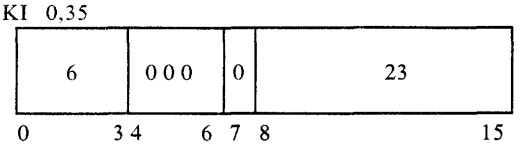
Bit positions 4-6 of the instruction are reserved and should contain 0's.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)

SET PROGRAMMED INTERRUPT REQUEST

KI 0,35



RI Format

Operation
PIRV<CPL> <— 1

Description
The bit position, corresponding to the number of the current priority level (CPL), in the 8-bit programmed interrupt request vector (PIRV) is made 1 (set).

Bit positions 4-6 of the instruction are reserved and should contain 0's.

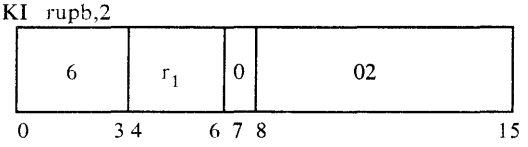
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (I/O-privileged operation)

Programming Note
This instruction may be used to generate a programmed interrupt request for the current priority level without first determining which level is active. This function is useful, for example, in a common I/O interruption-handling routine that is executed on two or more priority levels. The interruption-handling routine may execute this instruction, then execute an I/O instruction that removes the I/O interrupt request (allowing a subsequent I/O interrupt request to be indicated in the IOIRV), and then service the I/O device.

WRITE COMMON MASK

KI rupb,2



RI Format

Operation
CM <— (r_1)

Description
The byte at the first-operand location is placed unchanged in the 8-bit common mask (CM).

The first operand is located in bit positions 16-23 of the designated primary general register.

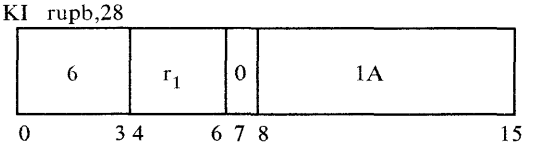
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)

Programming Note
For processor models that can disable priority level 0 for system-check interrupt requests, a system check detected while priority level 0 is disabled can result in loss of system integrity. Therefore, during normal system operation, the program should not write a 0 in bit position 0 of the common mask, thereby disabling level 0.

WRITE CONDITION INDICATORS

KI rupb,28



RI Format

Operation
Current-PSV<48,49,56,57> <— (r_1<4..7>)

Description
The 4 low-order bits of the byte at the first-operand location are placed unchanged in the condition-indicator bit positions of the current PSV.

Bits 4 and 5 of the first-operand byte are placed in condition-indicator bit positions 48 and 49 of the current PSV; bits 6 and 7 are placed in indicator bit positions 56 and 57 of the current PSV. Bit positions 0-3 of the first operand are ignored.

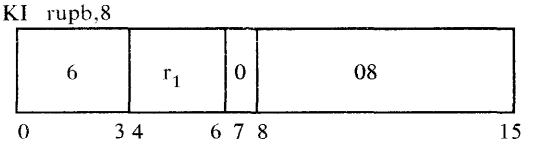
The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions
The states of the result conditions are determined by the condition indicators in the current PSV at the completion of this operation.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)

WRITE ERROR INTERRUPT REQUEST VECTOR

KI rupb,8



RI Format

Operation
EIRV <— (r_1)

Description
The byte at the first-operand location is placed unchanged in the 8-bit error interrupt request vector (EIRV).

The first operand is located in bit positions 16-23 of the designated primary general register.

Bit 7 of the error interrupt request vector is reserved; therefore, the corresponding low-order bit position of the first-operand byte should contain a 0. Depending on processor model, a specification exception may occur when bit 7 is not 0.

The conditions remain unchanged.

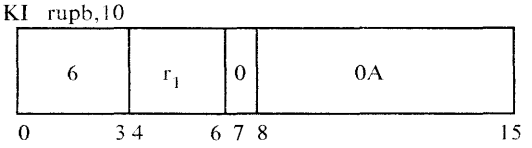
Result Conditions
The condition remains unchanged

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)
Specification (operand: depending on processor model, bit 7 not 0)

Programming Note
This instruction must be used to clear the EIRV. It may also be used to set a bit in the EIRV to 1. When any one of bits 0-5 is set to 1, an interrupt request for priority level 0 is generated.

WRITE PRIMARY REGISTER SET NUMBER

KI rupb,10



RI Format

Operation
Current-PSV<58..63> <— (r_1<2..7>)

Description
The 6 low-order bits of the byte at the first-operand location are placed unchanged in the primary-register-set field of the current PSV.

Bits 2-7 of the first-operand byte are placed in bit positions 58-63 of the current PSV. Bit positions 0 and 1 of the first-operand byte are reserved and should contain 0's. Depending on processor model, a specification exception may occur when bit positions 0 and 1 are not 0.

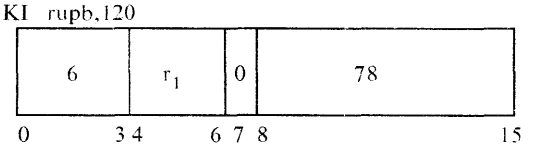
The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)
Specification (operand: depending on processor model, bits 0 and 1 not 0)

WRITE PROGRAM ACTIVATION VECTOR

KI rupb,120



RI Format

Operation
PAV <— (r_1)

Description
The byte at the first-operand location is placed unchanged in the 8-bit program activation vector (PAV).

The first operand is located in bit positions 16-23 of the designated primary general register. Depending on processor model, a specification exception may occur if the r_1 contents causes an attempt to change the state of the PAV bit that corresponds to the current priority level.

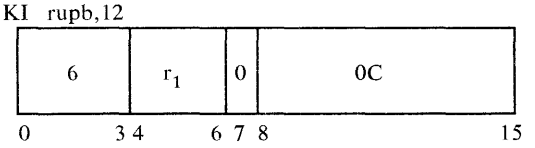
Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)
Specification (operand: depending on processor model, state change specified for current priority level)

Programming Note
The program should not use this instruction to change the state of the PAV bit corresponding to the current priority level. This action is prevented by hardware in some processor models. In models that allow this bit to be changed, program execution at the current level still may be prematurely concluded. This will occur, for example, if a higher priority level is given control because of an interruption before the program removes the interrupt request for the current level. When program execution at the current level is resumed, the opposite PSV/ACV pair is made active. Further, if the interruption is caused by a system check, the PAV provides a misleading indication by identifying the opposite PSV as being active when the interruption occurred.

WRITE SECONDARY REGISTER SET NUMBER

KI rupb,12



RI Format

Operation
Current-PSV<50..55> <— (r_1<2..7>)

Description
The 6 low-order bits of the byte at the first-operand location are placed unchanged in the secondary-register-set field of the current PSV.

Bits 2-7 of the first-operand byte are placed in bit positions 50-55 of the current PSV. Bit positions 0 and 1 of the first-operand byte are reserved and should contain 0's. Depending on processor model, a specification exception may occur when bit positions 0 and 1 are not 0.

The first operand is located in bit positions 16-23 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

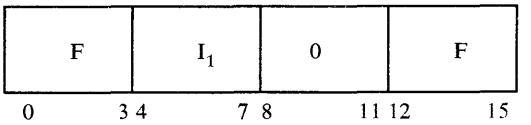
Program Exceptions (Suppression)
Operation (supervisor-privileged operation)
Specification (operand: depending on processor model, bits 0 and 1 not 0)

Instruction for Direct Control

The following instruction is provided for the direct control of system-control facilities.

CONTROL DIRECT OUT

KDO i4



RR Format

Operation
SCF <— I_1

Description
The 4-bit I_1 field is made available to system-control facilities (SCF). The information in the I_1 field is used to perform built-in system functions. The processor description manual for the specific processor model describes these functions.

Bit positions 8-11 of the instruction must be all 0's; otherwise, an operation exception is indicated.

Result Conditions
The conditions remain unchanged.

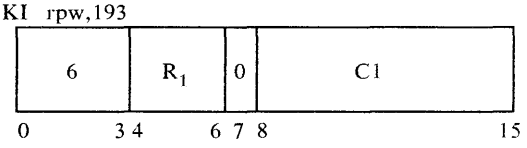
Program Exceptions (Suppression)
Operation (bits 8-11 of instructions not all 0's)

Instructions for Diagnostic Control Vector

The following instructions are provided for system maintenance and initialization functions in certain processor models.

READ DIAGNOSTIC CONTROL VECTOR

KI rpw,193



RI Format

Operation
(R_1) <— DCV

Description
The diagnostic control vector (DCV) is placed unchanged in the word at the first-operand location.

The DCV implemented in certain processor models provides system maintenance and initialization functions. Application and supervisory programs should not use the DCV.

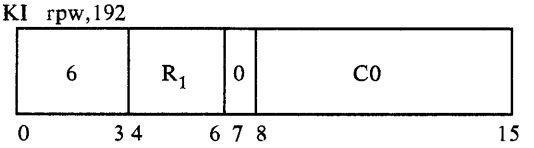
The first operand is located in bit positions 0-31 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

Program Exceptions
None

WRITE DIAGNOSTIC CONTROL VECTOR

KI rpw,192



RI Format

Operation
DCV <— (R_1)

Description
The word at the first-operand location is placed in the diagnostic control vector (DCV).

The DCV implemented in certain processor models provides system maintenance and initialization functions. Application and supervisory programs should not use the DCV.

The first operand is located in bit positions 0-31 of the designated primary general register.

Result Conditions
The conditions remain unchanged.

Program Exceptions (Suppression)
Operation (supervisor-privileged operation)

Chapter 10. Dual-Mode Processing

This chapter discusses dual-mode processing as it occurs in certain 8100 processor models. It describes the differences between single-PCE processors and dual-PCE processors.

Logical Structure

The logical structure of an 8100 Information System dual-PCE processor consists of shared main storage, two PCEs, either one or two channels, input/output devices attached to the channels through adapters, and an Interrupt Control Element (ICE) for communication between the PCEs. This logical structure is shown in Figure 10-1.

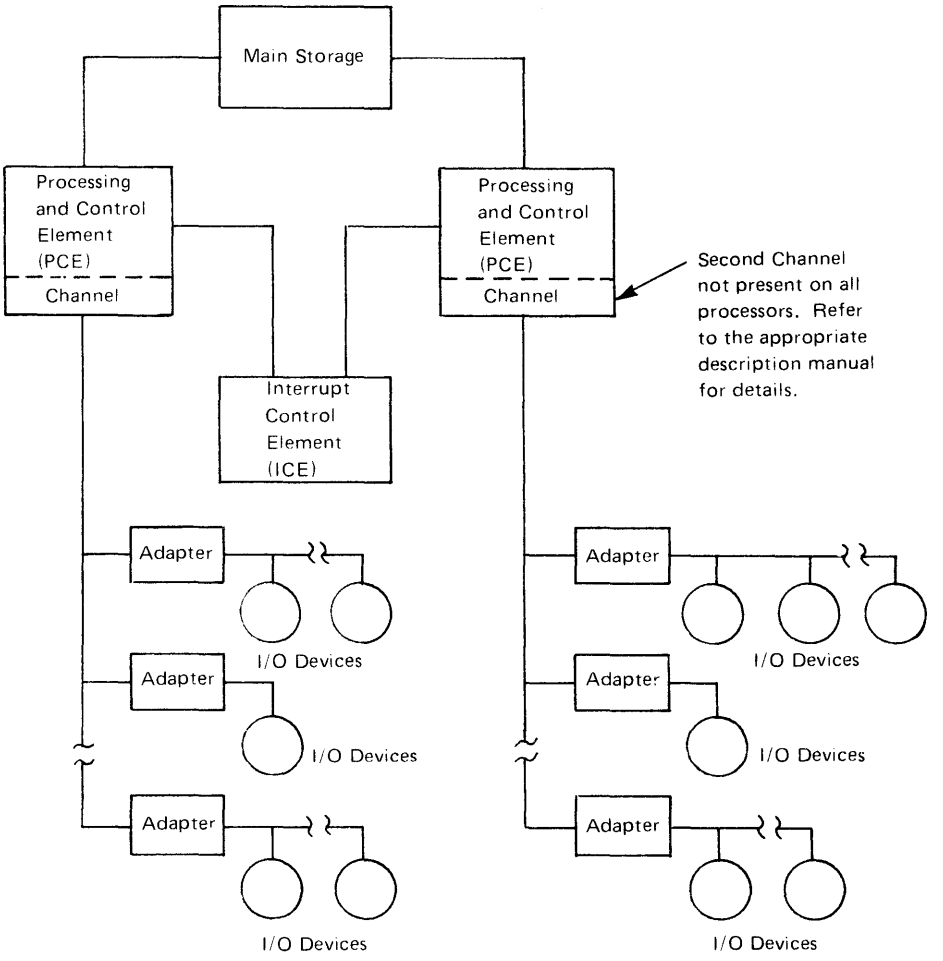


Figure 10-1. Logical Structure of Dual-PCE Processors

Depending on dual-PCE processor model, additional system control facilities provide functions that enhance system availability, such as the capability to reconfigure around a failing system component and to connect an I/O device to both channels for greater device availability. These system control facilities are described in the applicable processor description manuals.

The logical structure of both PCEs is identical. Each individual PCE provides independent sequencing and processing controls for instruction execution, interrupt action, dynamic address transformation, and other control and processing functions.

The shared main storage facility permits both PCEs to address main storage locations and therefore execute any of the programs resident in main storage. In addition to a shared main storage, both PCEs have access to private translation-table entries which are mapped to unique storage blocks. These entries are accessed and updated on an independent basis and are accessible only by the owning PCE. Additionally, both PCEs have access to common translation-table entries which are mapped to shared storage blocks.

On processor models that implement separation protection, both PCEs have access to private and common translation-lock-table entries, corresponding respectively to the private and common regions of the translation table.

Independent principal and adjunct register groups are provided for each PCE. Additionally, each PCE has independent, unique vectors that are permanently assigned to hold control information. Register groups and control vectors are addressed only by their respective PCE.

Floating-point registers, each 64 bits long, are available when the floating-point feature is installed. One floating-point feature is allowed on dual-PCE processors and is restricted to a specific PCE.

The ICE enables communication between the two PCEs during normal system operation. For a detailed description of the ICE, refer to either the *IBM 8140 Processor Description*, GA27-2880 or the *IBM 8150 Processor Description*, GA23-0122.

The normal mode of operation is dual mode, with both PCEs functional. The processor can also function in single mode, with only one PCE functional.

Storage and Registers

Both PCEs in dual-PCE processors have access to main storage. Both data and programs must be resident in main storage before they can be processed. Since main storage is common, both PCEs are capable of executing any of the programs resident in storage. An address used by the program or in a channel I/O operation is referred to as a logical address. The formation of real addresses from logical addresses is as described in Chapter 7. The same real address is used by either PCE to refer to a specific main storage location.

Each PCE can address information in its associated register sets and control vectors. One PCE cannot address the register sets or control vectors of the other PCE.

The organization and addressing of main storage and registers is described in Chapter 2.

Program Execution

The instructions executed by the PCEs in dual-PCE processors are identical to those executed in single-PCE processors. The PCE follows the same rules for register operand specification, immediate operand specification, storage operand specification, and address generation as described in Chapter 3.

Execution

The execution of instructions by each PCE is identical to the description in Chapter 3 with the following exception: In dual-PCE processors, two PCEs may be executing programs concurrently. As such, each PCE may be referencing storage to fetch instructions which occur in the flow of a program, to fetch and/or store data required to complete the execution of an instruction, or to transfer data during CHIO operations. The operation of an instruction or interruptible unit of operation for a single-operand reference is synchronized with storage. Thus, interference due to concurrent data or instruction references is prevented during the span of a single-operand reference for operands up to a fullword in length.

The Test and Set (TS) instruction is a special case. The nature of the function and operation of this instruction provides two non-interruptible references to an identical storage location. Therefore, instruction storage synchronization is guaranteed for the entire span of the TS instruction.

The operand length for most instructions is either a byte, a halfword, or a fullword. Such operands are always assumed to occur on an integral boundary. However, the operand lengths for the long-precision floating point and the LHQ/STHQ instructions are greater than a fullword. Therefore, if instruction storage synchronization is required for any instruction containing operands greater than a fullword in length, such synchronization must be explicitly provided by programming convention, using, for example, the TEST and SET instruction.

When CHIO operations occur concurrently with references to main storage by a PCE, storage is synchronized on an integral word boundary basis provided the channel storage references are within a single burst of halfword transfers. Storage synchronization for instruction and data references does not affect the interleaving of storage accesses for CHIO data bursts consisting of multiple halfwords which are greater than a word in length. Storage synchronization is not provided for CHIO byte transfers.

For the class of interruptible instructions (MVS, MVHS, CLS, and CLHS), instruction storage synchronization for an operand reference is only guaranteed within a unit of operation. Synchronization is not ensured for the period between two operand references within the unit of operation. Similarly, for the floating-point instruction RFS (READ FLOATING-POINT STATUS VECTOR), synchronization is only guaranteed for the operand references and not for the period between the operand references within the instruction.

Depending on processor model, more than one unit of operation may be executed between points in the operation at which an interruption is allowed. The program must not be dependent on the validity of instruction storage synchronization for the period between allowable interruptions. Synchronization is guaranteed only for the operand(s) within a single unit of operation. The operands within a unit of operation may be as short as one byte each.

Sequence of Execution

Conceptually, both PCEs in a dual-PCE processor execute instructions serially, with the execution of one instruction preceding the execution of the following instruction without interaction from the other PCE.

The conceptual sequential execution of instructions within a PCE is preserved when either (1) a new PSV is introduced, (2) an entry is stored by the executing PCE into the translation table or the translation lock table when dynamic address translation is active, or (3) a BRANCH or JUMP operation is executed which results in the instruction address being replaced with the branch or jump address.

Interaction with the translation table is such that the conceptual sequential order of execution is preserved within a precision of two instruction halfwords between two PCEs.

Programming Notes:

- Dynamic modification of the entries in both the common translation table and common translation lock table must be carefully synchronized. Conceptual sequential instruction execution following a modification of the common translation table or the common translation lock table is ensured for the PCE initiating the change. However, for processor models employing the concept of instruction prefetching, the effect of such a change could be delayed for up to two instruction halfwords on the other PCE. To ensure synchronization, the TEST AND SET (TS) instruction can be used when both PCEs are operating. To discard the prefetched instructions of a PCE in the wait state, a halfword unconditional BRANCH or JUMP operation can be executed as the first instruction when the wait state is ended.

- The TEST and SET (TS) instruction can be used for controlled sharing of a common storage area by two or more programs within or across PCEs. This controlled sharing can be achieved by establishing a programming convention. For example, an all-0's value in the byte located at the second operand address of the TS instruction indicates that the common area is available, and an all-1's value means that the area is being used. Each using program then must examine this byte by means of TEST AND SET before making access to the common area. Because TEST AND SET does not permit an interruption to occur between the moment of fetching (for testing) and the moment of storing (setting) and because of storage synchronization, the possibility of a second program testing this byte before the first program is able to set it is avoided.

Program Exceptions

Exceptions resulting from the execution of a program, including the improper specification or use of instructions and data, cause a program-exception interruption. The types of endings and program exception conditions are as described in Chapter 3 and are independent of the other PCE in the configuration.

General Instructions

The general instructions and their mnemonics, formats, and operations are as described in Chapter 4. From a user preception, the operation of the general instructions remains unchanged.

Floating-Point Instructions

The optional floating-point feature attaches to only one PCE and uses a unique set of instructions which execute only on that PCE. The floating-point instructions and their mnemonics, formats, and operations are as described in Chapter 5.

Register Organization

Each PCE has its own unique registers. One PCE cannot address the registers of the other PCE.

Two groups of registers are always provided by each PCE: the principal register group and the adjunct register group. When the optional floating-point feature is present, a third group of registers, the floating-point register group, is provided. The floating-point register group can be accessed only by the PCE that has the feature.

Register organization, addressing register contents and register-indirect instructions, their mnemonics, formats, and operations are described in Chapter 6.

Dynamic Address Relocation and Translation

The addressing arrangement of the PCEs is based on a logical separation of the addresses used by a program and a channel from the addresses assigned to the physical locations in main storage. During every storage reference by either PCE in the system, the logical address supplied by a program or a channel is mapped into the PCE address space. This mapping process is called *dynamic address relocation*, and is described in Chapter 7.

When dynamic address translation is active, a single logical copy each of the translation table and translation lock table (for processor models that implement separation protection) is accessed by each respective PCE during the translation of logical addresses to real storage locations. Both tables contain four private entries that are available only to a specific PCE and allow corresponding entries of the translation table to translate addresses to different storage blocks and corresponding entries of the translation lock table to contain different translation locks. The private area for each table is designated in terms of logical storage addresses and displaces the corresponding entries in both the common translation table and the common translation lock table. The logical address of the private area for both tables starts at 0; the remaining entries in the translation table and the translation lock table are common and are available to either PCE.

The dynamic address translation mechanism includes functions for protection against unauthorized store operations, instruction execution, or references of any type to main storage.

The process of dynamic address translation, storage access protection, separation protection, and the mnemonics, formats, and operations of the instructions which manipulate the translation-table and the translation-lock-table entries are described in Chapter 7.

Input/Output Operations

Input/output devices attach to a PCE through a channel. One PCE cannot access the channel attached to the other PCE. However, depending on dual-PCE processor model, an I/O device can be made accessible to either PCE by connecting it to each PCE's channel by means of system control facilities. These facilities are described in the applicable processor description manuals.

Programmed I/O and channel I/O operations and the mnemonics, formats, and operations of the instructions available are described in Chapter 8.

PCE Control

Each PCE in dual-PCE processors contains unique control information used in PCE management. This control information is accessible to the owning PCE. One PCE cannot access the control information of the other PCE. The operation of each individual PCE is described in Chapter 9.

The Interrupt Control Element (ICE) enables communication between the two PCEs during normal system operation. Specific functions provided by the ICE are described in the *IBM 8140 Processor Description*, GA27-2880, and the *IBM 8150 Processor Description*, GA23-1022.

The master mask operations of the PCEs are interlocked in such a manner that the two PCEs cannot operate simultaneously with their respective mask bits in the reset (set to 0) state. When one PCE is operating with its master mask reset and the other PCE executes a Reset Master Mask instruction (KI 0), the latter's master mask is reset but execution by the PCE is suspended. All interrupt requests at the suspended PCE, with the exception (depending on processor model) of a system-check interrupt request for priority level 0 during a CHIO operation, are held pending. If a PCE is suspended, CHIO operations for that PCE continue unless a subsequent channel error is detected. If an error is detected, the channel operation is terminated and, depending on processor model, either the interrupt request is held pending or an interruption to level 0 occurs. In either case, instruction execution remains suspended.

When the PCE that was executing with its master mask reset again sets its master mask (KI 14), the PCE which was suspended resumes operation. Execution continues with the next sequential instruction following the instruction which originally reset the master mask, or an interruption to a new priority level occurs based on the contents of the interrupt request vectors and masks. Any EIRV conditions which were pending during the time operation was suspended are now indicated to the corresponding PCE.

In normal (nonsuspended) operation, a channel becomes disabled when EIRV bits 0-3 or 5 are set to 1 in the corresponding PCE. When EIRV bit 4 only is set to 1 with a WRITE ERROR INTERRUPT REQUEST VECTOR instruction, the PCE's channel normally is not disabled. However, depending on processor model, a PCE that has set only EIRV bit 4 may have its channel disabled when instruction execution is suspended.

Master mask synchronization allows serialization of a sequence of code which is shared between PCEs. Once the master mask is reset, it should not be set again until execution of the code sequence is completed.

Whenever one active (nonsuspended) PCE detects a system-check condition, the other PCE receives a signal that a system-check condition has been detected in the configuration. A detailed description of system-check signaling is contained in the *IBM 8140 Processor Description* and the *IBM 8150 Processor Description*.

The instructions for PCE control and their mnemonics, formats, and operations are described in Chapter 9, with the exception of RESET MASTER MASK (KI 0), which is synchronized between the PCEs and SET MASTER MASK (KI 14), thus allowing for the resumption of execution of a suspended PCE (Master Mask reset).

Appendix A. Lists of Instructions

The following three lists are of instructions arranged by name, mnemonic, and type. The symbols used to label the characteristics columns mean:

F	Instruction format
T	Instruction type
C	Indication of whether the condition indicators are set
P	Indication of privilege classification of the instruction

The symbols in the characteristics columns mean:

Ac	Access exception
Ad	Address exception
C	Condition indicators are set
CC	PCE-control instruction
Eo	Exponent-overflow exception
Eu	Exponent-underflow exception
Fd	Floating-point divide exception
FF	FF instruction format
Fp	Floating-point privilege exception
FP	Floating-point feature instruction
Fs	Floating-point specification exception
FS	FS instruction format
G	General instruction
I	I/O-privileged instruction
IO	Input/Output instruction
J	J instruction format
Op	Operation exception
Ov	Fixed-point overflow exception
Re	Register-indirect exception
RI	RI instruction format
RN	Register-indirect instruction
RR	RR instruction format
RRL	RR-Long instruction format
RS	RS instruction format
RSL	RS-Long instruction format
S	Supervisor-privileged instruction
Se	Separation exception
Si	Significance exception
Sp	Specification exception
TL	Translation-lock-table instruction
TT	Translation-table instruction

The symbols representing the operand specifications are defined in Appendix B.

Refer to the instruction descriptions in the body of this manual for the types of endings associated with program exceptions recognized during instruction execution.

Instructions Arranged by Name

Name	Mnemonic	Operands	Characteristics F	T	C	P	Exceptions
ADD (byte, register)	AR	rpb,rpb rsb,rsb	RR	G	C		
ADD (byte, register-immediate)	ARI	rpb,i8s	RI	G	C		
ADD (halfword, register)	AHR	rh,rh	RR	G	C		
ADD (halfword, register-immediate)	AHRI	rh,i4	RR	G	C		
ADD NORMALIZED	AF	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eu Eo Si
ADD NORMALIZED (register)	AFR	f,f	FF	FP	C		Op Eu Eo Si
ADD UNNORMALIZED	AU	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eo Si
ADD UNNORMALIZED (register)	AUR	f,f	FF	FP	C		Op Eo Si
ADD WITH CARRY (byte, register)	AYR	rpb,rpb rsb,rsb	RR	G	C		
ADD WITH CARRY (halfword, register)	AYHR	rh,rh	RR	G	C		
ADD WITH CARRY (halfword, register, extended)	AYHRE	ruh,ruh	RR	G	C		
AND (byte, register)	NR	rpb,rpb rsb,rsb	RR	G	C		
AND (byte, register-immediate)	NRI	rpb,i8	RI	G	C		
AND (halfword, register)	NHR	rh,rh	RR	G	C		
AND WITH PROGRAMMED INTERRUPT REQUEST VECTOR	KI	rupb,6	RI	CC		S	Op
BRANCH AND LINK	BAL	ra,dh16s(ra)	RSL	G			Sp Ac Se Ad
BRANCH AND LINK (register)	BALR	ra,ra	RR	G			Sp Ac Se Ad
BRANCH ON CONDITION	BC	m4,dh16s(ra)	RSL	G			Sp Ac Se Ad
BRANCH ON CONDITION (register)	BCR	m4,ra	RR	G			Sp Ac Se Ad
BRANCH ON COUNT (byte, register)	BCTR	rpb,ra	RR	G			Sp Ac Se Ad
BRANCH ON INDEX (byte)	BNX	rpb,ra	RR	G			Sp Ac Se Ad
CALL PSV	KI	0,127	RI	G			
COMPARE	CF	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad
COMPARE (byte, register)	CR	rpb,rpb rsb,rsb	RR	G	C		
COMPARE (halfword, register)	CHR	rh,rh	RR	G	C		
COMPARE (register)	CFR	f,f	FF	FP	C		Op
COMPARE LOGICAL (bytes, storage)	CLS	ra,ra,rh[1]	RRL	G	C		Sp Ac Op[5] Se Ad
COMPARE LOGICAL (halfwords, storage)	CLHS	ra,ra,rh[1]	RRL	G	C		Sp Ac Op[5] Se Ad
COMPARE WITH CARRY (halfword, register, extended)	CYHRE	ruh,ruh	RR	G	C		
CONTROL DIRECT OUT	KDO	i4	RR	CC			Op
COUNT LEADING ZEROS (halfword)	CTLZ	rh,rh	RR	G	C		
DISPATCH NEW LEVEL	KI	rupb,28	RI	CC		S	Sp[5] Op
DIVIDE	DF	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad Eu Eo Fd
DIVIDE (halfword, register)	DHR	rh[2],rh	RRL	G			Op Ov
DIVIDE (register)	DFR	f,f	FF	FP			Op Eu Eo Fd
EXCLUSIVE OR (byte, register)	XR	rpb,rpb rsb,rsb	RR	G	C		
EXCLUSIVE OR (byte, register-immediate)	XRI	rpb,i8	RI	G	C		
EXCLUSIVE OR (halfword, register)	XHR	rh,rh	RR	G	C		
INPUT/OUTPUT (byte)	IO	rpb,rh	RR	IO	C	I	Op
INPUT/OUTPUT (byte, immediate)	IOI	rlpb,i8	RI	IO	C	I	Op
INPUT/OUTPUT (halfword)	IOH	rh,rh	RR	IO	C	I	Op
JUMP ON BIT ZERO (halfword)	JBZ	n4,dh7s	J	G			Sp Ac Se Ad
JUMP ON CONDITION	JC	m4,dh7s	J	G			Sp Ac Se Ad
LOAD	LF	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad
LOAD (byte)	L	rpb,db16s(ra)	RSL	G			Sp Ac Se Ad
LOAD (byte, register)	LR	rb,rb	RR	G			

Instructions Arranged by Name (continued)

Name	Mnemonic	Operands	Characteristics F	T	C	P	Exceptions
LOAD (byte, register-immediate)	LRI	rpb,i8	RI	G			
LOAD (byte, register-indirect)	LRN	rpb,ra[3]	RR	RN		S	Op Re
LOAD (byte, with index)	LN	rpb,ra	RR	G			Sp Ac Se Ad
LOAD (byte, with index decremented)	LND	rpb,ra	RR	G			Sp Ac Se Ad
LOAD (byte, with index incremented)	LNI	rpb,ra	RR	G			Sp Ac Se Ad
LOAD (halfword)	LH	rh,db16s(ra)	RSL	G			Sp Ac Se Ad
LOAD (halfword, register)	LHR	rh,rh	RR	G			
LOAD (halfword, register indirect)	LHRN	rh,ra[3]	RR	RN		S	Op Re
LOAD (halfword, register, lower half from upper)	LHRLU	rh,ruh	RR	G			
LOAD (halfword, register, upper half)	LHRU	ruh,ruh	RR	G			
LOAD (halfword, register, upper half from lower)	LHRUL	ruh,rh	RR	G			
LOAD (halfword, short form)	LHS	rh,dh5(ra[4])	RS	G			Sp Ac Se Ad
LOAD (halfword, with index)	LHN	rh,ra	RR	G			Sp Ac Se Ad
LOAD (halfword, with index decremented)	LHND	rh,ra	RR	G			Sp Ac Se Ad
LOAD (halfword, with index) incremented)	LHNI	rh,ra	RR	G			Sp Ac Se Ad
LOAD (halfwords, quadrant)	LHQ	q2,ra	RR	G			Sp Ac Op Se Ad
LOAD (register)	LFR	f,f	FF	FP			Op
LOAD (word)	LW	rw,db16s(ra)	RSL	G			Sp Ac Se Ad
LOAD ADDRESS	LA	ra,db16s(ra)	RSL	G			
LOAD AND TEST (register)	LTFR	f,f	FF	FP	C		Op
LOAD COMPLEMENT (register)	LCFR	f,f	FF	FP	C		Op
LOAD FROM ADDRESS TRANSLATION TABLE	LAT	rw,ra	RRL	TT		S	Sp Op
LOAD FROM ADDRESS TRANSLATION LOCK TABLE	LATL	rh,ra	RRL	TL		S	Sp Op
LOAD NEGATIVE (register)	LNFR	f,f	FF	FP	C		Op
LOAD POSITIVE (register)	LPFR	f,f	FF	FP	C		Op
LOAD ROUNDED (register)	LRFR	f,f	FF	FP			Op Eo
MOVE (bytes, storage)	MVS	ra,ra,rh[1]	RRL	G			Sp Ac Op[5] Se Ad
MOVE (halfwords, storage)	MVHS	ra,ra,rh[1]	RRL	G			Sp Ac Op[5] Se Ad
MULTIPLY	MF	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad Eu Eo
MULTIPLY (halfword, register)	MHR	rh,rh	RRL	G			Op
MULTIPLY (register)	MFR	f,f	FF	FP			Op Eu Eo
OR (byte, register)	OR	rpb,rpb rsb,rsb	RR	G	C		
OR (byte, register-immediate)	ORI	rpb,i8	RI	G	C		
OR (halfword, register)	OHR	rh,rh	RR	G	C		
OR WITH PROGRAMMED INTERRUPT REQUEST VECTOR	KI	rupb,4	RI	CC		S	Op
PROGRAM EXCEPTION	PC		RR	G			Op
READ CHANNEL MASK	KI	rupb,25	RI	CC			
READ COMMON MASK	KI	rupb,3	RI	CC			
READ CONDITION INDICATORS	KI	rupb,27	RI	CC			
READ CURRENT AND LAST LEVELS	KI	rupb,15	RI	CC			
READ DCV	KI	rpw,193	RI				
READ ERROR INTERRUPT REQUEST VECTOR	KI	rupb,9	RI	CC			
READ FLOATING-POINT CONTROL	RFC	dw14s(ra)	FS	FP			Sp Ac Op Se Ad
READ FLOATING-POINT STATUS VECTOR	RFS	dw14s(ra)	FS	FP			Sp Ac Op Se Ad Fs
READ I/O INTERRUPT REQUEST VECTOR	KI	rupb,7	RI	CC			
READ MASTER MASK	KI	rupb,1	RI	CC			
READ PRIMARY REGISTER SET NUMBER	KI	rupb,11	RI	CC			
READ PROGRAM ACTIVATION VECTOR	KI	rupb,121	RI	CC			

Instructions Arranged by Name (continued)

Name	Mnemonic	Operands	Characteristics F	T	C	P	Exceptions
READ PROGRAMMED INTERRUPT REQUEST VECTOR	KI	rupb,5	RI	CC			
READ SECONDARY REGISTER SET NUMBER	KI	rupb,13	RI	CC			
RESET CHANNEL MASK	KI	0,24	RI	CC		S	Op
RESET MASTER MASK	KI	0,0	RI	CC		S	Op
RESET PROGRAMMED INTERRUPT REQUEST	KI	0,37	RI	CC		I	Op
ROTATE LEFT (byte)	RL	rb,c3	RR	G	C		
ROTATE LEFT (halfword)	RLH	rh,c4	RR	G	C		
SET CHANNEL MASK	KI	0,38	RI	CC		S	Op
SET MASTER MASK	KI	0,14	RI	CC		S	Op
SET OVERFLOW MASK	SFOM	m1	FF	FP			Op
SET PRECISION MODE	SFPM	m1	FF	FP			Op
SET PROGRAMMED INTERRUPT REQUEST	KI	0,35	RI	CC		I	Op
SET SIGNIFICANCE MASK	SFSM	m1	FF	FP			Op
SET UNDERFLOW MASK	SFUM	m1	FF	FP			Op
SHIFT LEFT (byte, logical)	SLL	rb,c3	RR	G	C		Op
SHIFT LEFT (halfword, logical)	SLHL	rh,c4	RR	G	C		
STORE	STF	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad
STORE (byte)	ST	rpb,db16s(ra)	RSL	G			Sp Ac Se Ad
STORE (byte, register-indirect)	STRN	rpb,ra[3]	RR	RN		S	Op Re
STORE (byte, with index)	STN	rpb,ra	RR	G			Sp Ac Se Ad
STORE (byte, with index decremented)	STND	rpb,ra	RR	G			Sp Ac Se Ad
STORE (byte, with index incremented)	STNI	rpb,ra	RR	G			Sp Ac Se Ad
STORE (halfword)	STH	rh,db16s(ra)	RSL	G			Sp Ac Se Ad
STORE (halfword, register-indirect)	STHRN	rh,ra[3]	RR	RN		S	Op Re
STORE (halfword, short form)	STHS	rh,dh5(ra[4])	RS	G			Sp Ac Se Ad
STORE (halfword, with index)	STHN	rh,ra	RR	G			Sp Ac Se Ad
STORE (halfword, with index decremented)	STHND	rh,ra	RR	G			Sp Ac Se Ad
STORE (halfword, with index incremented)	STHNI	rh,ra	RR	G			Sp Ac Se Ad
STORE (halfwords, quadrant)	STHQ	q2,ra	RR	G			Sp Ac Op Se Ad
STORE (word)	STW	rw,db16s(ra)	RSL	G			Sp Ac Se Ad
STORE TO ADDRESS TRANSLATION TABLE	STAT	rw,ra	RRL	TT		S	Sp Op
STORE TO ADDRESS TRANSLATION LOCK TABLE	STATL	rh,ra	RRL	TL		S	Sp Op
SUBTRACT (byte, register)	SR	rpb,rpb rsb,rsb	RR	G	C		
SUBTRACT (halfword, register)	SHR	rh,rh	RR	G	C		
SUBTRACT (halfword, register-immediate)	SHRI	rh,i4	RR	G	C		
SUBTRACT NORMALIZED	SF	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eu Eo Si
SUBTRACT NORMALIZED (register)	SFR	f,f	FF	FP	C		Op Eu Eo Si
SUBTRACT UNNORMALIZED	SU	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eo Si
SUBTRACT UNNORMALIZED (register)	SUR	f,f	FF	FP	C		Op Eo Si
SUBTRACT WITH CARRY (byte, register)	SYR	rpb,rpb rsb,rsb	RR	G	C		
SUBTRACT WITH CARRY (halfword, register)	SYHR	rh,rh	RR	G	C		
SUBTRACT WITH CARRY (halfword, register, extended)	SYHRE	ruh,ruh	RR	G	C		
TEST (byte, register-immediate)	TRI	rpb,i8	RI	G	C		
TEST AND SET (byte)	TS	0,ra	RR	G	C		Sp Ac Op Se Ad
WRITE COMMON MASK	KI	rupb,2	RI	CC		S	Op
WRITE CONDITION INDICATORS	KI	rupb,26	RI	CC	C	S	Op
WRITE DCV	KI	rpw,192	RI			S	Op

Instructions Arranged by Name (continued)

Name	Mnemonic	Operands	Characteristics F	T	C	P	Exceptions
WRITE ERROR INTERRUPT REQUEST VECTOR	KI	rupb,8	RI	CC		S	Sp[5] Op
WRITE FLOATING-POINT CONTROL	WFC	dw14s(ra)	FS	FP			Sp Ac Op Se Ad
WRITE FLOATING-POINT STATUS VECTOR	WFS	dw14s(ra)	FS	FP		S	Sp Ac Op Se Ad Fp Fa
WRITE PRIMARY REGISTER SET NUMBER	KI	rupb,10	RI	CC		S	Sp[5] Op
WRITE PROGRAM ACTIVATION VECTOR	KI	rupb,120	RI	CC		S	Sp[5] Op
WRITE SECONDARY REGISTER SET NUMBER	KI	rupb,12	RI	CC		S	Sp[5] Op

Notes:

1. Only low-order 8 bits of register halfword participate in operation.
2. Register specification limited to 0, 4, 8, ..., or 28 (DPPX assembler numbering).
3. Only low-order 16 bits of register participate in operation.
4. Register specification limited to 12, 14, 28, or 30 (DPPX assembler numbering).
5. Depending on processor model.

Instructions Arranged by Mnemonic

Mnemonic	Name	Operands	Characteristics F	T	C	P	Exceptions
AF	ADD NORMALIZED	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eu Eo Si
AFR	ADD NORMALIZED (register)	f,f	FF	FP	C		Op Eu Eo Si
AHR	ADD (halfword, register)	rh,rh	RR	G	C		
AHRI	ADD (halfword, register-immediate)	rh,i4	RR	G	C		
AR	ADD (byte, register)	rpb,rpb rsb,rsb	RR	G	C		
ARI	ADD (byte, register-immediate	rpb,i8s	RI	G	C		
AU	ADD UNNORMALIZED	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eo Si
AUR	ADD UNNORMALIZED (register)	f,f	FF	FP	C		Op Eo Si
AYHR	ADD WITH CARRY (halfword, register)	rh,rh	RR	G	C		
AYHRE	ADD WITH CARRY (halfword, register, extended)	ruh,ruh	RR	G	C		
AYR	ADD WITH CARRY (byte, register)	rpb,rpb rsb,rsb	RR	G	C		
BAL	BRANCH AND LINK	ra,dh16s(ra)	RSL	G			Sp Ac Se Ad
BALR	BRANCH AND LINK (register)	ra,ra	RR	G			Sp Ac Se Ad
BC	BRANCH ON CONDITION	m4,dh16s(ra)	RSL	G			Sp Ac Se Ad
BCR	BRANCH ON CONDITION (register)	m4,ra	RR	G			Sp Ac Se Ad
BCTR	BRANCH ON COUNT (byte, register)	rpb,ra	RR	G			Sp Ac Se Ad
BNX	BRANCH ON INDEX (byte)	rpb,ra	RR	G			Sp Ac Se Ad
CF	COMPARE	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad
CFR	COMPARE (register)	f,f	FF	FP	C		Op
CHR	COMPARE (halfword, register)	rh,rh	RR	G	C		
CLHS	COMPARE LOGICAL (halfwords, storage)	ra,ra,rh[1]	RRL	G	C		Sp Ac Op[5] Se Ad
CLS	COMPARE LOGICAL (bytes, storage)	ra,ra,rh[1]	RRL	G	C		Sp Ac Op[5] Se Ad
CR	COMPARE (byte, register)	rpb,rpb rsb,rsb	RR	G	C		
CTLZ	COUNT LEADING ZEROS (halfword)	rh,rh	RR	G	C		
CYHRE	COMPARE WITH CARRY (halfword, register, extended)	ruh,ruh	RR	G	C		
DF	DIVIDE	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad Eu Eo Fd
DFR	DIVIDE (register)	f,f	FF	FP			Op Eu Eo Fd
DHR	DIVIDE (halfword, register)	rh[2],rh	RRL	G			Op Ov
IO	INPUT/OUTPUT (byte)	rpb,rh	RR	IO	C	I	Op
IOH	INPUT/OUTPUT (halfword)	rh,rh	RR	IO	C	I	Op
IOI	INPUT/OUTPUT (byte, immediate)	rlpb,i8	RI	IO	C	I	Op
JBZ	JUMP ON BIT ZERO (halfword)	n4,dh7s	J	G			Sp Ac Se Ad
JC	JUMP ON CONDITION	m4,dh7s	J	G			Sp Ac Se Ad
KDO	CONTROL DIRECT OUT	i4	RR	CC			Op
KI	AND WITH PROGRAMMED INTERRUPT REQUEST VECTOR	rupb,6	RI	CC		S	Op
KI	CALL PSV	0,127	RI	G			
KI	DISPATCH NEW LEVEL	rupb,28	RI	CC		S	Sp[5] Op
KI	OR WITH PROGRAMMED INTERRUPT REQUEST VECTOR)	rupb,4	RI	CC		S	Op
KI	READ CHANNEL MASK	rupb,25	RI	CC			
KI	READ COMMON MASK	rupb,3	RI	CC			
KI	READ CONDITION INDICATORS	rupb,27	RI	CC			
KI	READ CURRENT AND LAST LEVELS	rupb,15	RI	CC			
KI	READ DCV	rpw,193	RI				
KI	READ ERROR INTERRUPT REQUEST VECTOR	rupb,9	RI	CC			
KI	READ I/O INTERRUPT REQUEST VECTOR	rupb,7	RI	CC			
KI	READ MASTER MASK	rupb,1	RI	CC			
KI	READ PRIMARY REGISTER SET NUMBER	rupb,11	RI	CC			
KI	READ PROGRAM ACTIVATION VECTOR	rupb,121	RI	CC			
KI	READ PROGRAMMED INTERRUPT REQUEST VECTOR	rupb,5	RI	CC			

Instructions Arranged by Mnemonic (continued)

Mnemonic	Name	Operands	Characteristics F	T	C	P	Exceptions
KI	READ SECONDARY REGISTER SET NUMBER	rupb,13	RI	CC			
KI	RESET CHANNEL MASK	0,24	RI	CC		S	Op
KI	RESET MASTER MASK	0,0	RI	CC		S	Op
KI	RESET PROGRAMMED INTERRUPT REQUEST	0,37	RI	CC		I	Op
KI	SET CHANNEL MASK	0,38	RI	CC		S	Op
KI	SET MASTER MASK	0,14	RI	CC		S	Op
KI	SET PROGRAMMED INTERRUPT REQUEST	0,35	RI	CC		I	Op
KI	WRITE COMMON MASK	rupb,2	RI	CC		S	Op
KI	WRITE CONDITION INDICATORS	rupb,26	RI	CC	C	S	Op
KI	WRITE DCV	rpw,192	RI			S	Op
KI	WRITE ERROR INTERRUPT REQUEST VECTOR	rupb,8	RI	CC		S	Op
KI	WRITE PRIMARY REGISTER SET NUMBER	rupb,10	RI	CC		S	Sp[5] Op
KI	WRITE PROGRAM ACTIVATION VECTOR	rupb,120	RI	CC		S	Sp[5] Op
KI	WRITE SECONDARY REGISTER SET NUMBER	rupb,12	RI	CC		S	Sp[5] Op
L	LOAD (byte)	rpb,db16s(ra)	RSL	G			Sp Ac Se Ad
LA	LOAD ADDRESS	ra,db16s(ra)	RSL	G			
LAT	LOAD FROM ADDRESS TRANSLATION TABLE	rw,ra	RRL	TT		S	Sp Op
LATL	LOAD FROM ADDRESS TRANSLATION LOCK TABLE	rh,ra	RRL	TL		S	Sp Op
LCFR	LOAD COMPLEMENT (register)	f,f	FF	FP	C		Op
LF	LOAD	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad
LFR	LOAD (register)	f,f	FF	FP			Op
LH	LOAD (halfword)	rh,db16s(ra)	RSL	G			Sp Ac Se Ad
LHN	LOAD (halfword, with index)	rh,ra	RR	G			Sp Ac Se Ad
LHND	LOAD (halfword, with index decremented)	rh,ra	RR	G			Sp Ac Se Ad
LHNI	LOAD (halfword, with index incremented)	rh,ra	RR	G			Sp Ac Se Ad
LHQ	LOAD (halfwords, quadrant)	q2,ra	RR	G			Sp Ac Op Se Ad
LHR	LOAD (halfword, register)	rh,rh	RR	G			
LHRLU	LOAD (halfword, register, lower half from upper)	rh,ruh	RR	G			
LHRN	LOAD (halfword, register-indirect)	rh,ra[3]	RR	RN		S	Op Re
LHRU	LOAD (halfword, register, upper half)	ruh,ruh	RR	G			
LHRUL	LOAD (halfword, register, upper half from lower)	ruh,rh	RR	G			
LHS	LOAD (halfword, short form)	rh,dh5(ra[4])	RS	G			Sp Ac Se Ad
LN	LOAD (byte, with index)	rpb,ra	RR	G			Sp Ac Se Ad
LND	LOAD (byte, with index decremented)	rpb,ra	RR	G			Sp Ac Se Ad
LNFR	LOAD NEGATIVE (register)	f,f	FF	FP	C		Op
LNI	LOAD (byte, with index incremented)	rpb,ra	RR	G			Sp Ac Se Ad
LPFR	LOAD POSITIVE (register)	f,f	FF	FP	C		Op
LR	LOAD (byte, register)	rb,rb	RR	G			
LRFR	LOAD ROUNDED (register)	f,f	FF	FP			Op Eo
LRI	LOAD (byte, register-immediate)	rpb,i8	RI	G			
LRN	LOAD (byte, register-indirect)	rpb,ra[3]	RR	RN		S	Op Re
LTFR	LOAD AND TEST (register)	f,f	FF	FP	C		Op
LW	LOAD (word)	rw,db16s(ra)	RSL	G			Sp Ac Se Ad
MF	MULTIPLY	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad Eu Eo
MFR	MULTIPLY (register)	f,f	FF	FP			Op Eu Eo
MHR	MULTIPLY (halfword, register)	rh,rh	RRL	G			Op
MVHS	MOVE (halfwords, storage)	ra,ra,rh[1]	RRL	G			Sp Ac Op[5] Se Ad
MVS	MOVE (bytes, storage)	ra,ra,rh[1]	RRL	G			Sp Ac Op[5] Se Ad
NHR	AND (halfword, register)	rh,rh	RR	G	C		

Instructions Arranged by Mnemonic (continued)

Mnemonic	Name	Operands	Characteristics F	T	C	P	Exceptions
NR	AND (byte, register)	rpb,rpb rsb,rsb	RR	G	C		
NRI	AND (byte, register-immediate)	rpb,i8	RI	G	C		
OHR	OR (halfword, register)	rh,rh	RR	G	C		
OR	OR (byte, register)	rpb,rpb rsb,rsb	RR	G	C		
ORI	OR (byte, register-immediate)	rpb,i8	RI	G	C		
PC	PROGRAM EXCEPTION		RR	G			Op
RFC	READ FLOATING-POINT CONTROL	dw14s(ra)	FS	FP			Sp Ac Op Se Ad
RFS	READ FLOATING-POINT STATUS VECTOR	dw14s(ra)	FS	FP			Sp Ac Op Se Ad Fs
RL	ROTATE LEFT (byte)	rb,c3	RR	G	C		
RLH	ROTATE LEFT (halfword)	rh,c4	RR	G	C		
SF	SUBTRACT NORMALIZED	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eu Eo Si
SFOM	SET OVERFLOW MASK	m1	FF	FP			Op
SFPM	SET PRECISION MODE	m1	FF	FP			Op
SFR	SUBTRACT NORMALIZED (register)	f,f	FF	FP	C		Op Eu Eo Si
SFSM	SET SIGNIFICANCE MASK	m1	FF	FP			Op
SFUM	SET UNDERFLOW MASK	m1	FF	FP			Op
SHR	SUBTRACT (halfword, register)	rh,rh	RR	G	C		
SHRI	SUBTRACT (halfword, register-immediate)	rh,i4	RR	G	C		
SLHL	SHIFT LEFT (halfword, logical)	rh,c4	RR	G	C		
SLL	SHIFT LEFT (byte, logical)	rb,c3	RR	G	C		Op
SR	SUBTRACT (byte, register)	rpb,rpb rsb,rsb	RR	G	C		
ST	STORE (byte)	rpb,db16s(ra)	RSL	G			Sp Ac Se Ad
STAT	STORE TO ADDRESS TRANSLATION TABLE	rw,ra	RRL	TT		S	Sp Op
STATL	STORE TO ADDRESS TRANSLATION LOCK TABLE	rh,ra	RRL	TL		S	Sp Op
STF	STORE	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad
STH	STORE (halfword)	rh,db16s(ra)	RSL	G			Sp Ac Se Ad
STHN	STORE (halfword, with index)	rh,ra	RR	G			Sp Ac Se Ad
STHND	STORE (halfword, with index decremented)	rh,ra	RR	G			Sp Ac Se Ad
STHNI	STORE (halfword, with index incremented)	rh,ra	RR	G			Sp Ac Se Ad
STHQ	STORE (halfwords, quadrant)	q2,ra	RR	G			Sp Ac Op Se Ad
STHRN	STORE (halfword, register-indirect)	rh,ra[3]	RR	RN		S	Op Re
STHS	STORE (halfword, short form)	rh,dh5(ra[4])	RS	G			Sp Ac Se Ad
STN	STORE (byte, with index)	rpb,ra	RR	G			Sp Ac Se Ad
STND	STORE (byte, with index decremented)	rpb,ra	RR	G			Sp Ac Se Ad
STNI	STORE (byte, with index incremented)	rpb,ra	RR	G			Sp Ac Se Ad
STRN	STORE (byte, register-indirect)	rpb,ra[3]	RR	RN		S	Op Re
STW	STORE (word)	rw,db16s(ra)	RSL	G			Sp Ac Se Ad
SU	SUBTRACT UNNORMALIZED	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eo Si
SUR	SUBTRACT UNNORMALIZED (register)	f,f	FF	FP	C		Op Eo Si
SYHR	SUBTRACT WITH CARRY (halfword, register)	rh,rh	RR	G	C		
SYHRE	SUBTRACT WITH CARRY (halfword, register, extended)	ruh,ruh	RR	G	C		
SYR	SUBTRACT WITH CARRY (byte, register)	rpb,rpb rsb,rsb	RR	G	C		
TRI	TEST (byte, register-immediate)	rpb,i8	RI	G	C		
TS	TEST AND SET (byte)	0,ra	RR	G	C		Sp Ac Op Se Ad

Mnemonic	Name	Operands	Characteristics F	T	C	P	Exceptions
WFC	WRITE FLOATING-POINT CONTROL	dw14s(ra)	FS	FP			Sp Ac Op Se Ad
WFS	WRITE FLOATING-POINT STATUS VECTOR	dw14s(ra)	FS	FP		S	Sp Ac Op Se Ad Fp Fs
XHR	EXCLUSIVE OR (halfword, register)	rh,rh	RR	G	C		
XR	EXCLUSIVE OR (byte, register)	rpb,rpb rsb,rsb	RR	G	C		
XRI	EXCLUSIVE OR (byte, register-immediate)	rpb,i8	RI	G	C		

Notes:

1. Only low-order 8 bits of register halfword participate in operation.
2. Register specification limited to 0, 4, 8, ..., or 28 (DPPX assembler numbering).
3. Only low-order 16 bits of register participate in operation.
4. Register specification limited to 12, 14, 28, or 30 (DPPX assembler numbering).
5. Depending on processor model.

Instructions Arranged by Type

Name	Mnemonic	Operands	Characteristics F	T	C	P	Exceptions
General Instructions							
ADD (byte, register)	AR	rpb,rpb rsb,rsb	RR	G	C		
ADD (byte, register-immediate)	ARI	rpb,i8s	RI	G	C		
ADD (halfword, register)	AHR	rh,rh	RR	G	C		
ADD (halfword, register-immediate)	AHRI	rh,i4	RR	G	C		
ADD WITH CARRY (byte, register)	AYR	rpb,rpb rsb,rsb	RR	G	C		
ADD WITH CARRY (halfword, register)	AYHR	rh,rh	RR	G	C		
ADD WITH CARRY (halfword, register, extended)	AYHRE	ruh,ruh	RR	G	C		
AND (byte, register)	NR	rpb,rpb rsb,rsb	RR	G	C		
AND (byte, register-immediate)	NRI	rpb,i8	RI	G	C		
AND (halfword, register)	NHR	rh,rh	RR	G	C		
BRANCH AND LINK	BAL	ra,dh16s(ra)	RSL	G			Sp Ac Se Ad
BRANCH AND LINK (register)	BALR	ra,ra	RR	G			Sp Ac Se Ad
BRANCH ON CONDITION	BC	m4,dh16s(ra)	RSL	G			Sp Ac Se Ad
BRANCH ON CONDITION (register)	BCR	m4,ra	RR	G			Sp Ac Se Ad
BRANCH ON COUNT (byte, register)	BCTR	rpb,ra	RR	G			Sp Ac Se Ad
BRANCH ON INDEX (byte)	BNX	rpb,ra	RR	G			Sp Ac Se Ad
CALL PSV	KI	0,127	RI	G			
COMPARE (byte, register)	CR	rpb,rpb rsb,rsb	RR	G	C		
COMPARE (halfword, register)	CHR	rh,rh	RR	G	C		
COMPARE LOGICAL (bytes, storage)	CLS	ra,ra,rh[1]	RRL	G	C		Sp Ac Op[5] Se Ad
COMPARE LOGICAL (halfwords, storage)	CLHS	ra,ra,rh[1]	RRL	G	C		Sp Ac Op[5] Se Ad
COMPARE WITH CARRY (halfword, register, extended)	CYHRE	ruh,ruh	RR	G	C		
COUNT LEADING ZEROS (halfword)	CTLZ	rh,rh	RR	G	C		
DIVIDE (halfword, register)	DHR	rh[2],rh	RRL	G			Op Ov
EXCLUSIVE OR (byte, register)	XR	rpb,rpb rsb,rsb	RR	G	C		
EXCLUSIVE OR (byte, register-immediate)	XRI	rpb,i8	RI	G	C		
EXCLUSIVE OR (halfword, register)	XHR	rh,rh	RR	G	C		
JUMP ON BIT ZERO (halfword)	JBZ	n4,dh7s	J	G			Sp Ac Se Ad
JUMP ON CONDITION	JC	m4,dh7s	J	G			Sp Ac Se Ad
LOAD (byte)	L	rpb,db16s(ra)	RSL	G			Sp Ac Se Ad
LOAD (byte, register)	LR	rb,rb	RR	G			
LOAD (byte, register-immediate)	LRI	rpb,i8	RI	G			
LOAD (byte, with index)	LN	rpb,ra	RR	G			Sp Ac Se Ad
LOAD (byte, with index decremented)	LND	rpb,ra	RR	G			Sp Ac Se Ad
LOAD (byte, with index incremented)	LNI	rpb,ra	RR	G			Sp Ac Se Ad
LOAD (halfword)	LH	rh,db16s(ra)	RSL	G			Sp Ac Se Ad
LOAD (halfword, register)	LHR	rh,rh	RR	G			
LOAD (halfword, register, lower half from upper)	LHRLU	rh,ruh	RR	G			
LOAD (halfword, register, upper half)	LHRU	ruh,ruh	RR	G			

Instructions Arranged by Type (continued)

Name	Mnemonic	Operands	Characteristics F	T	C	P	Exceptions
LOAD (halfword, register, upper half from lower)	LHRUL	ruh,rh	RR	G			
LOAD (halfword, short form)	LHS	rh,dh5(ra[4])	RS	G			Sp Ac Se Ad
LOAD (halfword, with index)	LHN	rh,ra	RR	G			Sp Ac Se Ad
LOAD (halfword, with index decremented)	LHND	rh,ra	RR	G			Sp Ac Se Ad
LOAD (halfword, with index incremented)	LHNI	rh,ra	RR	G			Sp Ac Se Ad
LOAD (halfwords, quadrant)	LHQ	q2,ra	RR	G			Sp Ac Op Se Ad
LOAD (word)	LW	rw,db16s(ra)	RSL	G			Sp Ac Se Ad
LOAD ADDRESS	LA	ra,db16s(ra)	RSL	G			
MOVE (bytes, storage)	MVS	ra,ra,rh[1]	RRL	G			Sp Ac Op[5] Se Ad
MOVE (halfwords, storage)	MVHS	ra,ra,rh[1]	RRL	G			Sp Ac Op[5] Se Ad
MULTIPLY (halfword, register)	MHR	rh,rh	RRL	G			Op
OR (byte, register)	OR	rpb,rpb rsb,rsb	RR	G	C		
OR (byte, register-immediate)	ORI	rpb,i8	RI	G	C		
OR (halfword, register)	OHR	rh,rh	RR	G	C		
PROGRAM EXCEPTION	PC		RR	G			Op
ROTATE LEFT (byte)	RL	rb,c3	RR	G	C		
ROTATE LEFT (halfword)	RLH	rh,c4	RR	G	C		
SHIFT LEFT (byte, logical)	SLL	rb,c3	RR	G	C		Op
SHIFT LEFT (halfword, logical)	SLHL	rh,c4	RR	G	C		
STORE (byte)	ST	rpb,db16s(ra)	RSL	G			Sp Ac Se Ad
STORE (byte, with index)	STN	rpb,ra	RR	G			Sp Ac Se Ad
STORE (byte, with index decremented)	STND	rpb,ra	RR	G			Sp Ac Se Ad
STORE (byte, with index incremented)	STNI	rpb,ra	RR	G			Sp Ac Se Ad
STORE (halfword)	STH	rh,db16s(ra)	RSL	G			Sp Ac Se Ad
STORE (halfword, short form)	STHS	rh,dh5(ra[4])	RS	G			Sp Ac Se Ad
STORE (halfword, with index)	STHN	rh,ra	RR	G			Sp Ac Se Ad
STORE (halfword, with index decremented)	STHND	rh,ra	RR	G			Sp Ac Se Ad
STORE (halfword, with index incremented)	STHNI	rh,ra	RR	G			Sp Ac Se Ad
STORE (halfwords, quadrant)	STHQ	q2,ra	RR	G			Sp Ac Op Se Ad
STORE (word)	STW	rw,db16s(ra)	RSL	G			Sp Ac Se Ad
SUBTRACT (byte, register)	SR	rpb,rpb rsb,rsb	RR	G	C		
SUBTRACT (halfword, register)	SHR	rh,rh	RR	G	C		
SUBTRACT (halfword, register-immediate)	SHRI	rh,i4	RR	G	C		
SUBTRACT WITH CARRY (byte, register)	SYR	rpb,rpb rsb,rsb	RR	G	C		
SUBTRACT WITH CARRY (halfword, register)	SYHR	rh,rh	RR	G	C		
SUBTRACT WITH CARRY (halfword, register, extended)	SYHRE	ruh,ruh	RR	G	C		
TEST (byte, register-immediate)	TRI	rpb,i8	RI	G	C		
TEST AND SET (byte)	TS	0,ra	RR	G	C		Sp Ac Op Se Ad

Instructions Arranged by Type (continued)

Name	Mnemonic	Operands	Characteristics F	T	C	P	Exceptions
Floating-Point Instructions							
ADD NORMALIZED	AF	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eu Eo Si
ADD NORMALIZED (register)	AFR	f,f	FF	FP	C		Op Eu Eo Si
ADD UNNORMALIZED	AU	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eo Si
ADD UNNORMALIZED (register)	AUR	f,f	FF	FP	C		Op Eo Si
COMPARE	CF	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad
COMPARE (register)	CFR	f,f	FF	FP	C		Op
DIVIDE	DF	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad Eu Eo Fd
DIVIDE (register)	DFR	f,f	FF	FP			Op Eu Eo Fd
LOAD	LF	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad
LOAD (register)	LFR	f,f	FF	FP			Op
LOAD AND TEST (register)	LTFR	f,f	FF	FP	C		Op
LOAD COMPLEMENT (register)	LCFR	f,f	FF	FP	C		Op
LOAD NEGATIVE (register)	LNFR	f,f	FF	FP	C		Op
LOAD POSITIVE (register)	LPFR	f,f	FF	FP	C		Op
LOAD ROUNDED (register)	LRFR	f,f	FF	FP			Op Eo
MULTIPLY	MF	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad Eu Eo
MULTIPLY (register)	MFR	f,f	FF	FP			Op Eu Eo
READ FLOATING-POINT CONTROL	RFC	dw14s(ra)	FS	FP			Sp Ac Op Se Ad
READ FLOATING-POINT STATUS VECTOR	RFS	dw14s(ra)	FS	FP			Sp Ac Op Se Ad Fs
SET OVERFLOW MASK	SFOM	m1	FF	FP			Op
SET PRECISION MODE	SFPM	m1	FF	FP			Op
SET SIGNIFICANCE MASK	SFSM	m1	FF	FP			Op
SET UNDERFLOW MASK	SFUM	m1	FF	FP			Op
STORE	STF	f,dw14s(ra)	FS	FP			Sp Ac Op Se Ad
SUBTRACT NORMALIZED	SF	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eu Eo Si
SUBTRACT NORMALIZED (register)	SFR	f,f	FF	FP	C		Op Eu Eo Si
SUBTRACT UNNORMALIZED	SU	f,dw14s(ra)	FS	FP	C		Sp Ac Op Se Ad Eo Si
SUBTRACT UNNORMALIZED (register)	SUR	f,f	FF	FP	C		Op Eo Si
WRITE FLOATING-POINT CONTROL	WFC	dw14s(ra)	FS	FP			Sp Ac Op Se Ad
WRITE FLOATING-POINT STATUS VECTOR	WFS	dw14s(ra)	FS	FP		S	Sp Ac Op Se Ad Fp Fs
Register-Indirect Instructions							
LOAD (byte, register-indirect)	LRN	rpb,ra[3]	RR	RN		S	Op Re
LOAD (halfword, register-indirect)	LHRN	rh,ra[3]	RR	RN		S	Op Re
STORE (byte, register-indirect)	STRN	rpb,ra[3]	RR	RN		S	Op Re
STORE (halfword, register-indirect)	STHRN	rh,ra[3]	RR	RN		S	Op Re
Translation-Table Instructions							
LOAD FROM ADDRESS TRANSLATION TABLE	LAT	rw,ra	RRL	TT		S	Sp Op
STORE TO ADDRESS TRANSLATION TABLE	STAT	rw,ra	RRL	TT		S	Sp Op
Translation-Lock-Table Instructions							
LOAD FROM ADDRESS TRANSLATION LOCK TABLE	LATL	rh,ra	RRL	TL		S	Sp Op
STORE TO ADDRESS TRANSLATION LOCK TABLE	STATL	rh,ra	RRL	TL		S	Sp Op

Instructions Arranged by Type (continued)

Name	Mnemonic	Operands	Characteristics F	T	C	P	Exceptions
Input/Output Instructions							
INPUT/OUTPUT (byte)	IO	rpb,rh	RR	IO	C	I	Op
INPUT/OUTPUT (byte, immediate)	IOI	rlpb,i8	RI	IO	C	I	Op
INPUT/OUTPUT (halfword)	IOH	rh,rh	RR	IO	C	I	Op
PCE-Control Instructions							
AND WITH PROGRAMMED INTERRUPT REQUEST VECTOR	KI	rupb,6	RI	CC		S	Op
CONTROL DIRECT OUT	KDO	i4	RR	CC			Op
DISPATCH NEW LEVEL	KI	rupb,28	RI	CC		S	Sp[5] Op
OR WITH PROGRAMMED INTERRUPT REQUEST VECTOR	KI	rupb,4	RI	CC		S	Op
READ CHANNEL MASK	KI	rupb,25	RI	CC			
READ COMMON MASK	KI	rupb,3	RI	CC			
READ CONDITION INDICATORS	KI	rupb,27	RI	CC			
READ CURRENT AND LAST LEVELS	KI	rupb,15	RI	CC			
READ DCV	KI	rpw,193	RI				
READ ERROR INTERRUPT REQUEST VECTOR	KI	rupb,9	RI	CC			
READ I/O INTERRUPT REQUEST VECTOR	KI	rupb,7	RI	CC			
READ MASTER MASK	KI	rupb,1	RI	CC			
READ PRIMARY REGISTER SET NUMBER	KI	rupb,11	RI	CC			
READ PROGRAM ACTIVATION VECTOR	KI	rupb,121	RI	CC			
READ PROGRAMMED INTERRUPT REQUEST VECTOR	KI	rupb,5	RI	CC			
READ SECONDARY REGISTER SET NUMBER	KI	rupb,13	RI	CC			
RESET CHANNEL MASK	KI	0,24	RI	CC		S	Op
RESET MASTER MASK	KI	0,0	RI	CC		S	Op
RESET PROGRAMMED INTERRUPT REQUEST	KI	0,37	RI	CC		I	Op
SET CHANNEL MASK	KI	0,38	RI	CC		S	Op
SET MASTER MASK	KI	0,14	RI	CC		S	Op
SET PROGRAMMED INTERRUPT REQUEST	KI	0,35	RI	CC		I	Op
WRITE COMMON MASK	KI	rupb,2	RI	CC		S	Op
WRITE CONDITION INDICATORS	KI	rupb,26	RI	CC	C	S	Op
WRITE DCV	KI	rpw,192	RI			S	Op
WRITE ERROR INTERRUPT REQUEST VECTOR	KI	rupb,8	RI	CC		S	Op
WRITE PRIMARY REGISTER SET NUMBER	KI	rupb,10	RI	CC		S	Sp[5] Op
WRITE PROGRAM ACTIVATION VECTOR	KI	rupb,120	RI	CC		S	Sp[5] Op
WRITE SECONDARY REGISTER SET NUMBER	KI	rupb,12	RI	CC		S	Sp[5] Op

Notes:

1. Only low-order 8 bits of register halfword participate in operation.
2. Register specification limited to 0, 4, 8, ..., or 28 (DPPX assembler numbering).
3. Only low-order 16 bits of register participate in operation.
4. Register specification limited to 12, 14, 28, or 30 (DPPX assembler numbering).
5. Depending on processor model.

Appendix B. Assembler Language Operand Specification

This appendix first defines the generic assembler language operand specification used in this publication, and then describes the assembler language register specifications supported by the IBM 8100 DPPX assembler licensed program.

Generic Specification

The assembler language operand specifications used in this publication are represented symbolically. The symbols consist of one or more characters and denote, in mnemonic form, the values that are considered valid for the assembler statement. The symbols denoting register specifications also include an indication of the portion of the register that participates in the operation. Each symbol is defined below, after the following general derivations.

Immediate Specifications

i#[s]	Immediate operand
c#	Count
m#	Mask
n#	Bit number
q#	Register quadrant number

A numeral (represented as #) designates the number of bits in the instruction format that are used to represent the binary value of the immediate specification.

The character s denotes a signed immediate operand. The character is omitted if the operand is unsigned.

Register Specifications

r [u / l] [p / s] General register containing a byte operand

r[u]h	General register containing a halfword operand
rw	General register containing a word operand
ra	General register containing an address
f	Floating-point register

For byte operands, the character u or l denotes which byte (upper or lower) of the register participates in the operation. The character is omitted if either may be designated. The character p or s denotes the register set (primary or secondary) in which the register is located. The character is omitted if a register in either set may be designated.

For halfword operands, the character u denotes that the upper halfword of the register participates in the operation. The character is omitted if the lower halfword participates.

Displacement Specifications

d	b h w	#[s]	Displacement value

The character b, h, or w denotes that the binary value represented in the instruction format is in terms of bytes, halfwords, or words, respectively. Accordingly, the range of values defined below for the displacements, which are always given in terms of bytes, must be specified as an integral multiple of 1, 2, or 4.

One or two numerals (represented as #) designate the number of bits in the instruction format that are used to represent the binary value of the displacement specification.

The character s denotes that the displacement value is signed. The character is omitted if the displacement is unsigned.

Operand Specifications

The individual symbols that represent the operand specifications are defined as follows:

c3	Count value, 0 to 7.
c4	Count value, 0 to 15.
i4	Immediate value, 0 to 15.
i8	Immediate value, 0 to 255, or unstructured 8-bit value, 00000000 to 11111111.
i8s	Immediate value, -128 to 127.
m1	A 1-bit mask value, 0 or 1.
m4	Unstructured 4-bit mask value, 0000 to 1111 (may be specified as 0 to 15).
n4	Bit number, 0 to 15.
q2	Quadrant number, 0 to 3.
rb	Byte operand in the upper or lower byte-operand location (bit positions 16-23 or 24-31, respectively) of a primary or secondary general register.
rpb	Byte operand in the upper or lower byte-operand location (bit positions 16-23 or 24-31, respectively) of a primary general register.

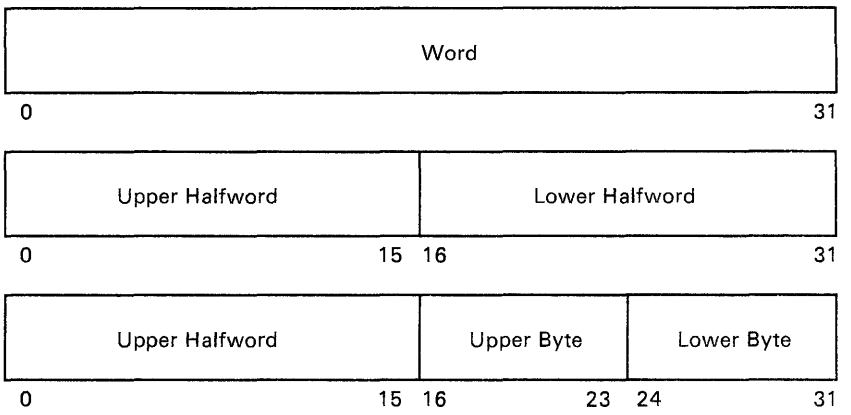
rsb	Byte operand in the upper or lower byte-operand location (bit positions 16-23 or 24-31, respectively) of a secondary general register.
rupb	Byte operand in the upper byte-operand location (bit positions 16-23) of a primary general register.
rlpb	Byte operand in the lower byte-operand location (bit positions 24-31) of a primary general register.
rh	General register containing a halfword operand in the lower half (bit positions 16-31) of the register.
ruh	General register containing a halfword operand in the upper half (bit positions 0-15) of the register.
rw	General register containing a word operand.
rpw	Primary general register containing a word operand.
ra	General register containing a storage address, a register-indirect addressing vector, a translation-table index, or a translation-lock-table index.
f	Floating-point register.
dh5	Displacement, 0 to 62, specified as a multiple of 2.
dh7s	Displacement, -128 to 126, specified as a multiple of 2.
dw14s	Displacement, -32768 to 32764, specified as a multiple of 4.
db16s	Displacement, -32768 to 32767.
dh16s	Displacement, -65536 to 65534, specified as a multiple of 2.

IBM 8100 DPPX Assembler Language Register Specifications

This part of the appendix describes the assembler language specifications, supported by the IBM 8100 DPPX assembler licensed program, that pertain to the 16 general registers and 4 floating-point registers available to a program. The specifications are the numbers used in the assembler language statement to designate the general registers, general-register operands, or floating-point registers for the machine instructions.

General Registers

General registers may be used to address main storage, the principal and adjunct register groups, the translation table, and the translation lock table; they are also used for operations involving general-register word, halfword, or byte operands. The allocation of a general register for word, halfword, or byte operands is illustrated below.



The IBM 8100 DPPX assembler treats the 16 general registers available to the program as being numbered 0,2,4,...,30. Numbers 0,2,...,14 specify general-registers 0-7, respectively, in the primary register set, and numbers 16,18,...,30 specify registers 0-7 in the secondary set. General-register word and halfword operands are specified by the numbers of the general registers in which they are located. Since two halfword operands may be allocated to one general register, the assembler language mnemonic indicates the operand location (upper or lower), where the lower location is considered the default.

The location (upper or lower) of a byte operand in a general register is not indicated with the mnemonic; it is indicated with the operand specification. Therefore, the IBM 8100 DPPX assembler treats the general-register byte-operand locations as being numbered 0,1,2,...,31. Byte-operand locations in the primary register set are numbered 0-15, and the locations in the secondary set are numbered 16-31. Each even/odd pair of numbers specifies an upper and a lower byte-operand location (register bit positions 16-23 and 24-31, respectively) in the general register identified by the even number. The specifications, including the instruction R-field (r-field for byte operands), are listed in the Figures B-1 and B-2.

Assembly Language Specification	Instruction R-Field	Register Number Within Set	
0	0000	0	Primary Register Set
2	0010	1	
4	0100	2	
6	0110	3	
8	1000	4	
10	1010	5	
12	1100	6	
14	1110	7	
16	0001	0	Secondary Register Set
18	0011	1	
20	0101	2	
22	0111	3	
24	1001	4	
26	1011	5	
28	1101	6	
30	1111	7	

Explanation:

The general register specifications correspond to the generic notation as follows:

Specification[1]		
Generic	Assembly Language	Designation
rh, ruh, rw, rpw, ra	0,2,4,. . .,30	Any of the 16 general registers

[1]For the instruction DIVIDE (halfword, register), the first-operand specification, denoted as rh, is limited to 0, 4, 8, . . . , or 28. For instructions LOAD (halfword, short) and STORE (halfword, short), the base-register specification, denoted as ra, is limited to 12, 14, 28, or 30.

Figure B-1. General Register Specifications

Assembly Language Specification		Instruction r-Field		Register Number Within Set	
Upper Byte	Lower Byte	Upper Byte	Lower Byte		
0	1	0000	0001	0	Primary Register Set
2	3	0010	0011	1	
4	5	0100	0101	2	
6	7	0110	0111	3	
8	9	1000	1001	4	
10	11	1010	1011	5	
12	13	1100	1101	6	
14	15	1110	1111	7	
16	17	0000	0001	0	Secondary Register Set
18	19	0010	0011	1	
20	21	0100	0101	2	
22	23	0110	0111	3	
24	25	1000	1001	4	
26	27	1010	1011	5	
28	29	1100	1101	6	
30	31	1110	1111	7	

Explanation:

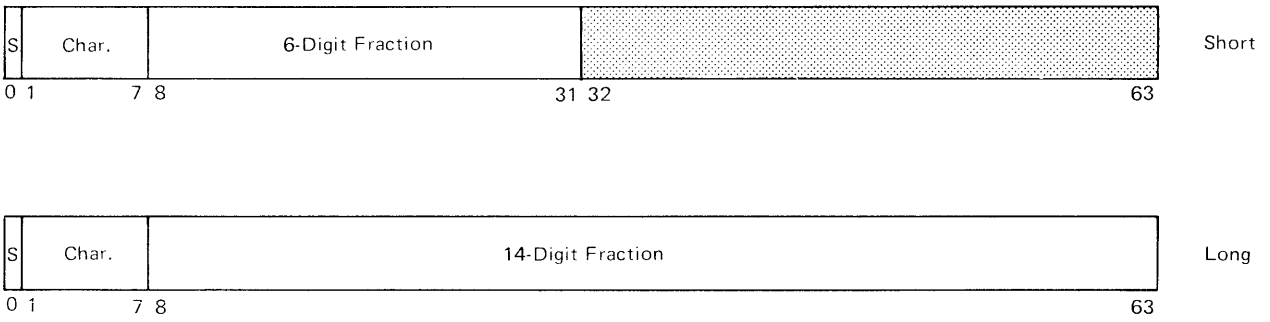
The byte operand specifications correspond to the generic notation as follows:

Specification		
Generic	Assembly Language	Designation
rb	0-31	Any of the 32 byte-operand locations in the primary and secondary sets.
rpb	0-15	Any of the 16 byte-operand locations in the primary set.
rsb	16-31	Any of the 16 byte-operand locations in the secondary set.
rupb	0, 2, 4, . . . , 14	Any of the 8 upper byte-operand locations in the primary set.
rlpb	1, 3, 5, . . . , 15	Any of the 8 lower byte-operand locations in the primary set.

Figure B-2. Byte Operand Specifications

Floating-Point Registers

Floating-point registers are used for floating-point operands. The allocation of a floating-point register for short or long floating-point numbers is illustrated below.



Floating-point register operands are specified by the numbers of the floating-point registers containing the operands. (Note that the distinction of a short or long floating-point operand is controlled by the precision bit in the FSV, and is not part of the instruction mnemonic.) The IBM 8100 DPPX assembler treats the 4 floating-point registers available to the program as being numbered 0, 4, 8, and 12.

The specifications for the registers, including the instruction F-field, are listed in Figure B-3.

Assembly Language Specification	Instruction F-Field	Register Number Within Set
0	00	0
4	01	1
8	10	2
12	11	3

Explanation:

Any of the floating-point register specifications, 0, 4, 8, or 12, may be designated for the generic specification, f.

Figure B-3. Floating-Point Register Specifications

Appendix C. Instruction Operations and Condition Settings

The first part of this appendix contains a summary listing of the instruction operations; the second part summarizes the result conditions for the instructions.

The symbols used in the operation expressions are listed below:

Symbol	Meaning
(n)	Contents of general register designated by n
<—	"is replaced by"
<==	"is determined by"
+	Addition
-	Subtraction
x	Multiplication
/	Division
MOD	Modulo division
||	Concatenation
•	Boolean AND
v	Boolean OR
¬	Boolean inverse (1's complement)
≠	Boolean exclusive OR
:	Logical comparison
=	Equal to
{n+a}	General register designated by the sum of n and a
≠	Not equal to
Bn	Instruction field designating a base register (operand number n)
C	Condition indicator for "carry" in current PSV (bit 56)
Dn	Displacement field of instruction (operand number n)
Fn	Instruction field designating a floating-point register (operand number n)
In	Immediate field of instruction (operand number n)
IA	Updated instruction address in current PSV
IOD[@]	I/O device designated by PIO address @
Mn	Mask field of instruction (operand number n)
MS[@]	Contents of main-storage location addressed by @
n<a>	Bit a of quantity identified by n
n<a..b>	Bits a through b of quantity identified by n
NSI	Next sequential instruction
PGRn	Implied primary general register number n
Qn	Instruction field designating register quadrant (operand number n)
rn	Field of instruction designating a general-register byte operand (operand n)
Rn	Field of instruction designating a general register (operand number n)
RG[@]	Contents of register-group location addressed by @
RQ<q>	Contents of general-register quadrant designated by q
TEMP	Temporary working register within PCE
TL[@]	Contents of translation-lock-table entry addressed by @
TT[@]	Contents of translation-table entry addressed by @

Instruction Operations

The following list of instruction operations is intended as a quick-reference reminder of each instruction's operation. Refer to the individual description of an instruction for its detailed specification.

The symbolic expressions used to denote the instruction operations are presented as a sequence of statements designating the operands used and the results produced. The syntax of the symbolic statements is similar to that of high-level programming languages. The order of the statements corresponds to the conceptual order in which the PCE executes the steps of the operation. Conditional execution of steps is indicated by statements of the form "If...Then...Else..."; unconditional changes in the sequence are indicated by statements using "Go To..."

Name	Mnemonic	Operation
General Instructions		
ADD (byte, register)	AR	If $r_2 \neq 0000$ Then $(r_1) \leftarrow (r_1) + (r_2)$ Else $(r_1) \leftarrow (r_1) + 00000000$
ADD (byte, register-immediate)	ARI	$(r_1) \leftarrow (r_1) + I_2$
ADD WITH CARRY (byte, register)	AYR	If $r_2 \neq 0000$ Then $(r_1) \leftarrow (r_1) + (r_2) + C$ Else $(r_1) \leftarrow (r_1) + 00000000 + C$
ADD (halfword, register)	AHR	$(R_1<16..31>) \leftarrow (R_1<16..31>) + (R_2<16..31>)$
ADD (halfword, register-immediate)	AHRI	$(R_1<16..31>) \leftarrow (R_1<16..31>) + 000000000000 \mid\mid I_2$
ADD WITH CARRY (halfword, register)	AYHR	$(R_1<16..31>) \leftarrow (R_1<16..31>) + (R<16..31>) + C$
ADD WITH CARRY (halfword, register, extended)	AYHRE	If $R_2 \neq 0000$ Then $(R_1<0..15>) \leftarrow (R_1<0..15>) + (R_2<0..15>) + C$ Else $(R_1<0..15>) \leftarrow (R_1<0..15>) + 0000000000000000 + C$
AND (byte, register)	NR	If $r_2 \neq 0000$ Then $(r_1) \leftarrow (r_1) \cdot (r_2)$ Else $(r_1) \leftarrow (r_1) \cdot 00000000$
AND (byte, register-immediate)	NRI	$(r_1) \leftarrow (r_1) \cdot I_2$
AND (halfword, register)	NHR	$(R_1<16..31>) \leftarrow (R_1<16..31>) \cdot (R_2<16..31>)$
BRANCH AND LINK	BAL	TEMP1 $\leftarrow D_2$ TEMP1 $\leftarrow$ TEMP1 rotated right 1 position If $B_2 \neq 0000$ Then TEMP2 $\leftarrow (B_2)$ + TEMP1 x 2 Else TEMP2 $\leftarrow$ IA + TEMP1 x 2 $(R_1) \leftarrow$ IA IA $\leftarrow$ TEMP2
BRANCH AND LINK (register)	BALR	TEMP $\leftarrow (R_2)$ If $R_2 \neq 0000$ Then IA $\leftarrow$ TEMP Else NSI

Name	Mnemonic	Operation
BRANCH ON CONDITION	BC	If M1 specifies an indicated result condition Then TEMP <— D_2 TEMP <— TEMP rotated right 1 position If $B_2 \neq 0000$ Then IA <— (B_2) + TEMP x 2 Else IA <— IA + TEMP x 2 Else NSI
BRANCH ON CONDITION (register)	BCR	If M1 specifies an indicated result condition Then IA <— (R_2) Else NSI
BRANCH ON COUNT (byte, register)	BCTR	TEMP <— (R_2) (r_1) <— (r_1) - 1 If $(r_1) \neq 00000000$ Then IA <— TEMP Else NSI
BRANCH ON INDEX (byte)	BNX	IA<16..31> <— MS[(R_2) + (r_1) x 2]
CALL PSV	KI	Current-PSV<40..47> <— 00000000 LPL <— CPL If Current-PSV = Primary Then Store Primary PSV PAV<CPL> <— 1 Load Secondary PSV and ACV Else Store Secondary PSV PAV<CPL> <— 0 Load Primary PSV and ACV
COMPARE (byte, register)	CR	If $r_2 \neq 0000$ Then Result-Conditions<==(r_1) + ¬(r_2) + 1 Else Result-Conditions<==(r_1) + ¬00000000 + 1
COMPARE (halfword, register)	CHR	Result-Conditions<==(R_1<16..31>) + ¬(R_2<16..31>) + 1
COMPARE WITH CARRY (halfword, register, extended	CYHRE	If $R_2 \neq 0000$ Then Result-Conditions<==(R_1<0..15>) + ¬(R_2<0..15>) + C Else Result-Conditions<==(R_1<0..15>) + ¬0000000000000000 + C
COMPARE LOGICAL (bytes, storage)	CLS	LOOP Result-Conditions<==MS[(R_1)]:MS[(R_2)] TEMP1 <— (R_1) TEMP2 <— (R_2) TEMP3 <— (R_3<24..31>) (R_1) <— TEMP1 + 1 (R_2) <— TEMP2 + 1 (R_3<24..31>) <— TEMP3 - 1 If Result-Condition=Equal and (R_3<24..31>) ≠ 00000000 Then Go To LOOP Else NSI
COMPARE LOGICAL (halfwords, storage)	CLHS	LOOP Result-Conditions<==MS[(R_1)]:MS[(R_2)] TEMP1 <— (R_1) TEMP2 <— (R_2) TEMP3 <— (R_3<24..31>) (R_1) <— TEMP1 + 2 (R_2) <— TEMP2 + 2 (R_3<24..31>) <— TEMP3 - 1 If Result-Condition=Equal and (R_3<24..31>) ≠ 00000000 Then Go To LOOP Else NSI

Name	Mnemonic	Operation
COUNT LEADING ZEROS (halfword)	CTLZ	TEMP <— 0 LOOP If (R_2<TEMP+16>) ≠ 0 Then (R_2<TEMP+16>) <— 0 Go To END Else TEMP <— TEMP + 1 If TEMP ≠ 16 Then Go To LOOP Else Continue END Result-Conditions<==(R_2<16..31>) (R_1< 16..31>) <— TEMP
DIVIDE (halfword, register)	DHR	TEMP1 <— (R_1<16..31>) | | (...R_1+0010 <16..31>) TEMP2 <— (R_2<16..31>) (R_1<16..31>) <— TEMP1 MOD TEMP2 (...R_1+0010 <16..31>) <— TEMP1 / TEMP2
EXCLUSIVE OR (byte, register)	XR	If r_2 ≠ 0000 Then (r_1) <— (r_1) ≠ (r_2) Else (r_1) <— (r_1) ≠ 00000000
EXCLUSIVE OR (byte, register-immediate)	XRI	(r_1) <— (r_1) ≠ I_2
EXCLUSIVE OR (halfword, register)	XHR	(R_1<16..31>) <— (R_1<16..31>) ≠ (R_2 <16..31>)
JUMP ON BIT ZERO (halfword)	JBZ	If (PGR1<I_1+16>) = 0 Then IA <— IA + D_2 x 2 Else NSI
JUMP ON CONDITION	JC	If M1 specifies an indicated result condition Then IA <— IA + D_2 x 2 Else NSI
LOAD ADDRESS	LA	If B_2 ≠ 0000 Then (R_1) <— (B_2) + D_2 Else (R_1) <— IA + D_2
LOAD (byte)	L	If B_2 ≠ 0000 Then (r_1) <— MS[(B_2) + D_2] Else (r_1) <— MS[IA + D_2]
LOAD (byte, with index)	LN	(r_1) <— MS[(R_2)]
LOAD (byte, with index decremented)	LND	(R_2) <— (R_2) - 1 (r_1) <— MS[(R_2)]
LOAD (byte, with index incremented)	LNI	TEMP <— MS[(R_2)] (R_2) <— (R_2) + 1 (r_1) <— TEMP
LOAD (byte, register)	LR	(r_1) <— (r_2)
LOAD (byte, register-immediate)	LRI	(r_1) <— I_2
LOAD (halfword)	LH	If B_2 ≠ 0000 Then (R_1<16..31>) <— MS[(B_2) + D_2] Else (R_1<16..31>) <— MS[IA + D_2]
LOAD (halfword, short form)	LHS	(R_1<16..31>) <— MS[(B_2) + D_2 x 2]
LOAD (halfword, with index)	LHN	(R_1<16..31>) <— MS[(R_2)]

Name	Mnemonic	Operation
LOAD (halfword, with index decremented)	LHND	$(R_2) \leftarrow (R_2) - 2$ $(R_1\langle 16..31\rangle) \leftarrow MS[(R_2)]$
LOAD (halfword, with index incremented)	LHNI	$TEMP \leftarrow MS[(R_2)]$ $(R_2) \leftarrow (R_2) + 2$ $(R_1\langle 16..31\rangle) \leftarrow TEMP$
LOAD (halfword, register)	LHR	$(R_1\langle 16..31\rangle) \leftarrow (R_2\langle 16..31\rangle)$
LOAD (halfword, register, lower half from upper)	LHRLU	$(R_1\langle 16..31\rangle) \leftarrow (R_2\langle 0..15\rangle)$
LOAD (halfword, register, upper half)	LHRU	$(R_1\langle 0..15\rangle) \leftarrow (R_2\langle 0..15\rangle)$
LOAD (halfword, register, upper half from lower)	LHRUL	$(R_1\langle 0..15\rangle) \leftarrow (R_2\langle 16..31\rangle)$
LOAD (halfwords, quadrant)	LHQ	$TEMP \leftarrow (R_2)$ $RQ\langle Q1\rangle \leftarrow MS[TEMP]$ $(R_2) \leftarrow TEMP + 16$
LOAD (word)	LW	If $B_2 \neq 0000$ Then $(R_1) \leftarrow MS[(B_2) + D_2]$ Else $(R_1) \leftarrow MS[IA + D_2]$
MOVE (bytes, storage)	MVS	LOOP $MS[(R_1)] \leftarrow MS[(R_2)]$ $TEMP1 \leftarrow (R_1)$ $TEMP2 \leftarrow (R_2)$ $TEMP3 \leftarrow (R_3\langle 24..31\rangle)$ $(R_1) \leftarrow TEMP1 + 1$ $(R_2) \leftarrow TEMP2 + 1$ $(R_3\langle 24..31\rangle) \leftarrow TEMP3 - 1$ If $(R_3\langle 24..31\rangle) \neq 00000000$ Then Go To LOOP Else NSI
MOVE (halfwords, storage)	MVHS	LOOP $MS[(R_1)] \leftarrow MS[(R_2)]$ $TEMP1 \leftarrow (R_1)$ $TEMP2 \leftarrow (R_2)$ $TEMP3 \leftarrow (R_3\langle 24..31\rangle)$ $(R_1) \leftarrow TEMP1 + 2$ $(R_2) \leftarrow TEMP2 + 2$ $(R_3\langle 24..31\rangle) \leftarrow TEMP3 - 1$ If $(R_3\langle 24..31\rangle) \neq 00000000$ Then Go To LOOP Else NSI
MULTIPLY (halfword, register)	MHR	If R_1 = xx0x (where x can be 0 or 1) Then $(R_1\langle 16..31\rangle) \mid\mid (...R_1+0010\ \langle 16..31\rangle) \leftarrow (...R_1+0010\ \langle 16..31\rangle) \times (R_2\langle 16..31\rangle)$ Else $(R_1\langle 16..31\rangle) \leftarrow (R_1\langle 16..31\rangle) \times (R_2\langle 16..31\rangle)$
OR (byte, register)	OR	If $r_2 \neq 0000$ Then $(r_1) \leftarrow (r_1) \vee (r_2)$ Else $(r_1) \leftarrow (r_1) \vee 00000000$
OR (byte, register-immediate)	ORI	$(r_1) \leftarrow (r_1) \vee I_2$
OR (halfword, register)	OHR	$(R_1\langle 16..31\rangle) \leftarrow (R_1\langle 16..31\rangle) \vee (R_2\langle 16..31\rangle)$
PROGRAM EXCEPTION	PC	Operation Exception Indicated
ROTATE LEFT (byte)	RL	$(r_1) \leftarrow (r_1)$ rotated left by I_2 amount

Name	Mnemonic	Operation
ROTATE LEFT (halfword)	RLH	$(R_1<16..31>) \leftarrow (R_1<16..31>)$ rotated left by I_2 amount
SHIFT LEFT (byte, logical)	SLL	$(r_1) \leftarrow (r_1)$ shifted left by I_2 amount
SHIFT LEFT (halfword, logical)	SLHL	$(R_1<16..31>) \leftarrow (R_1<16..31>)$ shifted left by I_2 amount
STORE (byte)	ST	If $B_2 \neq 0000$ Then $MS[(B_2) + D_2] \leftarrow (r_1)$ Else $MS[IA + D_2] \leftarrow (r_1)$
STORE (byte, with index)	STN	$MS[(R_2)] \leftarrow (r_1)$
STORE (byte, with index decremented)	STND	$TEMP \leftarrow (r_1)$ $(R_2) \leftarrow (R_2) - 1$ $MS[(R_2)] \leftarrow TEMP$
STORE (byte, with index incremented)	STNI	$MS[(R_2)] \leftarrow (r_1)$ $(R_2) \leftarrow (R_2) + 1$
STORE (halfword)	STH	If $B_2 \neq 0000$ Then $MS[(B_2) + D_2] \leftarrow (R_1<16..31>)$ Else $MS[IA + D_2] \leftarrow (R_1<16..31>)$
STORE (halfword, short form)	STHS	$MS[(B_2) + D_2 \times 2] \leftarrow (R_1<16..31>)$
STORE (halfword, with index)	STHN	$MS[(R_2)] \leftarrow (R_1<16..31>)$
STORE (halfword, with index decremented)	STHND	$TEMP \leftarrow (R_1<16..31>)$ $(R_2) \leftarrow (R_2) - 2$ $MS[(R_2)] \leftarrow TEMP$
STORE (halfword, with index incremented)	STHNI	$MS[(R_2)] \leftarrow (R_1<16..31>)$ $(R_2) \leftarrow (R_2) + 2$
STORE (halfwords, quadrant)	STHQ	$MS[(R_2)] \leftarrow RQ<Q1>$ $(R_2) \leftarrow (R_2) + 16$
STORE (word)	STW	If $B_2 \neq 0000$ Then $MS[(B_2) + D_2] \leftarrow (R_1)$ Else $MS[IA + D_2] \leftarrow (R_1)$
SUBTRACT (byte, register)	SR	If $r_2 \neq 0000$ Then $(r_1) \leftarrow (r_1) + \neg(r_2) + 1$ Else $(r_1) \leftarrow (r_1) + \neg 00000000 + 1$
SUBTRACT WITH CARRY (byte, register)	SYR	If $r_2 \neq 0000$ Then $(r_1) \leftarrow (r_1) + \neg(r_2) + C$ Else $(r_1) \leftarrow (r_1) + \neg 00000000 + C$
SUBTRACT (halfword, register)	SHR	$(R_1<16..31>) \leftarrow (R_1<16..31>) + \neg(R_2<16..31>) + 1$
SUBTRACT (halfword, register-immediate)	SHRI	$(R_1<16..31>) \leftarrow (R_1<16..31>) + \neg 000000000000 \mid\mid I_2 + 1$
SUBTRACT WITH CARRY (halfword, register)	SYHR	$(R_1<16..31>) \leftarrow (R_1<16..31>) + \neg(R_2<16..31>) + C$
SUBTRACT WITH CARRY (halfword, register, extended)	SYHRE	If $R_2 \neq 0000$ Then $(R_1<0..15>) \leftarrow (R_1<0..15>) + \neg(R_2<0..15>) + C$ Else $(R_1<0..15>) \leftarrow (R_1<0..15>) + \neg 0000000000000000 + C$

Name	Mnemonic	Operation
TEST AND SET (byte)	TS	Result-Conditions<==MS[(R_2)] MS[(R_2)] <— 11111111
TEST (byte, register-immediate)	TRI	Result-Conditions<==(r_1) tested using mask I_2
Floating-Point Instructions		
ADD NORMALIZED	AF	(F_1) <— (F_1) + MS[(B_2) + D_2 x 4]
ADD NORMALIZED (register)	AFR	(F_1) <— (F_1) + (F_2)
ADD UNNORMALIZED	AU	(F_1) <— (F_1) + MS[(B_2) + D_2 x 4]
ADD UNNORMALIZED (register)	AUR	(F_1) <— (F_1) + (F_2)
COMPARE	CF	Result-Conditions<==(F_1) - MS[(B_2) + D_2 x 4]
COMPARE (register)	CFR	Result-Conditions<==(F_1) - (F_2)
DIVIDE	DF	(F_1) <— (F_1) / MS[(B_2) + D_2 x 4]
DIVIDE (register)	DFR	(F_1) <— (F_1) / (F_2)
LOAD	LF	(F_1) <— MS[(B_2) + D_2 x 4]
LOAD (register)	LFR	(F_1) <— (F_2)
LOAD AND TEST (register)	LTFR	(F_1) <— (F_2) Result-Conditions<==(F_1)
LOAD COMPLEMENT (register)	LCFR	(F_1) <— (F_2) (F_1<0>) <— ¬(F_1<0>)
LOAD NEGATIVE (register)	LNFR	(F_1) <— (F_2) (F_1<0>) <— 1
LOAD POSITIVE (register)	LPFR	(F_1) <— (F_2) (F_1<0>) <— 0
LOAD ROUNDED (register)	LRFR	(F_1) <— (F_2) rounded long to short
MULTIPLY	MF	(F_1) <— (F_1) x MS[(B_2) + D_2 x 4]
MULTIPLY (register)	MFR	(F_1) <— (F_1) x (F_2)
READ FLOATING-POINT CONTROL	RFC	MS[(B1) + D1 x 4 + 2] <— Current-FSV<8..23>
READ FLOATING-POINT STATUS VECTOR	RFS	MS[(B1) + D1 x 4 + 1] <— FSV[MS[(B1) + D1 x 4]<5..7>]
SET OVERFLOW MASK	SFOM	Current-FSV<14> <— M
SET PRECISION MODE	SFPM	Current-FSV<11> <— M
SET SIGNIFICANCE MASK	SFSM	Current-FSV<13> <— M
SET UNDERFLOW MASK	SFUM	Current-FSV<15> <— M
STORE	STF	MS[(B_2) + D_2 x 4] <— (F_1)
SUBTRACT NORMALIZED	SF	(F_1) <— (F_1) - MS[(B_2) + D_2 x 4]

Name	Mnemonic	Operation
SUBTRACT NORMALIZED (register)	SFR	$(F_1) \leftarrow (F_1) - (F_2)$
SUBTRACT UNNORMALIZED	SU	$(F_1) \leftarrow (F_1) - MS[(B_2) + D_2 \times 4]$
SUBTRACT UNNORMALIZED (register)	SUR	$(F_1) \leftarrow (F_1) - (F_2)$
WRITE FLOATING-POINT CONTROL	WFC	Current-FSV<8..23> <— MS[(B1) + D1 x 4 + 2]
WRITE FLOATING-POINT STATUS VECTOR	WFS	FSV[MS[(B1) + D1 x 4<5..7>] <— MS[(B1) + D1 x 4 + 1]
Register-Indirect Instructions		
LOAD (byte, register-indirect)	LRN	$(r_1) \leftarrow RG[(R_2<16..31>)]$
LOAD (halfword, register-indirect)	LHRN	$(R_1<16..31>) \leftarrow RG[(R_2<16..31>)]$
STORE (byte, register-indirect)	STRN	$RG[(R_2<16..31>)] \leftarrow (r_1)$
STORE (halfword, register-indirect)	STHRN	$RG[(R_2<16..31>)] \leftarrow (R_1<16..31>)$
Translation-Table Instructions		
LOAD FROM ADDRESS TRANSLATION TABLE	LAT	TEMP <— TT[(R_2<11..31>)] (R_2<11..31>) <— (R_2<11..31>) + 1 (R_1) <— TEMP
STORE TO ADDRESS TRANSLATION TABLE	STAT	TT[(R_2<11..31>)] <— (R_1) (R_2<11..31>) <— (R_2<11..31>) + 1
Translation-Lock-Table Instructions		
LOAD FROM ADDRESS TRANSLATION LOCK TABLE	LATL	TEMP <— TL[(R_2<11..31>)] (R_2<11..31>) <— (R_2<11..31>) + 1 (R_1<16..23>) <— 00000000 (R_1<24..31>) <— TEMP
STORE TO ADDRESS TRANSLATION LOCK TABLE	STATL	TL[(R_2<11..31>)] <— (R_1<24..31>) (R_2<11..31>) <— (R_2<11..31>) + 1
Input/Output Instructions		
INPUT/OUTPUT (byte)	IO	IOD[(R_2<16..23>)] <— (R_2<24..31>) If (R_2<31>) = 0 Then IOD[(R_2<16..23>)] <— (r_1) Else (r_1) <— IOD[(R_2<16..23>)]
INPUT/OUTPUT (byte, immediate)	IOI	IOD[(PGR0<16..23>)] <— I_2 If I_2<7> = 0 Then IOD[(PGR0<16..23>)] <— (r_1) Else (r_1) <— IOD[(PGR0<16..23>)]
INPUT/OUTPUT (halfword)	IOH	IOD[(R_2<16..23>)] <— (R_2<24..31>) If (R_2<31>) = 0 Then IOD[(R_2<16..23>)] <— (R_1<16..31>) Else (R_1<16..31>) <— IOD[(R_2<16..23>)]

Name	Mnemonic	Operation
PCE-Control Instructions		
AND WITH PROGRAMMED INTERRUPT REQUEST VECTOR	KI	PIRV <— PIRV • (r_1)
DISPATCH NEW LEVEL	KI	If CPL ≠ (r_1<5..7>) Then Store Current PSV LPL <— CPL CPL <— (r_1<5..7>) Load new PSV/ACV pair indicated by PAV<CPL> Else NSI
OR WITH PROGRAMMED INTERRUPT REQUEST VECTOR	KI	PIRV <— PIRV v (r_1)
READ CHANNEL MASK	KI	(r_1) <— 0000000 | | CHM
READ COMMON MASK	KI	(r_1) <— CM
READ CONDITION INDICATORS	KI	(r_1) <— 0000 | | Current-PSV<48,49,56,57>
READ CURRENT AND LAST LEVELS	KI	(r_1) <— 0 | | CPL | | 0 | | LPL
READ DCV	KI	(R_1) <— DCV
READ ERROR INTERRUPT REQUEST VECTOR	KI	(r_1) <— EIRV
READ I/O INTERRUPT REQUEST VECTOR	KI	(r_1) <— IOIRV
READ MASTER MASK	KI	(r_1) <— 0000000 | | MM
READ PRIMARY REGISTER SET NUMBER	KI	(r_1) <— 00 | | Current-PSV<58..63>
READ PROGRAM ACTIVATION VECTOR	KI	(r_1) <— PAV
READ PROGRAMMED INTERRUPT REQUEST VECTOR	KI	(r_1) <— PIRV
READ SECONDARY REGISTER SET NUMBER	KI	(r_1) <— 00 | | Current-PSV<50..55>
RESET CHANNEL MASK	KI	CHM <— 0
RESET MASTER MASK	KI	MM <— 0
RESET PROGRAMMED INTERRUPT REQUEST	KI	PIRV<CPL> <— 0
SET CHANNEL MASK	KI	CHM <— 1
SET MASTER MASK	KI	MM <— 1
SET PROGRAMMED INTERRUPT REQUEST	KI	PIRV<CPL> <— 1

Name	Mnemonic	Operation
WRITE COMMON MASK	KI	CM <— (r_1)
WRITE CONDITION INDICATORS	KI	Current-PSV<48,49,56,57> <— (r_1<4..7>)
WRITE DCV	KI	DCV <— (R_1)
WRITE ERROR INTERRUPT REQUEST VECTOR	KI	EIRV <— (r_1)
WRITE PRIMARY REGISTER SET NUMBER	KI	Current-PSV<58..63> <— (r_1<2..7>)
WRITE PROGRAM ACTIVATION VECTOR	KI	PAV <— (r_1)
WRITE SECONDARY REGISTER SET NUMBER	KI	Current-PSV<50..55> <— (r_1<2..7>)
Direct-Control Instruction		
CONTROL DIRECT OUT	KDO	SCF <— I_1

Result Conditions

The following table summarizes the result conditions for all instructions that cause new result conditions to be indicated.

	Result Conditions				
Instruction	**8**	**4**	**2**	**1**	**0**
General Instructions					
ADD (byte, register)	0	< 0	> 0	overflow	carry
ADD (byte, register-immediate)	0	< 0	> 0	overflow	carry
ADD WITH CARRY (byte, register)	0	< 0	> 0	overflow	carry
ADD (halfword, register)	0	< 0	> 0	overflow	carry
ADD (halfword, register-immediate)	0	< 0	> 0	overflow	carry
ADD WITH CARRY (halfword, register)	0	< 0	> 0	overflow	carry
ADD WITH CARRY (halfword, register, extended)	0	< 0	> 0	overflow	carry
AND (byte, register)	0's	1's	mixed	--	--
AND (byte, register-immediate)	0's	1's	mixed	--	--
AND (halfword, register)	0's	1's	mixed	--	--
COMPARE (byte, register)	equal	low	high	overflow	carry
COMPARE (halfword, register)	equal	low	high	overflow	carry
COMPARE WITH CARRY (halfword, register, extended)	equal	low	high	overflow	carry
COMPARE LOGICAL (bytes, storage)	equal	low	high	--	carry
COMPARE LOGICAL (halfwords, storage)	equal	low	high	--	carry
COUNT LEADING ZEROS (halfword)	0's	--	mixed	--	--
EXCLUSIVE OR (byte, register)	0's	1's	mixed	--	--
EXCLUSIVE OR (byte, register-immediate)	0's	1's	mixed	--	--
EXCLUSIVE OR (halfword, register)	0's	1's	mixed	--	--
OR (byte, register)	0's	1's	mixed	--	--
OR (byte, register-immediate)	0's	1's	mixed	--	--
OR (halfword, register)	0's	1's	mixed	--	--
ROTATE LEFT (byte)	0's	leftmost bit 1	leftmost bit 0	rotated out 1	--
ROTATE LEFT (halfword)	0's	leftmost bit 1	leftmost bit 0	rotated out 1	--
SHIFT LEFT (byte, logical)	0's	leftmost bit 1	leftmost bit 0	shifted out 1	--
SHIFT LEFT (halfword, logical)	0's	leftmost bit 1	leftmost bit 0	shifted out 1	--
SUBTRACT (byte, register)	0	< 0	> 0	overflow	carry
SUBTRACT WITH CARRY (byte, register)	0	< 0	> 0	overflow	carry
SUBTRACT (halfword, register)	0	< 0	> 0	overflow	carry
SUBTRACT (halfword, register, immediate)	0	< 0	> 0	overflow	carry
SUBTRACT WITH CARRY (halfword, register)	0	< 0	> 0	overflow	carry
SUBTRACT WITH CARRY (halfword, register, extended)	0	< 0	> 0	overflow	carry
TEST AND SET (byte)	0's	1's	mixed	--	--
TEST (byte, register-immediate)	0's	1's	mixed	equal	--

Instruction	Result Conditions 8	4	2	1	0
Floating-Point Instructions					
ADD NORMALIZED	0	< 0	> 0	–	–
ADD NORMALIZED (register)	0	< 0	> 0	–	–
ADD UNNORMALIZED	0	< 0	> 0	–	–
ADD UNNORMALIZED (register)	0	< 0	> 0	–	–
COMPARE	equal	low	high	–	–
COMPARE (register)	equal	low	high	–	–
LOAD AND TEST (register)	0	< 0	> 0	–	–
LOAD COMPLEMENT (register)	0	< 0	> 0	–	–
LOAD NEGATIVE (register)	0	< 0	–	–	–
LOAD POSITIVE (register)	0	–	> 0	–	–
SUBTRACT NORMALIZED	0	< 0	> 0	–	–
SUBTRACT NORMALIZED (register)	0	< 0	> 0	–	–
SUBTRACT UNNORMALIZED	0	< 0	> 0	–	–
SUBTRACT UNNORMALIZED (register)	0	< 0	> 0	–	–
Input/Output Instructions					
INPUT/OUTPUT (byte)	–	–	PIO complete	data check	device exception
INPUT/OUTPUT (byte, immediate)	–	–	PIO complete	data check	device exception
INPUT/OUTPUT (halfword)	–	–	PIO complete	data check	device exception

Explanation:

The states of the result conditions are represented by the condition indicators in the current PSV (PSV bits 48, 49, 56, and 57) as follows, where x means that the condition-indicator bit is not significant in determining the result condition indicated:

	(PSV Bits 48,49,56,57)				
Condition Indicators	**1xxx**	**01xx**	**00xx**	**xxx1**	**xx1x**
Result Condition Indicated	8	4	2	1	0

The result conditions may also be changed by WRITE CONDITION INDICATORS and by the introduction of a new PSV.

< 0	Result is less than 0.
> 0	Result is greater than 0.
low	First operand compares low.
high	First operand compares high.
equal	First operand is identical to second operand.
0's	Result contains all 0's.
1's	Result contains all 1's.
mixed	Result contains 0's and 1's.
overflow	The operation resulted in an overflow.
carry	Operation produced a carry, PSV-bit 56 set.
--	Result condition is not indicated.

For SHIFT LEFT (byte/halfword, logical) and ROTATE LEFT (byte/halfword):

0's	Result is all 0's.
leftmost bit 1	High-order bit position is a 1.
leftmost bit 0	High-order bit position is a 0 and one or more 1's are in the remaining bit positions.
shifted out 1	One or more 1's were shifted out of the high-order bit position.
rotated out 1	One or more 1's were rotated out of the high-order bit position.

For TEST (byte,register-immediate):

0's	Bits selected under mask are all 0's, or mask is all 0's.
1's	Bits selected under mask are all 1's.
mixed	Bits selected under mask are mixed, 0's and 1's.
equal	First-operand byte is identical to mask.

For INPUT/OUTPUT (byte/byte immediate/halfword):

PIO complete	PIO operation is completed.
data check	Parity check on inbound data.
device exception	Device signaled an exception condition not normally encountered. Specific reason is device dependent.

Appendix D. Instruction Formats

This appendix contains three parts, each pertaining to instruction formats. The first part provides a summary of all formats that differ by the location of operation code fields. The second part summarizes the representations of the different displacement fields. The third part lists the specific formats of all instructions.

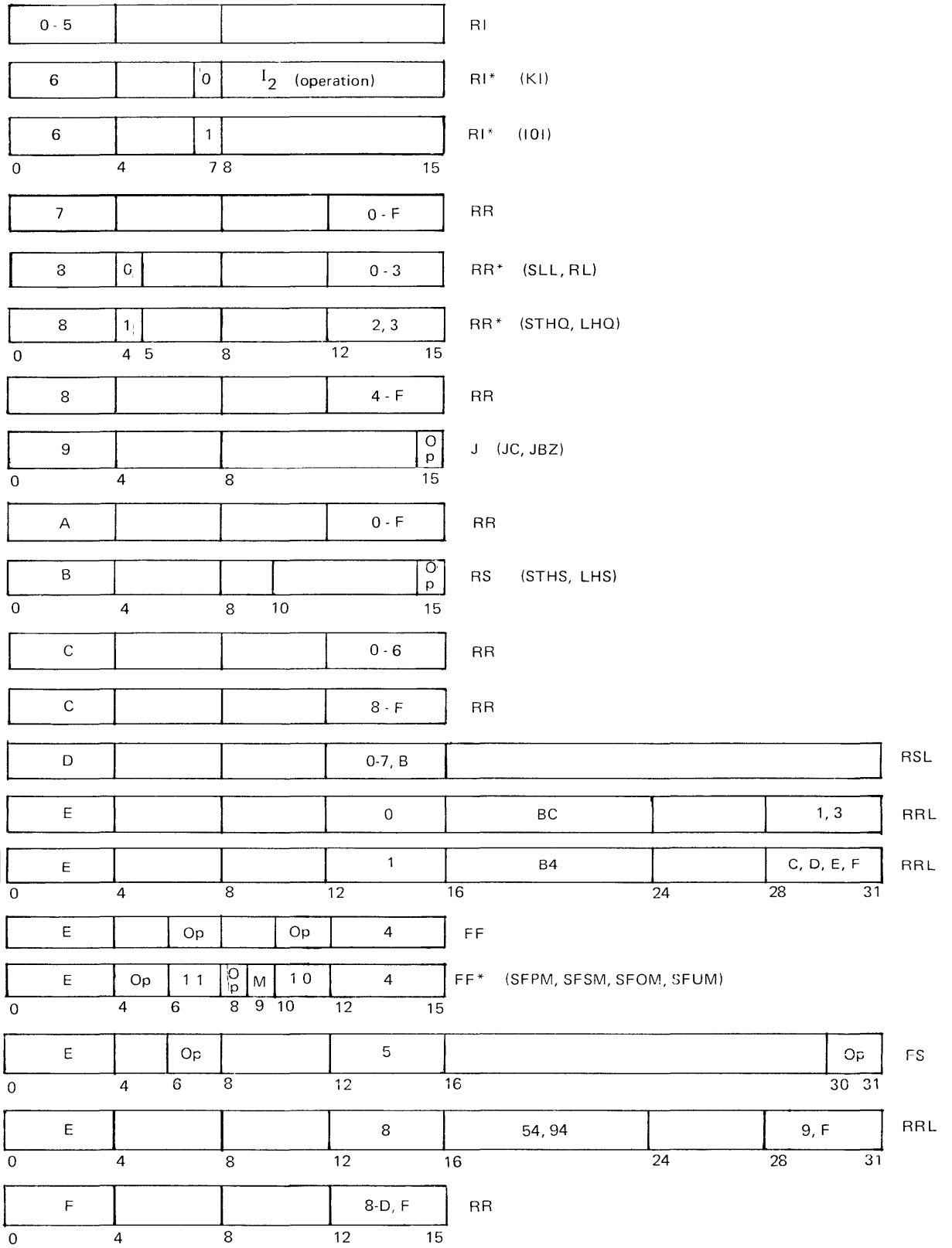
Operation Code

The following illustration summarizes the instruction formats by displaying the contents of only those fields that are used for the operation code. The formats are shown in ascending sequence based upon the value of the operation code. Listed to the right of each format is the applicable general format name. An asterisk (*) indicates that the bit positions used for the operation code differ slightly from the general format. For these formats, as well as the two general formats, J and RS, the assembler language mnemonics are also listed. For the remaining formats, refer to the third part of this appendix, "Comprehensive List of Instruction Formats", which lists all instructions and their mnemonics.

With the single exception pertaining to "Op Code" or "Op", noted below, only the operation codes that are explicitly listed are valid. Fields that are left blank are generally used for operand specifications (such as register designation, displacement value, or immediate value). These fields are not operation-code fields; their contents depend on the type of operand specification and can usually be any bit combination. The symbol "Op Code" or "Op" in the formats indicates that the specified bit positions constitute a part of the operation-code field(s); however, not all possible bit combinations represent valid operation codes.

For all instruction formats, bits 0-3 represent a portion of the operation code. These first 4 operation-code bits determine the general format of the instruction for each combination of these 4 bits, except 1110 (hexadecimal E). For this latter case, bits 12-15 also have to be considered in determining the general format. Depending on the specific format, additional operation-code information may be contained in other bit positions of the instruction format.

Contents of Fields Used for Operation Code

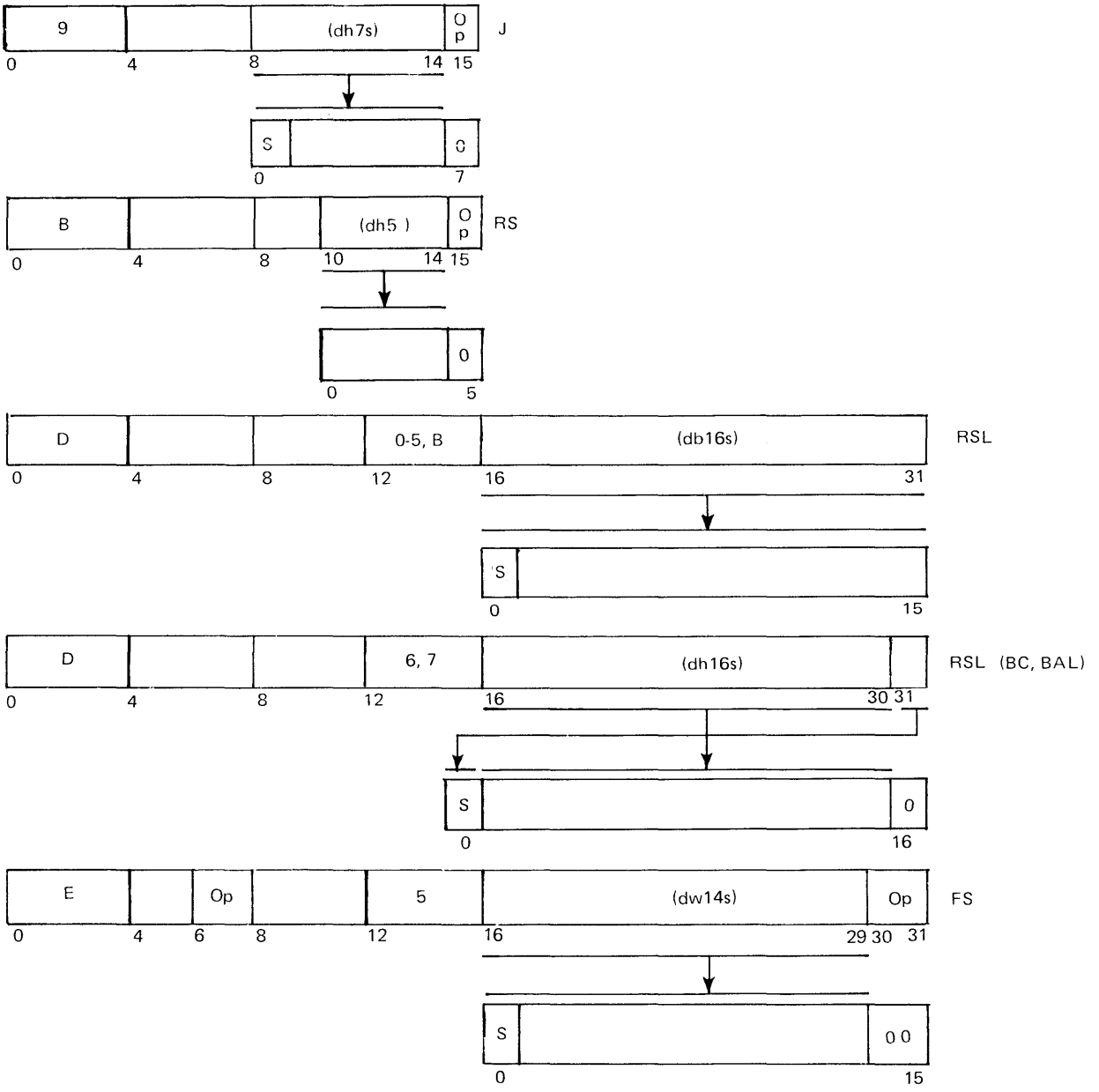


Note: * = op code bits differ from general format

Displacement Representation

The following illustration summarizes the five different displacement representations utilized within the instruction formats. For the RS-Long (RSL) format, two different representations are employed, one for the two branch instructions, BC and BAL, and the other for the remaining instructions that use this format. Each of the three other instruction formats (J, RS, and FS) contains a single displacement representation.

Except for the RS format, all displacement values are represented as signed integers, the sign bit being indicated by the letter S. For the BC and BAL instructions, the low-order bit of the displacement field contains the sign bit, which is moved to the high-order bit positions in the PCE's representation of the displacement. Otherwise, the sign bit is located in the high-order bit position of the displacement field. The notation for the assembler-language operand specification is shown in each of the displacement fields. Refer to Appendix B for a definition of these symbols.



Comprehensive List of Instruction Formats

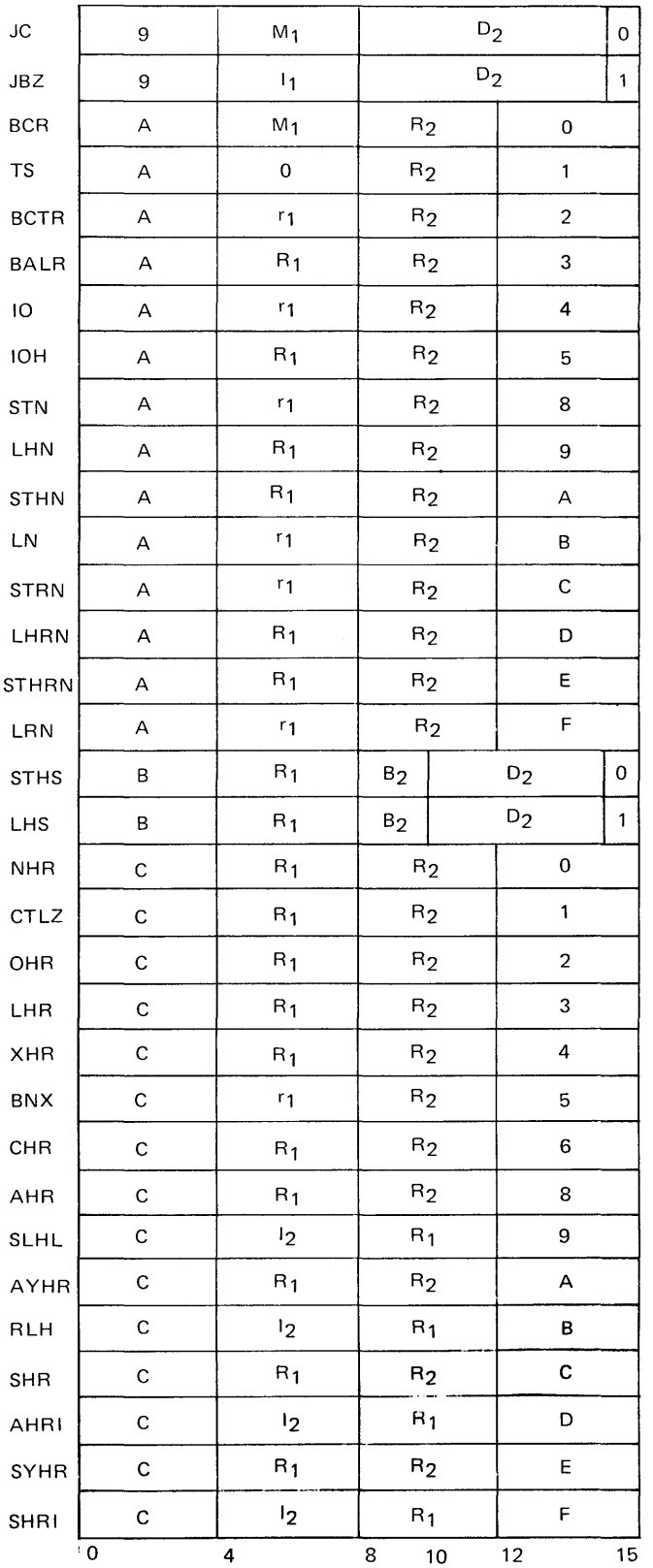
The following table illustrates the specific formats for all the instructions. The formats are shown in ascending sequence based upon the value in the operation-code fields.

The assembler-language mnemonic for each instruction appears to the left of the formats. In some cases, two or more instructions are assigned one assembler-language mnemonic. In such cases, additional information, such as the individual instruction name, appears to the right of the format to distinguish it from other formats assigned the same mnemonic.

NRI	0	r_1		I_2	
ORI	1	r_1		I_2	
XRI	2	r_1		I_2	
ARI	3	r_1		I_2	
LRI	4	r_1		I_2	
TRI	5	r_1		I_2	
KI	6	0 0 0	0	00	RESET MASTER MASK
KI	6	r_1	0	01	READ MASTER MASK
KI	6	r_1	0	02	WRITE COMMON MASK
KI	6	r_1	0	03	READ COMMON MASK
KI	6	r_1	0	04	OR WITH PROGRAMMED INTERRUPT REQUEST VECTOR
KI	6	r_1	0	05	READ PROGRAMMED INTERRUPT REQUEST VECTOR
KI	6	r_1	0	06	AND WITH PROGRAMMED INTERRUPT REQUEST VECTOR
KI	6	r_1	0	07	READ I/O INTERRUPT REQUEST VECTOR
KI	6	r_1	0	08	WRITE ERROR INTERRUPT REQUEST VECTOR
KI	6	r_1	0	09	READ ERROR INTERRUPT REQUEST VECTOR
KI	6	r_1	0	0A	WRITE PRIMARY REGISTER SET NUMBER
KI	6	r_1	0	0B	READ PRIMARY REGISTER SET NUMBER
KI	6	r_1	0	0C	WRITE SECONDARY REGISTER SET NUMBER
KI	6	r_1	0	0D	READ SECONDARY REGISTER SET NUMBER
KI	6	0 0 0	0	0E	SET MASTER MASK
KI	6	r_1	0	0F	READ CURRENT AND LAST LEVELS
KI	6	0 0 0	0	18	RESET CHANNEL MASK
KI	6	r_1	0	19	READ CHANNEL MASK
KI	6	r_1	0	1A	WRITE CONDITION INDICATORS
KI	6	r_1	0	1B	READ CONDITION INDICATORS
KI	6	r_1	0	1C	DISPATCH NEW LEVEL
KI	6	0 0 0	0	23	SET PROGRAMMED INTERRUPT REQUEST
KI	6	0 0 0	0	25	RESET PROGRAMMED INTERRUPT REQUEST
KI	6	0 0 0	0	26	SET CHANNEL MASK
KI	6	r_1	0	78	WRITE PROGRAM ACTIVATION VECTOR
KI	6	r_1	0	79	READ PROGRAM ACTIVATION VECTOR
KI	6	0 0 0	0	7F	CALL PSV
KI	6	R_1	0	C0	WRITE DIAGNOSTIC CONTROL VECTOR
KI	6	R_1	0	C1	READ DIAGNOSTIC CONTROL VECTOR
IOI	6	r_1	1	I_2	

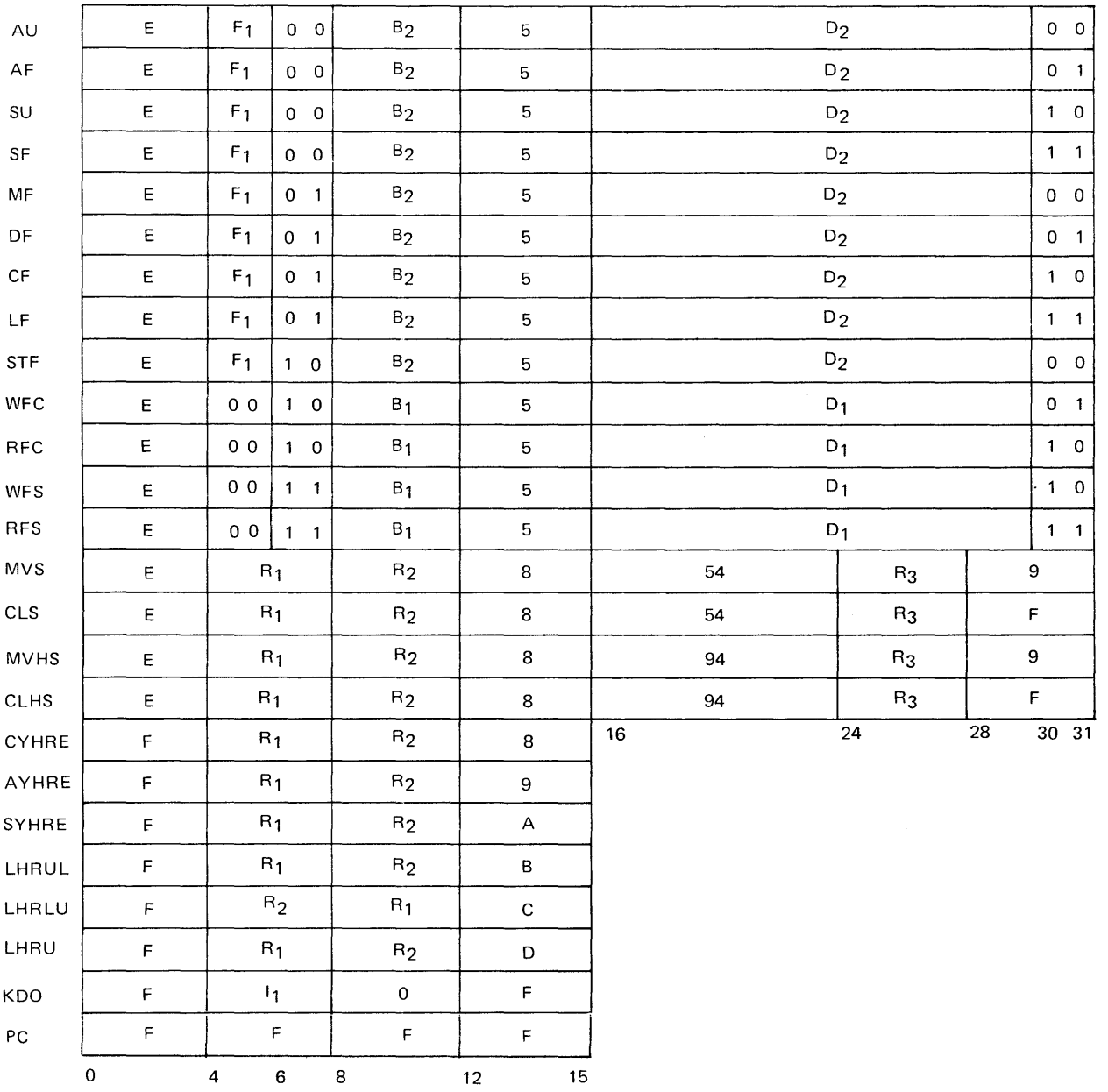
0 4 7 8 15

NR	7	r_1	r_2	0	primary-register-set operands
NR	7	r_1	r_2	1	secondary-register-set operands
OR	7	r_1	r_2	2	primary-register-set operands
OR	7	r_1	r_2	3	secondary-register-set operands
XR	7	r_1	r_2	4	primary-register-set operands
XR	7	r_1	r_2	5	secondary-register-set operands
CR	7	r_1	r_2	6	primary-register-set operands
CR	7	r_1	r_2	7	secondary-register-set operands
AR	7	r_1	r_2	8	primary-register-set operands
AR	7	r_1	r_2	9	secondary-register-set operands
AYR	7	r_1	r_2	A	primary-register-set operands
AYR	7	r_1	r_2	B	secondary-register-set operands
SR	7	r_1	r_2	C	primary-register-set operands
SR	7	r_1	r_2	D	secondary-register-set operands
SYR	7	r_1	r_2	E	primary-register-set operands
SYR	7	r_1	r_2	F	secondary-register-set operands
SLL	8	0 I_2	r_1	0	primary-register-set operand
SLL	8	0 I_2	r_1	1	secondary-register-set operand
RL	8	0 I_2	r_1	2	primary-register-set operand
STHQ	8	1 0 Q_1	R_2	2	
RL	8	0 I_2	r_1	3	secondary-register-set operand
LHQ	8	1 0 Q_1	R_2	3	
LR	8	r_1	r_2	4	r_1: primary; r_2: primary
LR	8	r_1	r_2	5	r_1: primary; r_2: secondary
LR	8	r_1	r_2	6	r_1: secondary; r_2: primary
LR	8	r_1	r_2	7	r_1: secondary; r_2: secondary
STNI	8	r_1	R_2	8	
LHNI	8	R_1	R_2	9	
STHNI	8	R_1	R_2	A	
LNI	8	r_1	R_2	B	
STND	8	r_1	R_2	C	
LHND	8	R_1	R_2	D	
STHND	8	R_1	R_2	E	
LND	8	r_1	R_2	F	
	0	4 5 6	8	12 15	


JC 9 M1 D2 0
JBZ 9 I1 D2 1
BCR A M1 R2 0
TS A 0 R2 1
BCTR A r1 R2 2
BALR A R1 R2 3
IO A r1 R2 4
IOH A R1 R2 5
STN A r1 R2 8
LHN A R1 R2 9
STHN A R1 R2 A
LN A r1 R2 B
STRN A r1 R2 C
LHRN A R1 R2 D
STHRN A R1 R2 E
LRN A r1 R2 F
STHS B R1 B2 D2 0
LHS B R1 B2 D2 1
NHR C R1 R2 0
CTLZ C R1 R2 1
OHR C R1 R2 2
LHR C R1 R2 3
XHR C R1 R2 4
BNX C r1 R2 5
CHR C R1 R2 6
AHR C R1 R2 8
SLHL C I2 R1 9
AYHR C R1 R2 A
RLH C I2 R1 B
SHR C R1 R2 C
AHRI C I2 R1 D
SYHR C R1 R2 E
SHRI C I2 R1 F
0 4 8 10 12 15

Instruction	0	4	8	12	16	24	28–31
L	D	r_1	B_2	0	D_2		
ST	D	r_1	B_2	1	D_2		
LH	D	R_1	B_2	2	D_2		
STH	D	R_1	B_2	3	D_2		
LW	D	R_1	B_2	4	D_2		
STW	D	R_1	B_2	5	D_2		
BC	D	M_1	B_2	6	D_2		
BAL	D	R_1	B_2	7	D_2		
LA	D	R_1	B_2	B	D_2		
MHR	E	R_1	R_2	0	BC	0	1
DHR	E	R_1	R_2	0	BC	0	3
LAT	E	R_1	R_2	1	B4	0	C
STAT	E	R_1	R_2	1	B4	0	D
LATL	E	R_1	R_2	1	B4	0	E
STATL	E	R_1	R_2	1	B4	0	F

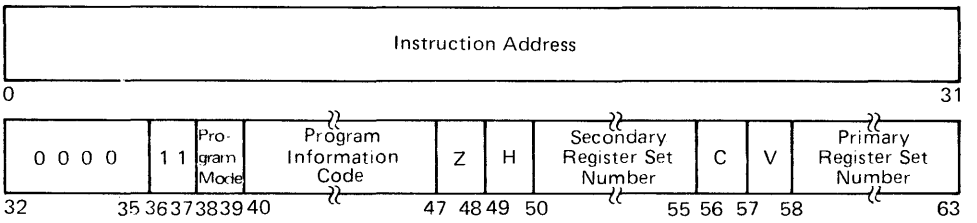
Instruction	0	4	6	8	9	10	12–15
AUR	E	F_1	0 0	F_2		0 0	4
AFR	E	F_1	0 0	F_2		0 1	4
SUR	E	F_1	0 0	F_2		1 0	4
SFR	E	F_1	0 0	F_2		1 1	4
MFR	E	F_1	0 1	F_2		0 0	4
DFR	E	F_1	0 1	F_2		0 1	4
CFR	E	F_1	0 1	F_2		1 0	4
LFR	E	F_1	0 1	F_2		1 1	4
LPFR	E	F_1	1 0	F_2		0 0	4
LNFR	E	F_1	1 0	F_2		0 1	4
LCFR	E	F_1	1 0	F_2		1 0	4
LRFR	E	F_1	1 0	F_2		1 1	4
LTFR	E	F_1	1 1	F_2		0 0	4
SFPM	E	0 1	1 1	1	M	1 0	4
SFSM	E	1 0	1 1	1	M	1 0	4
SFOM	E	1 1	1 1	0	M	1 0	4
SFUM	E	1 1	1 1	1	M	1 0	4


AU E F1 0 0 B2 5 D2 0 0
AF E F1 0 0 B2 5 D2 0 1
SU E F1 0 0 B2 5 D2 1 0
SF E F1 0 0 B2 5 D2 1 1
MF E F1 0 1 B2 5 D2 0 0
DF E F1 0 1 B2 5 D2 0 1
CF E F1 0 1 B2 5 D2 1 0
LF E F1 0 1 B2 5 D2 1 1
STF E F1 1 0 B2 5 D2 0 0
WFC E 0 0 1 0 B1 5 D1 0 1
RFC E 0 0 1 0 B1 5 D1 1 0
WFS E 0 0 1 1 B1 5 D1 1 0
RFS E 0 0 1 1 B1 5 D1 1 1
MVS E R1 R2 8 54 R3 9
CLS E R1 R2 8 54 R3 F
MVHS E R1 R2 8 94 R3 9
CLHS E R1 R2 8 94 R3 F
CYHRE F R1 R2 8
AYHRE F R1 R2 9
SYHRE F R1 R2 A
LHRUL F R1 R2 B
LHRLU F R2 R1 C
LHRU F R1 R2 D
KDO F I1 0 F
PC F F F F
0 4 6 8 12 15
16 24 28 30 31

Appendix E. Control Information Formats

This appendix summarizes of all the information formats used by the PCE to control instruction processing and interruption action, by the floating-point feature to control execution of floating-point operations, and by the channel to control channel I/O operations.

Program Status Vector



38-39 Program Mode

00	Master
01	Supervisor
11	Input/Output
10	Application

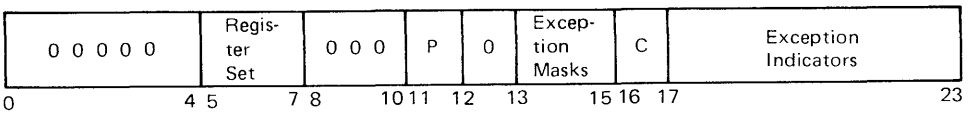
40-47 Program Information Code (after CALL PSV or program exception)

00 000000	Call PSV Executed
1m000000	Specification Exception
1m000100	Access Exception
1m001000	Operation Exception
1m001100	Separation Exception
1m010000	Address Exception
1m010100	Register-Indirect Exception
1m011000	Fixed-Point Overflow Exception
1m011100	Floating-Point Exception (see FSV)

48, 49
56, 57 (ZHCV) Condition Indicators

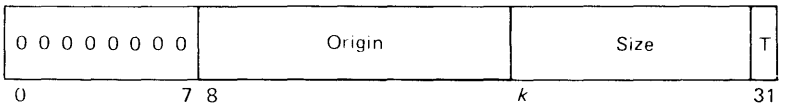
1xxx (Z=1)	Condition Indicated = 8
01xx (ZH=01)	Condition Indicated = 4
00xx (ZH=00)	Condition Indicated = 2
xxx1 (V=1)	Condition Indicated = 1
xx1x (C=1)	Condition Indicated = 0

Floating-Point Status Vector



11 Precision Mode (0 = Short, 1 = Long)
13 Significance Mask (1 = Exception masked)
14 Exponent Overflow Mask (1 = Exception masked)
15 Exponent Underflow Mask (1 = Exception masked)
16 Floating-Point Equipment Check
17 Floating-Point Operation Exception
18 Floating-Point Privileged Operation Exception
19 Floating-Point Specification Exception
20 Floating-Point Divide Exception
21 Significance Exception
22 Exponent-Overflow Exception
23 Exponent Underflow Exception

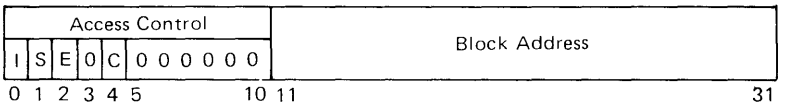
Address Control Vector



Bit position

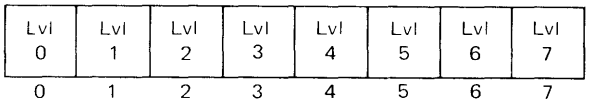
k-30	Address-Space Size	(8≤ *k* ≤29)		
		00 (29 - 30)	2,048	(2K)
		001 (28 - 30)	4,096	(4K)
		0010 (27 - 30)	8,192	(8K)
		00011 (26 - 30)	16,384	(16K)
		000111 (25 - 30)	32,768	(32K)
		0001011 (24 - 30)	65,536	(64K)
		00001111 (23 - 30)	131,072	(128K)
		000010011 (22 - 30)	262,144	(256K)
		0000010111 (21 - 30)	524,288	(512K)
		00000011011 (20 - 30)	1,048,576	(1M)
		000000011111 (19 - 30)	2,097,152	(2M)
		0000000111111 (18 - 30)	4,194,304	(4M)
		00000001011111 (17 - 30)	8,388,608	(8M)
		000000001111111 (16 - 30)	16,777,216	(16M)
		0000000011111111 (15 - 30)	33,554,432	(32M)
		00000000101111111 (14 - 30)	67,108,864	(64M)
		000000000111111111 (13 - 30)	134,217,728	(128M)
		0000000001111111111 (12 - 30)	268,435,456	(256M)
		00000000010111111111 (11 - 30)	536,870,912	(512M)
		000000000011111111111 (10 - 30)	1,073,741,824	(1G)
		0000000000111111111111 (9 - 30)	2,147,483,648	(2G)
		00000000001011111111111 (8 - 30)	4,294,967,296	(4G)
31	(T)	Translation Control Bit (1 = Translation Active)		

Dynamic Address Translation Table Entry

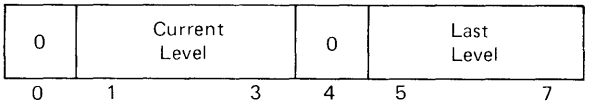


0 (I) Block Invalid (1 = No access of any type allowed)
1 (S) Store Protection (1 = Storing by program not allowed)
2 (E) Execute Protection (1 = Instruction execution not allowed)
4 (C) Channel-Store Protection (1 = Storing by channel not allowed)

Program Activation Vector



Current and Last Priority Levels



Contents of first-operand after execution of
READ CURRENT AND LAST LEVELS (K1 rubp, 15)

Interrupt Request Vectors

Programmed Interrupt Request Vector

Lvl 0	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5	Lvl 6	Lvl 7
0	1	2	3	4	5	6	7

Input/Output Interrupt Request Vector

Lvl 0	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5	Lvl 6	Lvl 7
0	1	2	3	4	5	6	7

Error Interrupt Request Vector

I	T	S	E	C	N	M	0
0	1	2	3	4	5	6	7

0	(I)	I/O Control Check
1	(T)	I/O Timeout Check
2	(S)	Storage Data Check
3	(E)	Exception
4	(C)	Channel I/O Check
5	(N)	Internal Control Check
2 and 5	(S and N)	Internal Data Check
6	(M)	Instruction Address Modifier
7		Reserved

Masks

Common Mask

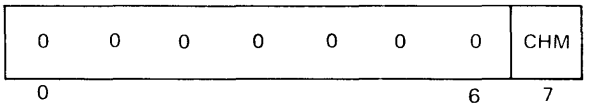
Lvl 0	Lvl 1	Lvl 2	Lvl 3	Lvl 4	Lvl 5	Lvl 6	Lvl 7
0	1	2	3	4	5	6	7

Master Mask

0	0	0	0	0	0	0	MM
0						6	7

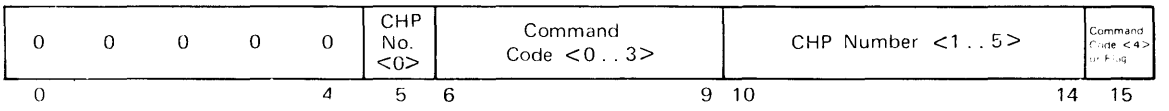
Contents of first-operand location after execution of
READ MASTER MASK (KI rupb,1)

Channel Mask



Contents of first-operand location after execution of
READ CHANNEL MASK (KI rupb, 25)

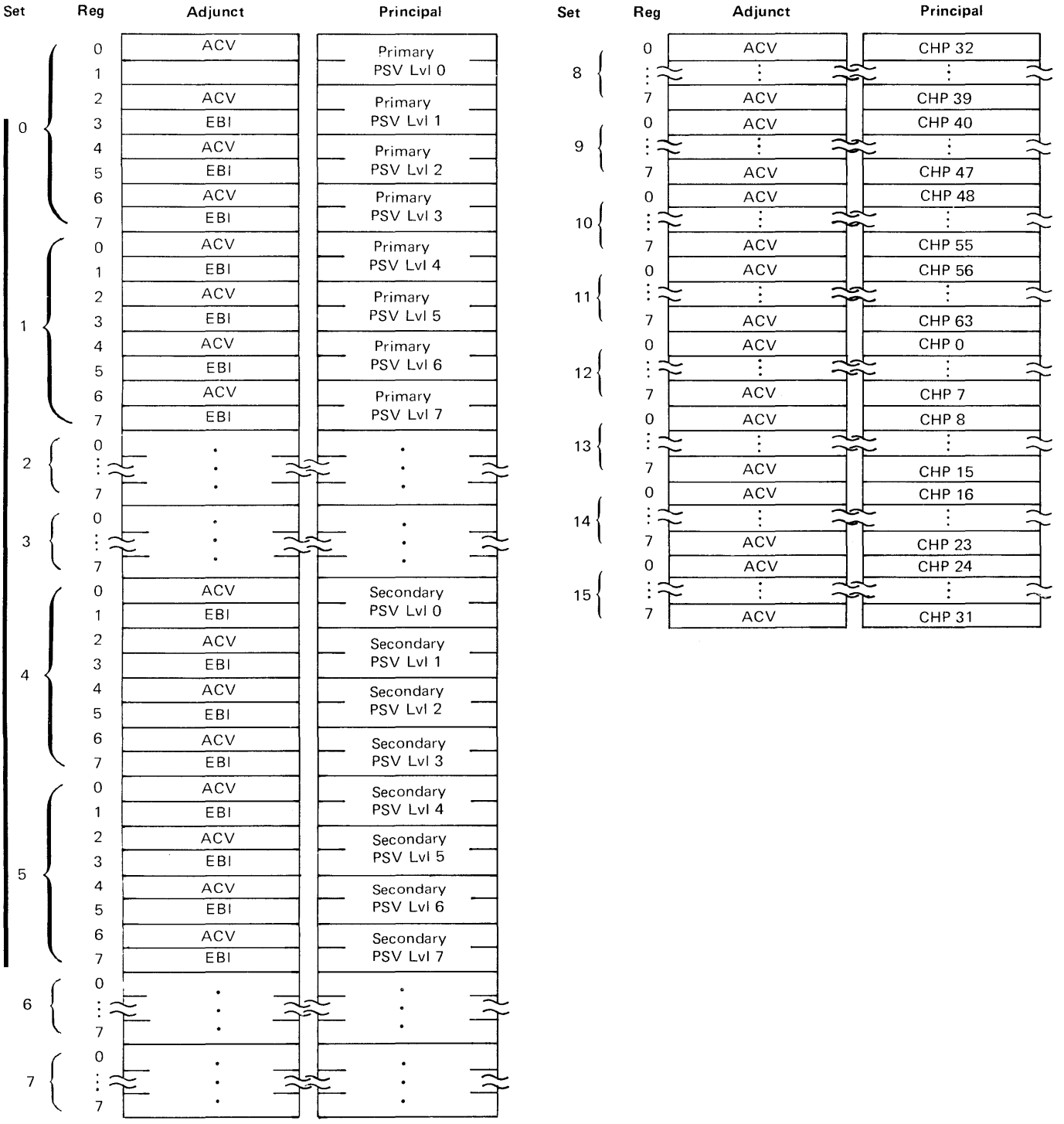
Channel Control Vector



6-9, 15 Command Code (bit 15=Command-Code<4> for byte-mode devices)
- 0 0 0 0 0 Write Data
- 0 0 1 0 0 Read Data
- 0 m0 1 m Write Data Address
- 0 m1 1 m Read Data Address
- 1 m0 0 m Read Data Address and Write Data
- 1 m1 0 m Read Data Address and Read Data

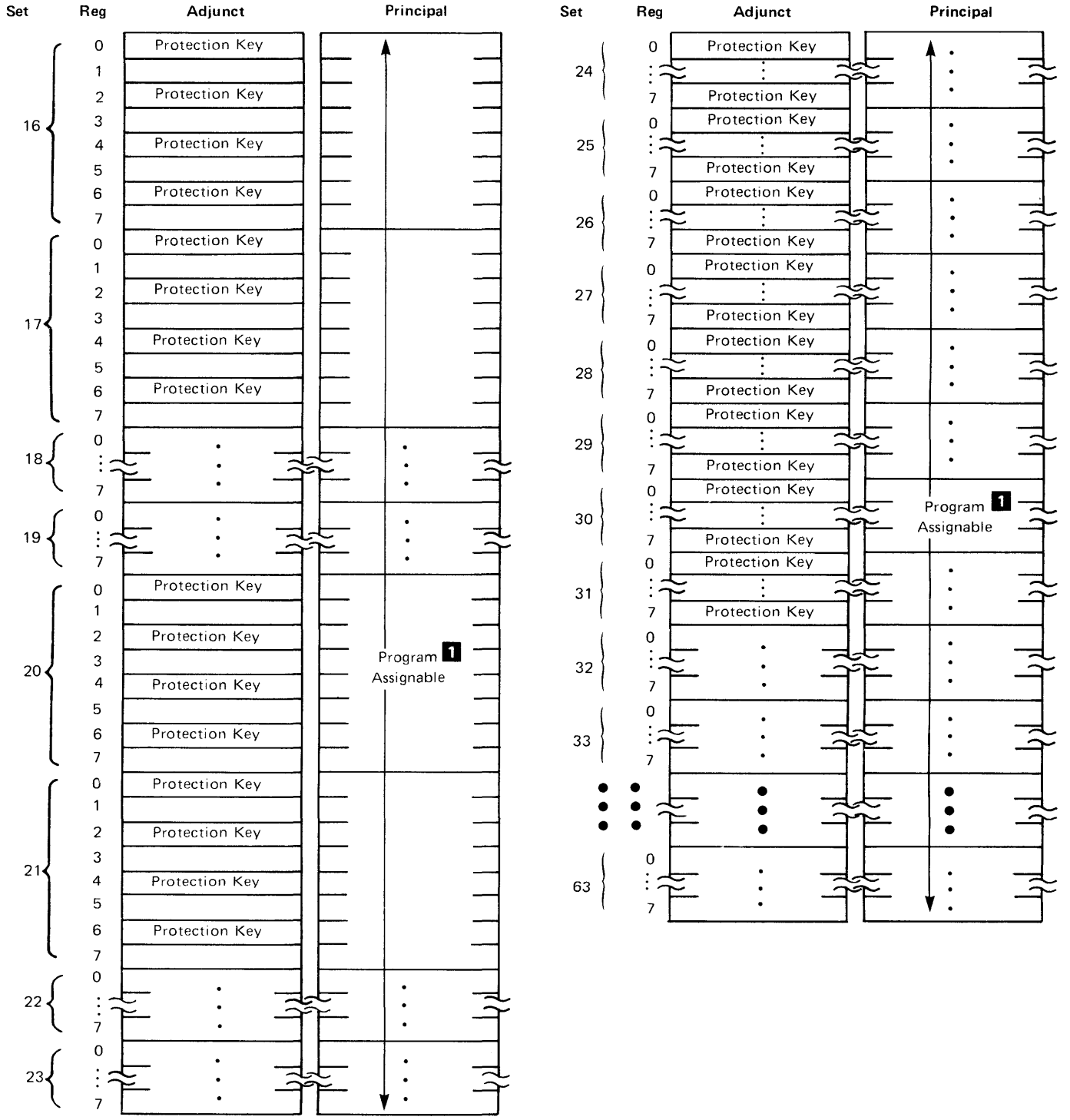
15 Flag (bit 15=CHCV Flag for halfword-mode devices)
- 0 All 16 bits of CHCV are significant
- 1 Only bits 8-15 of CHCV are significant, bits 0-7 assumed 0's

Appendix F. Assigned Register Locations



Explanation: Registers shown without designation are reserved.

Assigned Register Locations (Part 1 of 2)



Explanation: Registers shown without designation are reserved.

1 Principal register sets 16-63 may be assigned, by means of PSV bits 50-55 and 58-63, as secondary and primary register sets, respectively, for use as general registers.

Assigned Register Locations (Part 2 of 2)

Appendix G. Processor-Specific Functions

Logical Storage Addressing

The processors provide a subset of the full addressing capability defined for the 8100 system. The parameters of the specific design subset implemented on the processors are described below.

	8130 Model A	8130 Model B	8140 Models A,B	8140 Model C	8150 Model A	8150 Model B
PCE Address Space (Bytes) Without Translation With Translation	1,048,576 4,194,304	2,097,152 8,388,608	1,048,576 4,194,304	2,097,152 16,777,216	8,388,608 16,777,216	8,388,608 16,777,216
Physical Main Storage Bytes	1,048,576	2,097,152	1,048,576	2,097,152	4,194,304	8,388,608
Translation-Table Entries	2048	4096	2048	8192	8192	8192
Translation-Table Entry Restrictions	Bits 3, 5–22 must be zeros	Bits 3, 5–21 must be zeros	Bits 3, 5–22 must be zeros	Bits 3, 5–21 must be zeros	Bits 3, 5–19 must be zeros	Bits 3, 5–19 must be zeros
Translation-Table Index (R2 in LAT/STAT)	Bits 16–20 must be zeros	Bits 0–19 must be zeros	Bits 16–20 must be zeros	Bits 16–18 must be zeros	Bits 0–18 must be zeros	Bits 0–18 must be zeros

Other Processor-Specific Functions

	8130 Model A	8130 Model B	8140 Models A,B	8140 Model C	8150 Model A	8150 Model B
Floating-Point Feature	No	No	Yes	Yes [1]	Yes	Yes [1]
Instruction Pre-fetch	One Halfword	Two Halfwords	One Halfword	One [2] Halfword	Two Halfwords	Two [2] Halfwords
Exception Block Index Registers	No	Yes	No	No	Yes	Yes
Separation Protection	No	No	No	No	Yes	Yes
Translation-Lock-Table Entries	No	No	No	No	8192	8192
Channels	One	One	One	One	One	Two
LATL/STATL	No	No	No	No	Yes	Yes
READ DCV/WRITE DCV	No	Yes	No	No	No	No

[1] Available only on one PCE.

[2] For each PCE.

Address Range Control

Bits 36 and 37 of the current PSV control the storage-operand and instruction address ranges on all processors.

Bit 36 controls the range of addresses for storage-operand references when the address is generated using the contents of a general register. When the bit is 0, the range covered by a storage-operand address (A) is $0 \leq A \leq 2^{16}-1$. The high-order 16 bits of the designated general register are ignored and all 0's are logically substituted in their place during address generation.

Bit 36 also controls the updating of the storage-operand address for instructions that include address modification in their operation. When the bit is 0, only the low-order 16 bits of the register are replaced with the updated address; the high-order 16 bits remain unchanged.

Bit 37 controls the range of addresses for sequential instruction execution, for successful branches and jumps, and for storage-operand references when the storage-operand address is generated using the instruction address in the PSV. When the bit is 0, the range covered by the instruction address for sequential instruction execution, by a jump address, by a branch address, and by a storage-operand address based upon the instruction address is $M \times 2^{16} \leq A \leq (M+1) \times 2^{16}-1$, where A denotes the address and M denotes the value in the high-order 16 bits of the instruction address. When bit 37 is 0 and a branch address is designated using the contents of a general register, the high-order 16 bits of the register are ignored, and the high-order 16 bits of the instruction address are logically substituted in their place during address generation.

Bit 37 also controls all modifications to the instruction address, and controls the loading of the instruction address into the register specified in the BRANCH AND LINK operations. When the bit is 0, only the low-order 16 bits of the instruction address are updated by sequential instruction execution and are replaced by successful jumps and branches; the high-order 16 bits are unchanged. When the bit is 0, BRANCH AND LINK operations place only the low-order 16 bits of the instruction address in the specified register and leave the high-order 16 bits of the register unchanged.

Dynamic address relocation and translation take place after address generation and are not affected by PSV bits 36 and 37. During normal system operation, bits 36 and 37 of the current PSV are 1's.

Processor-Specific Error Reporting

The 8130, 8140, and 8150 report processor-specific error information in the following cases. Refer to the paragraph according to error type; within the error type, refer to processor model. If a particular processor model is not listed under the error condition, no processor-specific error information applies.

Disabling System-Check Interrupt Requests

8130 Model A, 8140. Priority level 0 is disabled for system-check interrupt requests when bit 0 of the common mask is reset to 0. Additionally, when a system-check interrupt request is indicated in the EIRV, bit 0 of the IOIRV is also set to 1.

8130 Model B, 8150. Priority level 0 cannot be disabled for system-check interrupt requests.

Reserved Program Information Code (PIC) Field

The PIC field in the stored PSV is reserved for all interruption types except for CALL PSV and program exceptions.

8130 Model A, 8140. When the PSV is stored for all other interruption types, the reserved PIC field contains zeros.

8130 Model B, 8150. When the PSV is stored for all other interruption types, the reserved PIC field remains unchanged.

Write Program Activation Vector Instruction

8130 Model B. Instruction execution is suppressed and a specification exception is indicated when the instruction attempts to change the state of the PAV bit corresponding to the current priority level.

8130 Model A, 8140, 8150. When the instruction attempts to change the state of the PAV bit corresponding to the current priority level, a specification exception is not indicated and the state change occurs. Refer to the Programming Note under "WRITE PROGRAM ACTIVATION VECTOR" in Chapter 9 for possible consequences.

Reserved Operand Bits in PCE-Control Instructions

8130 Model A, 8140. The following reserved bits in register r1 of the PCE-control instruction are ignored and execution of the instruction completes with normal results.

- WRITE EIRV — bit 7
- WRITE PRIMARY REGISTER SET NUMBER — bits 0, 1
- WRITE SECONDARY REGISTER SET NUMBER — bits 0, 1
- DISPATCH NEW LEVEL — bit 4

8130 Model B, 8150. A specification exception is indicated and the execution of the instruction is suppressed if the above reserved bits are not equal to zero.

Reserved Channel Control Vector (CHCV) Command Codes

8130 Model A, 8140. If the CHCV specifies a reserved command code, the results of the channel I/O operation are unpredictable.

8130 Model B, 8150. If the CHCV specifies a reserved command code, a channel exception is indicated with EIRV bits 3 and 4 set to 1, and a system-check interruption occurs.

Specification of Count for Interruptible Instructions

8130 Model A, 8140. When the register specified for the third operand (count) is the same as that specified for the first or second operand of the MOVE or COMPARE LOGICAL interruptible instructions, the results of the instruction are unpredictable and a program exception is not indicated.

8130 Model B. When the register specified for the third operand (count) is the same as that specified for the first or second operand of the MOVE or COMPARE LOGICAL interruptible instructions, an operation exception is indicated and instruction execution is suppressed.

8150. When the register specified for the third operand (count) is the same as that specified for the first or second operand of the MOVE or COMPARE LOGICAL interruptible instructions, an operation exception is indicated and instruction execution is suppressed only if the primary and secondary register set number fields in the current PSV specify different register sets; otherwise, the results of the instruction are unpredictable and a program exception is not indicated.

Suspended PCE Operation in Dual-PCE Processors

8140 Model C. When instruction execution in the PCE that attaches the channel was suspended because that PCE has reset its master mask, and an error condition during a channel I/O operation is detected in which one or more EIRV bits are set, an interruption does not occur immediately. PCE instruction execution remains suspended until the other PCE again sets its master mask. When instruction execution resumes, the PCE begins processing with a pending system-check interrupt request for priority level 0.

When EIRV bit 4 has been set to 1 by a WRITE EIRV instruction and instruction execution is subsequently suspended in the PCE that attaches the channel, channel I/O operations are allowed to occur.

8150 Model B. When instruction execution in one of the PCEs was suspended because that PCE has reset its master mask, and an error condition during a channel I/O operation is detected in which one or more EIRV bits are set, an interruption for priority level 0 occurs. PCE instruction execution remains suspended until the other PCE again sets its master mask. When instruction execution resumes, the PCE is at priority level 0 waiting to process the error condition.

When EIRV bit 4 has been set to 1 by a WRITE EIRV instruction and instruction execution in one of the PCEs is subsequently suspended, channel I/O operations in the suspended PCE are inhibited as long as the PCE remains suspended.

Instruction Address

8130 Model A, 8140 Models A and B. Introducing a PSV having an invalid instruction address causes a program-exception interruption. During the storing of the PSV because of this interruption, only bits 9-31 of the instruction address are stored; bits 0-7 in the PSV register location remain unchanged, and bit 8 is set to 0.

Address Exception

8130 Model A, 8140 Models A and B. When a logical address that exceeds 8388607 is generated by adding a positive displacement to the contents of a register containing a base address which is less than 8388607, the high-order 16 bits of the generated address are set to zeros. If and when the modified logical address is used to reference main storage, a resulting program exception, if any, is

indicated as either a specification exception (code 0), an access exception (code 1), or an address exception (code 4), depending upon the attributes associated with the modified address and its storage reference.

When a logical address exceeding 8388607 is obtained in any way other than that described in the preceding paragraph, (for example, the initial contents of a base-address register or the register containing a logical address that is used to reference storage are greater than 8388607), a specification exception (code 0) is indicated instead of an address exception (code 4). The exception is detected even if the address is not used to refer to main storage, such as in a branching operation in which the branch is not taken.

8140 Model C. When a logical address exceeding 33554431 is generated, a specification exception (code 0) is indicated instead of an address exception (code 4). The exception is detected even if the address is not used to refer to main storage, such as in a branching operation in which the branch is not taken.

Unit of Operation

8130 Model A, 8140. The interruptible instructions CLS, CLHS, MVS, and MVHS execute up to 8 units of operation between allowable interruptions, depending on the count value in R3 and, for COMPARE LOGICAL instructions, the point at which an inequality is detected. The first grouping of units of operation will be less than 8 if the count is not a multiple of 8. For example, for MOVE operations, the groupings for a count of 19 would be 3, 8, and 8 units of operation. The operand address and count value are updated at the end of every grouping at which an interruption is allowed. If a specification, access, or address exception is recognized, the registers containing the addresses and count will reflect the last updated values. Thus, the updated addresses and count will be within 8 units of the invalid address. The units of operation corresponding to the range of addresses beginning with the one following the last updated address and ending with the invalid address are terminated.

8130 Model B, 8150. The interruptible instructions CLS, CLHS, MVS, and MVHS execute one unit of operation between allowable interruptions. If a program exception occurs, the instruction execution is suspended. The registers containing the addresses and count will reflect the invalid address, and the instruction may be retried if the cause of the suspension is removed.

Prefetch Errors

8130 Model A, 8140. When a storage data check or internal data check is detected during the prefetching of information, a system-check interruption occurs even if the information is not used.

EIRV Variations

8140. EIRV bit 5 may be set in addition to bit 2 when a storage data check is detected during the fetching of the second operand specified in a floating-point FS instruction. Additionally, floating-point status vector bit 16 (Floating-Point Check) may be set when executing the FS instruction.

8150. EIRV bit 5 may be set in addition to bit 2 when a storage data check is detected during the fetching of either the second operand specified in a floating-point FS instruction or the first halfword of the first operand specified in

the LHQ instruction. Additionally, floating-point status vector bit 16 (Floating-Point Check) may be set when executing the FS instruction.

When dynamic address translation is active and an internal data check is detected during the fetching of the second operand from the translation table or the translation lock table specified in the LAT or LATL instruction, respectively, only EIRV bit 5 is set.

When an I/O control or timeout check is detected during a channel I/O operation, EIRV bit 5 may be set in addition to bits 0 and 4, or 1 and 4, respectively.

Detection of Concurrent Program Exceptions

8150. When an address limit condition is detected and subsequent program exception conditions are also detected during the fetching of the first halfword of an instruction, an address exception may not always be indicated. This can occur if the referenced storage location contains information that can be interpreted as an invalid instruction. In this case, an operation exception may be indicated if it is the only other program exception detected. If a separation, access, or specification exception is also detected, an address exception is indicated. When the address-limit condition is detected for the target address of a BRANCH or JUMP operation that is taken, an address exception is indicated.

Address Range Error

8130 Model A, 8140. If either bit 36 or 37 in the current PSV is 0, a program exception is indicated when a carry out of the low-order 16 bits of the address occurs during address generation or sequential updating of the instruction address. Either a program exception code of 0 or 4 is indicated, depending on the operation and the activity in the PCE and channel at the time the exception is detected. Execution of the current instruction is terminated.

8130 Model B. If either bit 36 or 37 in the current PSV is 0, a program exception is indicated when a carry out of the low-order 16 bits of the address occurs during address generation or sequential updating of the instruction address. An address exception (code 4) is indicated, and instruction execution is terminated. When an address exception results from the carry out of the low-order 16 bits of the instruction address and bit 37 of the current PSV is 0, the PSV is stored with bit 7 of the instruction address set to 1 to indicate the address overflow.

8150. If either bit 36 or 37 in the current PSV is 0, a program exception is indicated when a carry out of the low-order 16 bits of the address occurs during address generation or sequential updating of the instruction address. An address exception (code 4) is indicated, and instruction execution is terminated.

Execution of floating-point instructions produces unpredictable results when either bit 36 or 37 in the current PSV is 0.

Appendix H. Tables of Powers of 2

PLUS		*MINUS*
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.01562 5
128	7	0.00781 25
256	8	0.00390 625
512	9	0.00195 3125
1,024	10	0.00097 65625
2,048	11	0.00048 82812 5
4,096	12	0.00024 41406 25
8,192	13	0.00012 20703 125
16,384	14	0.00006 10351 5625
32,768	15	0.00003 05175 78125
65,536	16	0.00001 52587 89062 5
131,072	17	0.00000 76293 94531 25
262,144	18	0.00000 38146 97265 625
524,288	19	0.00000 19073 48632 8125
1,048,576	20	0.00000 09536 74316 40625
2,097,152	21	0.00000 04768 37158 20312 5
4,194,304	22	0.00000 02384 18579 10156 25
8,388,608	23	0.00000 01192 09289 55078 125
16,777,216	24	0.00000 00596 04644 77539 0625
33,554,432	25	0.00000 00298 02322 38769 53125
67,108,864	26	0.00000 00149 01161 19384 76562 5
134,217,728	27	0.00000 00074 50580 59692 38281 25
268,435,456	28	0.00000 00037 25290 29846 19140 625
536,870,912	29	0.00000 00018 62645 14923 09570 3125
1,073,741,824	30	0.00000 00009 31322 57461 54785 15625
2,147,483,648	31	0.00000 00004 65661 28730 77392 57812 5
4,294,967,296	32	0.00000 00002 32830 64365 38696 28906 25
8,589,934,592	33	0.00000 00001 16415 32182 69348 14453 125
17,179,869,184	34	0.00000 00000 58207 66091 34674 07226 5625
34,359,738,368	35	0.00000 00000 29103 83045 67337 03613 28125
68,719,476,736	36	0.00000 00000 14551 91522 83668 51806 64062 5
137,438,953,472	37	0.00000 00000 07275 95761 41834 25903 32031 25
274,877,906,944	38	0.00000 00000 03637 97880 70917 12951 66015 625
549,755,813,888	39	0.00000 00000 01818 98940 35458 56475 83007 8125
1,099,511,627,776	40	0.00000 00000 00909 49470 17729 28237 91503 90625
2,199,023,255,552	41	0.00000 00000 00454 74735 08864 64118 95751 95312 5
4,398,046,511,104	42	0.00000 00000 00227 37367 54432 32059 47875 97656 25
8,796,093,022,208	43	0.00000 00000 00113 68683 77216 16029 73937 98828 125
17,592,186,044,416	44	0.00000 00000 00056 84341 88608 08014 86968 99414 0625
35,184,372,088,832	45	0.00000 00000 00028 42170 94304 04007 43484 49707 03125
70,368,744,177,664	46	0.00000 00000 00014 21085 47152 02003 71742 24853 51562 5
140,737,488,355,328	47	0.00000 00000 00007 10542 73576 01001 85871 12426 75781 25
281,474,976,710,656	48	0.00000 00000 00003 55271 36788 00500 92935 56213 37890 625
562,949,953,421,312	49	0.00000 00000 00001 77635 68394 00250 46467 78106 68945 3125
1,125,899,906,842,624	50	0.00000 00000 00000 88817 84197 00125 23233 89053 34472 65625
2,251,799,813,685,248	51	0.00000 00000 00000 44408 92098 50062 61616 94526 67236 32812 5
4,503,599,627,370,496	52	0.00000 00000 00000 22204 46049 25031 30808 47263 33618 16406 25
9,007,199,254,740,992	53	0.00000 00000 00000 11102 23024 62515 65404 23631 66809 08203 125
18,014,398,509,481,984	54	0.00000 00000 00000 05551 11512 31257 82702 11815 83404 54101 5625
36,028,797,018,963,968	55	0.00000 00000 00000 02775 55756 15628 91351 05907 91702 27050 78125
72,057,594,037,927,936	56	0.00000 00000 00000 01387 77878 07814 45675 52953 95851 13525 39062 5
144,115,188,075,855,872	57	0.00000 00000 00000 00693 88939 03907 22837 76476 97925 56762 69531 25
288,230,376,151,711,744	58	0.00000 00000 00000 00346 94469 51953 61418 88238 48962 78381 34765 625
576,460,752,303,423,488	59	0.00000 00000 00000 00173 47234 75976 80709 44119 24481 39190 67382 8125
1,152,921,504,606,846,976	60	0.00000 00000 00000 00086 73617 37988 40354 72059 62240 69595 33691 40625
2,305,843,009,213,693,952	61	0.00000 00000 00000 00043 36808 68994 20177 36029 81120 34797 66845 70312 5
4,611,686,018,427,387,904	62	0.00000 00000 00000 00021 68404 34497 10088 68014 90560 17398 83422 85156 25
9,223,372,036,854,775,808	63	0.00000 00000 00000 00010 84202 17248 55044 34007 45280 08699 41711 42578 125
18,446,744,073,709,551,616	64	0.00000 00000 00000 00005 42101 08624 27522 17003 72640 04349 70855 71289 0625

Powers of 2 (Part 1 of 2)

18,446,744,073,709,551,616	64
36,893,488,147,419,103,232	65
73,786,976,294,838,206,464	66
147,573,952,589,676,412,928	67
295,147,905,179,352,825,856	68
590,295,810,358,705,651,712	69
1,180,591,620,717,411,303,424	70
2,361,183,241,434,822,606,848	71
4,722,366,482,869,645,213,696	72
9,444,732,965,739,290,427,392	73
18,889,465,931,478,580,854,784	74
37,778,931,862,957,161,709,568	75
75,557,863,725,914,323,419,136	76
151,115,727,451,828,646,838,272	77
302,231,454,903,657,293,676,544	78
604,462,909,807,314,587,353,088	79
1,208,925,819,614,629,174,706,176	80
2,417,851,639,229,258,349,412,352	81
4,835,703,278,458,516,698,824,704	82
9,671,406,556,917,033,397,649,408	83
19,342,813,113,834,066,795,298,816	84
38,685,626,227,668,133,590,597,632	85
77,371,252,455,336,267,181,195,264	86
154,742,504,910,672,534,362,390,528	87
309,485,009,821,345,068,724,781,056	88
618,970,019,642,690,137,449,562,112	89
1,237,940,039,285,380,274,899,124,224	90
2,475,880,078,570,760,549,798,248,448	91
4,951,760,157,141,521,099,596,496,896	92
9,903,520,314,283,042,199,192,993,792	93
19,807,040,628,566,084,398,385,987,584	94
39,614,081,257,132,168,796,771,975,168	95
79,228,162,514,264,337,593,543,950,336	96
158,456,325,028,528,675,187,087,900,672	97
316,912,650,057,057,350,374,175,801,344	98
633,825,300,114,114,700,748,351,602,688	99
1,267,650,600,228,229,401,496,703,205,376	100
2,535,301,200,456,458,802,993,406,410,752	101
5,070,602,400,912,917,605,986,812,821,504	102
10,141,204,801,825,835,211,973,625,643,008	103
20,282,409,603,651,670,423,947,251,286,016	104
40,564,819,207,303,340,847,894,502,572,032	105
81,129,638,414,606,681,695,789,005,144,064	106
162,259,276,829,213,363,391,578,010,288,128	107
324,518,553,658,426,726,783,156,020,576,256	108
649,037,107,316,853,453,566,312,041,152,512	109
1,298,074,214,633,706,907,132,624,082,305,024	110
2,596,148,429,267,413,814,265,248,164,610,048	111
5,192,296,858,534,827,628,530,496,329,220,096	112
10,384,593,717,069,655,257,060,992,658,440,192	113
20,769,187,434,139,310,514,121,985,316,880,384	114
41,538,374,868,278,621,028,243,970,633,760,768	115
83,076,749,736,557,242,056,487,941,267,521,536	116
166,153,499,473,114,484,112,975,882,535,043,072	117
332,306,998,946,228,968,225,951,765,070,086,144	118
664,613,997,892,457,936,451,903,530,140,172,288	119
1,329,227,995,784,915,872,903,807,060,280,344,576	120
2,658,455,991,569,831,745,807,614,120,560,689,152	121
5,316,911,983,139,663,491,615,228,241,121,378,304	122
10,633,823,966,279,326,983,230,456,482,242,756,608	123
21,267,647,932,558,653,966,460,912,964,485,513,216	124
42,535,295,865,117,307,932,921,825,928,971,026,432	125
85,070,591,730,234,615,865,843,651,857,942,052,864	126
170,141,183,460,469,231,731,687,303,715,884,105,728	127
340,282,366,920,938,463,463,374,607,431,768,211,456	128

Powers of 2 (Part 2 of 2)

Appendix I. Hexadecimal Tables

The following tables aid in converting hexadecimal values to decimal values, or the reverse.

Direct Conversion Tables

These table provide 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 tables.

	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

$3380_{10} = D34_{16}$

POWERS OF 16 TABLE

Example: $268,435,456_{10} = (2.68435456 \times 10^8)_{10} = 1000\ 0000_{16} = (10^7)_{16}$

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10 = A
17 592 186 044 416	11 = B
281 474 976 710 656	12 = C
4 503 599 627 370 496	13 = D
72 057 594 037 927 936	14 = E
1 152 921 504 606 846 976	15 = F

Decimal Values

Conversion Table: Hexadecimal and Decimal Fractions

HALFWORD													
BYTE					BYTE								
BITS 0123		4567			0123				4567				
Hex	Decimal	Hex	Decimal		Hex	Decimal			Hex	Decimal Equivalent			
.0	.0000	.00	.0000	0000	.000	.0000	0000	0000	.0000	.0000	0000	0000	0000
.1	.0625	.01	.0039	0625	.001	.0002	4414	0625	.0001	.0000	1525	8789	0625
.2	.1250	.02	.0078	1250	.002	.0004	8828	1250	.0002	.0000	3051	7578	1250
.3	.1875	.03	.0117	1875	.003	.0007	3242	1875	.0003	.0000	4577	6367	1875
.4	.2500	.04	.0156	2500	.004	.0009	7656	2500	.0004	.0000	6103	5156	2500
.5	.3125	.05	.0195	3125	.005	.0012	2070	3125	.0005	.0000	7629	3945	3125
.6	.3750	.06	.0234	3750	.006	.0014	6484	3750	.0006	.0000	9155	2734	3750
.7	.4375	.07	.0273	4375	.007	.0017	0898	4375	.0007	.0001	0681	1523	4375
.8	.5000	.08	.0312	5000	.008	.0019	5312	5000	.0008	.0001	2207	0312	5000
.9	.5625	.09	.0351	5625	.009	.0021	9726	5625	.0009	.0001	3732	9101	5625
.A	.6250	.0A	.0390	6250	.00A	.0024	4140	6250	.000A	.0001	5258	7890	6250
.B	.6875	.0B	.0429	6875	.00B	.0026	8554	6875	.000B	.0001	6784	6679	6875
.C	.7500	.0C	.0468	7500	.00C	.0029	2968	7500	.000C	.0001	8310	5468	7500
.D	.8125	.0D	.0507	8125	.00D	.0031	7382	8125	.000D	.0001	9836	4257	8125
.E	.8750	.0E	.0546	8750	.00E	.0034	1796	8750	.000E	.0002	1362	3046	8750
.F	.9375	.0F	.0585	9375	.00F	.0036	6210	9375	.000F	.0002	2888	1835	9375
1		2			3				4				

TO CONVERT .ABC HEXADECIMAL TO DECIMAL

Find .A in position 1 .6250

Find .0B in position 2 .0429 6875

Find .00C in position 3 .0029 2968 7500

.ABC Hex is equal to .6708 9843 7500

TO CONVERT .13 DECIMAL TO HEXADECIMAL

1. Find .1250 next lowest to .1300
   subtract -.1250 = .2 Hex
2. Find .0039 0625 next lowest to .0050 0000
   -.0039 0625 = .01
3. Find .0009 7656 2500 .0010 9375 0000
   -.0009 7656 2500 = .004
4. Find .0001 0681 1523 4375 .0001 1718 7500 0000
   -.0001 0681 1523 4375 = .0007
   .0000 1037 5976 5625 = .2147 Hex
5. .13 Decimal is approximately equal to ↑

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: $.8A7 = .540771_{10}$

$8A7_{16} = 2215_{10}$

$16^3 = 4096$

$$4096\overline{)2215.000000} \quad .540771$$

DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.

Example: $.5408_{10} = .8A7_{16}$

.5408

8 ← [8].6528 (x16)

A ← [10].4448 (x16)

7 ← [7].1168 (x16)

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 J. EBCDIC Chart

Bit Positions 4, 5, 6, 7	Second Hexadecimal Digit	Digit Punches	00				01				10				11				
			00	01	10	11	00	01	10	11	00	01	10	11	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	(1) NUL	(2) DLE	DS (3)	(4)	SP (5)	& (6)	- (7)	(8)					} (9)	{ (10)	\ (11)	0 (12)	8-1
0001	1	1	SOH	DC1	SOS				/ (13)		a	j	~		A	J	(14)	1	1
0010	2	2	STX	DC2	FS	SYN					b	k	s		B	K	S	2	2
0011	3	3	ETX	TM							c	l	t		C	L	T	3	3
0100	4	4	PF	RES	BYP	PN					d	m	u		D	M	U	4	4
0101	5	5	HT	NL	LF	RS					e	n	v		E	N	V	5	5
0110	6	6	LC	BS	ETB	UC					f	o	w		F	O	W	6	6
0111	7	7	DEL	IL	ESC	EOT					g	p	x		G	P	X	7	7
1000	8	8	GE	CAN							h	q	y		H	Q	Y	8	8
1001	9	8-1	RLF	EM						\	i	r	z		I	R	Z	9	9
1010	A	8-2	SMM	CC	SM		¢	!	¦ (15)	:								|	8-2
1011	B	8-3	VT	CU1	CU2	CU3	.	$	,	#									8-3
1100	C	8-4	FF	IFS		DC4	<	*	%	@					⌠		⌂		8-4
1101	D	8-5	CR	IGS	ENQ	NAK	(	)	_	'									8-5
1110	E	8-6	SO	IRS	ACK		+	;	>	=					⌡				8-6
1111	F	8-7	SI	IUS	BEL	SUB	|	¬	?	"								EO	8-7
			12				12				12	12		12	12	12		12	Zone Punches
				11				11				11	11	11		11	11	11	
					0				0		0		0	0	0		0	0	
			9	9	9	9									9	9	9	9	

Card Hole Patterns

(1)	12-0-9-8-1	(5)	No Punches	(9)	12-0
(2)	12-11-9-8-1	(6)	12	(10)	11-0
(3)	11-0-9-8-1	(7)	11	(11)	0-8-2
(4)	12-11-0-9-8-1	(8)	12-11-0	(12)	0

(13)	0-1
(14)	11-0-9-1
(15)	12-11

Control Character Representations

ACK	Acknowledge	EOT	End of Transmission	PF	Punch Off
BEL	Bell	ESC	Escape	PN	Punch On
BS	Backspace	ETB	End of Transmission Block	RES	Restore
BYP	Bypass	ETX	End of Text	RLF	Reverse Line Feed
CAN	Cancel	FF	Form Feed	RS	Reader Stop
CC	Cursor Control	FS	Field Separator	SI	Shift In
CR	Carriage Return	GE	Graphic Escape	SM	Set Mode
CU1	Customer Use 1	HT	Horizontal Tab	SMM	Start of Manual Message
CU2	Customer Use 2	IFS	Interchange File Separator	SO	Shift Out
CU3	Customer Use 3	IGS	Interchange Group Separator	SOH	Start of Heading
DC1	Device Control 1	IL	Idle	SOS	Start of Significance
DC2	Device Control 2	IRS	Interchange Record Separator	SP	Space
DC4	Device Control 4	IUS	Interchange Unit Separator	STX	Start of Text
DEL	Delete	LC	Lower Case	SUB	Substitute
DLE	Data Link Escape	LF	Line Feed	SYN	Synchronous Idle
DS	Digit Select	NAK	Negative Acknowledge	TM	Tape Mark
EM	End of Medium	NL	New Line	UC	Upper Case
ENQ	Enquiry	NUL	Null	VT	Vertical Tab
EO	Eight Ones				

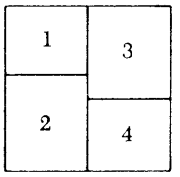
Special Graphic Characters

¢	Cent Sign	>	Greater-than Sign
.	Period, Decimal Point	?	Question Mark
<	Less-than Sign	`	Grave Accent
(	Left Parenthesis	:	Colon
+	Plus Sign	#	Number Sign
|	Logical OR	@	At Sign
&	Ampersand	'	Prime, Apostrophe
!	Exclamation Point	=	Equal Sign
$	Dollar Sign	"	Quotation Mark
*	Asterisk	~	Tilde
)	Right Parenthesis	{	Opening Brace
;	Semicolon	⌠	Hook
¬	Logical NOT	⌡	Fork
-	Minus Sign, Hyphen	}	Closing Brace
/	Slash	\	Reverse Slant
¦	Vertical Line	⌂	Chair
,	Comma	|	Long Vertical Mark
%	Percent		
_	Underscore		

Extended Binary-Coded-Decimal Interchange Code (EBCDIC)

The 256-position EBCDIC table, outlined by the heavy black lines, shows the graphic characters and control character representations for EBCDIC. The bit-position numbers, bit patterns, hexadecimal representations and card hole patterns for these and other possible EBCDIC characters are also shown.

To find the card hole patterns for most characters, partition the 256-position table into four blocks as follows:



Block 1: Zone punches at top of table; digit punches at left
Block 2: Zone punches at bottom of table; digit punches at left
Block 3: Zone punches at top of table; digit punches at right
Block 4: Zone punches at bottom of table; digit punches at right

Fifteen positions in the table are exceptions to the above arrangement. These positions are indicated by small numbers in the upper right corners of their boxes in the table. The card hole patterns for these positions are given at the bottom of the table. Bit-position numbers, bit patterns, and hexadecimal representations for these positions are found in the usual manner.

Following are some examples of the use of the EBCDIC chart:

Character	Type	Bit Pattern	Hex	Hole Pattern	
				Zone Punches	Digit Punches
PF	Control Character	00 00 0100	04	12 - 9	- 4
%	Special Graphic	01 10 1100	6C	0	- 8 - 4
R	Upper Case	11 01 1001	D9	11	- 9
a	Lower Case	10 00 0001	81	12 - 0	- 1
	Control Character, function not yet assigned	00 11 0000	30	12 - 11 - 0 - 9	- 8 - 1

Bit Positions
01 23 4567

Appendix K. Number Representation

Fixed-Point with Two's Complement

A fixed-point number is a signed value, recorded as a binary integer. It is called fixed-point because the programmer determines the fixed positioning of the radix point. In any length representation, the first bit position (0) holds the sign of the number, with the remaining bit positions used to designate the magnitude of the number. Positive fixed-point numbers are represented in true binary form with a zero sign bit. Negative fixed-point numbers are represented in two's-complement notation with a 1 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 0's for positive numbers, all 1's for negative numbers). Negative fixed-point numbers are formed in two's-complement notation by inverting each bit of the positive binary number and adding 1. For example, the true binary form of the decimal value (+26) is made negative (-26) in the following manner:

	S	*Integer*				
+26	0	000	0000	0001	1010	
Invert	1	111	1111	1110	0101	
Add 1					1	
– 26	1	111	1111	1110	0110	(Two's-complement form)

This is equivalent to subtracting the number

```
  0000 0000 0001 1010
  from
1 0000 0000 0000 0000
```

The following addition examples illustrate the two's-complement arithmetic. Only eight bit positions are used. All negative numbers are in two's-complement form.

1.
```
+57 =  0011 1001
+35 =  0010 0011
-----------------
+92 =  0101 1100
```

2.
```
+57 =  0011 1001
-35 =  1101 1101      No overflow.
-----------------
+22 =  0001 0110      Ignore carry; carry into high-order
                      position and carry out.
```

3.
```
+35 =  0010 0011
-57 =  1100 0111
-----------------
-22 =  1110 1010      Sign change only; no carry
```

4.
```
-57 =  1100 0111
-35 =  1101 1101      No overflow.
-----------------
-92 =  1010 0100      Ignore carry; carry into high-order
                      position and carry out.
```

5.
$$\begin{array}{rrl} -57 & = & 1100\ 0111 \\ -92 & = & 1010\ 0100 \\ \hline -149 & = & *0110\ 1011 \end{array}$$
*Overflow; no carry into high-order position but carry out.

6.
$$\begin{array}{rrl} +57 & = & 0011\ 1001 \\ +92 & = & 0101\ 1100 \\ \hline 149 & = & *1001\ 0101 \end{array}$$
*Overflow; carry into high-order position, no carry out.

The presence or absence of an overflow condition may be recognized by the condition of the carries.

- There is no overflow:
  1. If there is a carry into the high-order bit position and also a carry out (examples 2 and 4).
  2. If there is no carry into the high-order bit position and no carry out (examples 1 and 3).
- There is an overflow:
  1. If there is no carry into the high-order position but there is a carry out (example 5).
  2. If there is a carry into the high-order position but no carry out (example 6).

The following are 16-bit fixed-point numbers. The first is the largest 16-bit positive number and the last, the largest 16-bit negative number:

Number			*Decimal*		*S Integer*
2^{15}	-1	=	32,767	=	0 111 1111 1111 1111
2^0		=	1	=	0 000 0000 0000 0001
0		=	0	=	0 000 0000 0000 0000
-2^0		=	-1	=	1 111 1111 1111 1111
-2^{15}		=	-32,768	=	1 000 0000 0000 0000

Floating Point

Floating-point arithmetic simplifies the programming of computations in which the range of values used varies widely. It is called floating-point because the radix placement, or scaling is automatically maintained by the machine. The key to floating-point data representation is the separation of the significant digits of a number from the size (scale) of the number. Thus, the number is expressed as a fraction times a power of 16. A floating-point number has two associated sets of values. One set represents the significant digits of the number and is called the fraction. The second set specifies the power (exponent) to which 16 is raised and indicates the location of the binary point of the number. The two numbers (the fraction and exponent) are recorded in a single word or a double word. Since each of these two numbers is signed, some method must be employed to express two signs in an area that provides for a single sign. This is accomplished by having the fraction sign use the sign associated with the word (or double word) and expressing the exponent in excess-64 notation; that is, the exponent is added as a signed number to 64. The resulting number is called the *characteristic*. The characteristic can vary from 0 to 127, permitting the exponent to vary from -64 through 0 to +63. This provides a scale multiplier in the range of 16^{-64} to 16^{63} A nonzero fraction, if normalized, must be less than 1 and greater than or equal to 1/16, so the range covered by the magnitude (M) of a floating-point number is:

$$16^{-65} \leq M < 16^{63}$$

or more precisely:

In the short format:

$$16^{-65} \leq M \leq (1 - 16^{-6}) \times 16^{63}$$

In the long format:

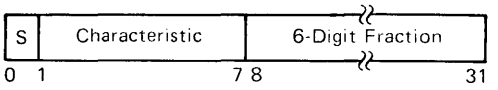
$$16^{65} \leq M \leq (1 - 16^{-14}) \times 16^{63}$$

In decimal terms:

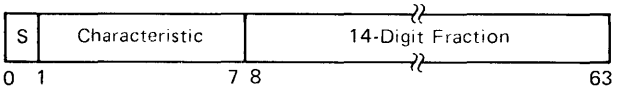
16^{-65} is approximately equal to 5.4×10^{-79}
16^{63} is approximately equal to 7.2×10^{75}

Floating-point data may be recorded in short or long formats. Each format uses a sign bit in position 0, followed by a characteristic in bit positions 1-7. Short floating-point operands contain the fraction in bit positions 8-31; long operands have the fraction in bit positions 8-63.

Short Floating-Point Number



Long Floating-Point Number



The sign of the fraction is indicated by a zero or one bit in position 0 to denote a positive or negative fraction, respectively.

With a given fraction length (6 or 14 digits), a floating-point operation provides the greatest precision if the fraction is normalized. A fraction is normalized when the high-order digit (bit positions 8, 9, 10, and 11) is nonzero. It is unnormalized if the high-order digit contains all zeros.

If normalization of the operand is desired, the floating-point instructions that provide automatic normalization are used. This automatic normalization is accomplished by left-shifting the fraction (four bits per shift) until a nonzero digit occupies the high-order digit position. The characteristic is reduced by 1 for each digit shifted.

Conversion Example

Convert the decimal number 149.25 to a short-format floating-point operand. (Appendix H provides tables for the conversion of hexadecimal and decimal integers and fractions.

1. The number is decomposed into decimal integer and decimal fraction:

   149.25 = 149 plus 0.25

2. The decimal integer is converted to its hexadecimal representation.

   $149_{10} = 95_{16}$

3. The decimal fraction is converted to its hexadecimal representation:

   $0.25_{10} = 0.4_{16}$

4. Combine the integral and fractional parts and express as a fraction times a power of 16 (exponent):

   $95.4_{16} = 0.954_{16} \times 16^2$

5. The characteristic is developed from the exponent and converted into binary:

   base + exponent = characteristic
   64 + 2 =66 = 1000010

6. The fraction is converted to binary and grouped hexadecimally:

   $0.954_{16} = 1001\ 0101\ 0100$

7. The characteristic and the fraction are stored in the short format. The sign position contains the sign of the fraction:

S	*Char*	*Fraction*
0	1000010	1001 0101 0100 0000 0000 0000

The following are sample normalized short floating-point numbers. The last two numbers represent the smallest and the largest positive normalized numbers:

Number		*Powers of 16*		*S*	*Char*	*Fraction*
1.0	=	$+1/16 \times 16^{1}$	=	0	100 0001	0001 0000 0000 0000 0000 0000
0.5	=	$+8/16 \times 16^{0}$	=	0	100 0000	1000 0000 0000 0000 0000 0000
1/64	=	$+4/16 \times 16^{-1}$	=	0	011 1111	0100 0000 0000 0000 0000 0000
0.0	=	$+0 \times 16^{-64}$	=	0	000 0000	0000 0000 0000 0000 0000 0000
−15.0	=	$-15/16 \times 16^{1}$	=	1	100 0001	1111 0000 0000 0000 0000 0000
5.4×10^{-79}	$\simeq$	$+1/16 \times 16^{-64}$	=	0	000 0000	0001 0000 0000 0000 0000 0000
7.2×10^{75}	$\simeq$	$(1 - 16^{-6}) \times 16^{63}$	=	0	111 1111	1111 1111 1111 1111 1111 1111

Glossary

This glossary includes definitions developed by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). This material is reproduced from the *American National Dictionary for Information Processing,* copyright 1977 by the Computer and Business Equipment Manufacturers Association, copies of which may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018.

Most entries in this glossary are defined as they apply to the 8100 Information System.

A

access control. The field of a translation-table entry that controls the types of storage accesses permitted during the fetching and execution of an instruction or during a channel I/O operation.

ACV. See *address control vector.*

adapter. Hardware that is generally required to transfer data and commands between the PCE and an I/O device.

address base. The field of an address control vector that designates the origin of a logical address space in the PCE address space. It is concatenated with a logical address during dynamic address relocation.

address control vector (ACV). The formatted information used to control dynamic address relocation and the activation of dynamic address translation.

address limit. The field of an address control vector that designates the maximum logical address in a logical address space. It is used to check the validity of a logical address during dynamic address relocation.

address-space origin. The field of an address control vector that designates the beginning location of a logical address space in the PCE address space. It is concatenated with the significant bits of a logical address during dynamic address relocation.

address-space size. The field of an address control vector that designates the size of a logical address space. It is used by the PCE to check the validity of a logical address during dynamic address relocation.

adjunct register. A 32-bit register used as storage for either an address control vector (ACV), an exception block index (EBI), or a protection key; only the low-order 16 bit positions are available to the program.

adjunct register group. All the adjunct registers available to the PCE.

adjunct register set. A set of eight adjunct registers located consecutively in the adjunct register group.

application mode. The mode of program execution that allows processing of all instructions, except those which are supervisor-privileged or I/O-privileged.

assembler. (ISO) A computer program used to assemble.

B

base address. Either the instruction address or the content of a general register from which a logical address is derived during instruction execution by combination with a displacement.

basic status register (BSTAT). A 1- or 2-byte register that contains adapter status information.

block address. The field in a translation-table entry that contains the common high-order bits of the real addresses associated with a 2048-byte block of physical main storage.

BSTAT. Basic status register.

byte operand. An eight-bit unit of data referenced as an operand of an instruction.

C

C-bit. One of the four condition indicators.

channel. The facility that controls the transmission of information between the PCE or main storage and an I/O device.

channel control vector (CHCV). The formatted information that specifies the controlling parameters, such as the channel I/O command, used during a channel I/O operation.

channel I/O (CHIO) burst transfer. That portion of a channel operation during which the channel and an I/O device adapter are logically connected for transferring information.

channel I/O command. The field of a channel control vector that directs a channel and an I/O device adapter to perform a channel I/O burst operation.

channel I/O (CHIO) operation. The transfer of data between main storage and an I/O device. The operation consists of one or more channel I/O burst operations. It is initiated by the I/O device adapter rather than by the PCE, and is controlled by the PCE's channel logic. PCE instruction execution is temporarily suspended while a CHIO operation is in progress.

channel mask. The one-bit mask used to suspend channel I/O operations.

channel pointer (CHP). The principal register containing the logical address used during a channel I/O operation.

channel pointer number. The field of a channel control vector that designates the channel pointer to be used during a channel I/O operation.

CHCV. See *channel control vector.*

CHIO. See *channel I/O (CHIO) operation.*

CHP. See *channel pointer.*

common mask. The eight-bit mask used to selectively enable or disable the dispatching of priority levels.

condition indicators. The four bits in a program status vector (PSV) that reflect the result of a previous arithmetic, logical, or I/O operation.

condition values. The values that are assigned to various combinations of the condition indicators and that may be used as mask values in conditional branching operations.

configuration. (1) (TC97) The arrangement of a computer system or network as defined by the nature, number, and the chief characteristics of its functional units. More specifically, the term *configuration* may refer to a hardware configuration or a software configuration. (2) The devices and programs that make up a system, subsystem, or network between the PCE or main storage and an I/O device.

current priority level. The number of the active or controlling priority level. Contrast with *last priority level.*

D

DAT. See *dynamic address translation.*

data area. A storage area used by a program to hold information.

DCV. See *diagnostic control vector.*

diagnostic control vector (DCV). An implementation-dependent register that provides system maintenance and initialization functions.

displacement. The field of an instruction containing a signed or unsigned value that is combined with a base address to generate a logical address during instruction execution.

dual mode. Normal mode of processing for dual-PCE processors, with both PCEs active.

dual program status vectors. The association of two program status vectors with each priority level, used to facilitate the definition of both an application program and a supervisory program on a single priority level.

dynamic address relocation. The mapping of logical storage addresses to relocated storage addresses.

dynamic address translation (DAT). The mapping of relocated storage addresses to real storage addresses.

E

EBI. See *exception block index registers.*

EIRV. See *error interrupt request vector.*

error interrupt request vector (EIRV). The formatted information used to indicate an interrupt request generated by the PCE when a system-check condition is detected, and to identify the system-check condition.

exception block index (EBI) registers. Fifteen registers, each associated with a PSV/ACV pair and used during dynamic address translation. The EBI is used to store the block (translation-table) index of the address in error when an access or separation exception occurs during a main storage operation.

F

floating-point register. A 64-bit register used for floating-point operation.

floating-point register group. All of the floating-point registers provided with the floating-point feature.

floating-point register set. A set of four floating-point registers located consecutively in the floating-point register group.

floating-point register set number. The field in a floating-point status vector that designates the number of the floating-point register set assigned to a priority level.

floating-point status vector (FSV). The formatted information used to allocate floating-point registers, to control exception masking, to control precision, and to hold and indicate floating-point check and program-exception conditions related to floating-point operations.

FSV. See *floating-point status vector.*

G

general register. A 32-bit register, in the primary or secondary register set, generally used for storage-address modification and generation, fixed-point (binary) arithmetic, and logical (boolean) operations.

H

H-bit. One of the four condition indicators.

halfword (HW). Two bytes of information.

halfword operand. A 16-bit unit of data referenced as an operand of an instruction.

hex. hexadecimal

HW. See *halfword.*

I

ICE. See *Interrupt Control Element.*

initial program load (IPL). (1) The initialization procedure that causes an operating system to begin operation. (2) The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction.

input/output (I/O). (1) (ISO) Pertaining to a device whose parts can be performing an input process and an output process at the same time. (2) Pertaining to either input or output, or both.

instruction address. The logical address that is used to fetch an instruction.

instruction address modifier. The bit that indicates whether the instruction address designates the starting location, or two bytes beyond it, of the instruction being executed when a program-exception or system-check interruption occurs.

interrupt control element (ICE). Logic that controls communication between the two PCEs in dual-PCE processors.

interrupt request. A request for processing on a particular priority level. It may be generated by the active program, the PCE, or an I/O device.

I/O. See *input/output.*

I/O interrupt request vector (IOIRV). The formatted information used to indicate an interrupt request generated by an I/O device.

IOIRV. See *I/O interrupt request vector.*

I/O mode. The mode of program execution that allows processing of all instructions except those which are supervisor-privileged.

I/O-privileged instruction. An instruction that may be executed in I/O, supervisor, or master modes but not in application mode.

IOIRV. See *I/O interrupt request vector.*

IPL. See *initial program load.*

K

KDO. Control direct out.

L

last priority level. The number of the last (most recent) priority level that was active prior to dispatching the current program status vector. Contrast with *current priority level.*

lock. See *translation lock* and *translation lock table.*

logical address. The storage address that is either supplied to or by a program during the fetching and execution of an instruction, or is used as a channel pointer during a channel I/O operation. Contrast with *relocated address.*

logical address space. The set of logical addresses numbered sequentially from zero to one less than the address limit. See also *address limit.*

logical storage. The concept of storage space that may be regarded as addressable main storage by a program or channel I/O operation in which logical addresses are mapped into real addresses.

M

master mask. A one-bit mask used to suspend the dispatching of a new priority level.

master mode. The mode of program execution that allows processing of all instructions and permits overriding store-protection and execution-protection access control.

P

PAV. See *program activation vector.*

PCE. See *processing and control element.*

PCE address space. The set of relocated addresses numbered sequentially from zero to the maximum available address.

PEC. See *program exception code.*

PIC. See *program information code.*

PIO. See *programmed I/O.*

PIRV. See *programmed interrupt request vector.*

primary PSV. One of two program status vectors (PSVs), associated with each priority level, normally used for the definition of a supervisory program.

primary register set. One of two principal register sets assigned to a program for use as general registers. See also *secondary register set.*

primary register set number. The field in a program status vector that designates the number of the primary register set.

principal register. A 32-bit register used as a general register, as storage for half of a program status vector, or for storage of a channel pointer.

principal register group. All principal registers available to the PCE.

principal register set. A set of eight principal registers located consecutively in the principal register group.

priority level. A number ranging from 0 to 7 that designates a relative precedence among interrupt requests, so that processing on one level may be temporarily suspended when an interrupt request is generated for a level of higher priority (lower number).

processing and control element (PCE). The part of the processor that contains the sequencing and processing controls used for instruction execution, interruption control, dynamic address transformation, and other control and processing functions.

program activation vector (PAV). The formatted information used to control which of two program status vectors is introduced when a new priority level is made active. See also *dual program status vectors.*

program exception. The condition recognized by the PCE resulting from execution of a program, including the improper specification or use of instructions, operands, or control information.

program exception code (PEC). A four-bit code that identifies the cause of a program exception.

program information code (PIC). A field in a program status vector that either contains the program exception code or indicates that a Call PSV instruction was executed.

program mode (PM). The field in a program status vector that controls which instructions may be executed by the associated program. See also *application mode, I/O mode, master mode,* and *supervisor mode.*

program status vector (PSV). The formatted information used to control the order in which instructions are executed, to allocate general registers, and to hold and indicate the status of the PCE in relation to a particular program.

programmed interrupt request vector (PIRV). The formatted information used to indicate an interrupt request generated by the executing program.

programmed I/O (PIO) address. The information specified as an operand of an I/O instruction that identifies the I/O device adapter to be selected for a programmed I/O (PIO) operation.

programmed I/O (PIO) command. The information specified as an operand of an I/O instruction that directs an I/O device adapter to perform a programmed I/O operation.

programmed I/O (PIO) operation. The transfer of data between the PCE and an I/O device as part of the execution of an I/O instruction. The I/O instruction designates the address of the I/O device adapter, the command to be performed, and the register into or from which the data is transferred.

protection key. One of 80 8-bit registers associated with a PSV/ACV and CHP/ACV pair and used in conjunction with the 8-bit translation lock to access addresses within that 2K-byte block. See also *translation lock.*

PSV. See *program status vector.*

R

real address. The address of a physical main storage location.

register (ISO). A storage device having a specified storage capacity such as a bit, a byte, or a computer word, and usually intended for a special purpose.

relocated address. The address in the PCE address space that is derived during dynamic address relocation by concatenating the high-order bits of the address base with the low-order bits of the logical address. Synonym for *real address* when dynamic address translation is not active.

result condition. One of five logical entities that describe the result of arithmetic, logical, or I/O operations. Each result condition has two possible states: indicated or not-indicated.

S

SCF. See *system control facilities.*

secondary PSV. One of two program status vectors, associated with each priority level, normally used for the definition of an application program.

secondary register set. One of two principal register sets assigned to a program for use as general registers. See also *primary register set.*

secondary register set number. The field in a program status vector that designates the number of the secondary register set.

separation protection. Provides a method to logically separate programs and/or channel I/O operations within a logical address space through the use of translation locks and protection keys.

single mode. The mode that exists in dual-PCE processors when only one PCE is operational.

supervisor mode. The mode of program execution that allows processing of all instructions.

supervisor-privileged instruction. An instruction that may be executed in supervisor or master modes but not in application or I/O modes.

suspended. The state of a PCE in dual-PCE processors when its master mask is reset after the other PCE has reset its master mask. While in this state, instruction execution cannot occur.

system check. An error that is detected by the PCE, channel, or floating-point feature and is identified in the error interrupt request vector. The error may be due to equipment malfunctioning, an I/O check (such as an invalid PIO command), an exception related to a channel I/O operation, or a program exception recognized when a primary PSV is active.

system control facilities (SCF). Control facilities that provide system functions, such as I/O interrupt request identification, programmable assignment of I/O devices to priority levels, execution of direct-control instructions, initial program load (IPL), and system and I/O reset.

T

translated address. The real address that is derived during dynamic address translation by concatenating the block address from a translation-table entry with the low-order 11 bits of the relocated address.

translation-control bit. The bit in an address control vector used to activate dynamic address translation.

translation lock. An 8-bit lock associated with each 2K-byte block of logical storage.

translation lock table. A table that provides an 8-bit lock for each 2K-byte block of logical storage. See also *translation lock.*

translation table. The table that maps blocks of relocated addresses to blocks of real addresses during dynamic address translation.

translation-table entry. An entry in the translation table containing access control information and the block address that designates a 2048-byte block of physical main storage.

V

V-bit. One of the four condition indicators.

vector. One or more related fields of information in a specified format, associated with the control of a PCE, a channel, or floating-point facility.

W

wait. The state of the PCE when it cannot fetch or execute instructions because no interrupt request is present for an enabled priority level and the master mask is set to 1.

word. Four bytes of information.

word operand. A 32-bit unit of data referenced as an operand of an instruction.

Z

Z-bit. One of the four condition indicators.

Index

NOTE: STAPLES CAN CAUSE PROBLEMS WITH AUTOMATIC MAIL SORTING EQUIPMENT.
PLEASE USE PRESSURE SENSITIVE OR OTHER GUMMED TAPE TO SEAL THIS FORM.

IBM 8100 Information System
Principles of Operation

Order No. GA23-0031-4

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.*

How did you use this publication?

[ ] As an Introduction
[ ] As a text (student)
[ ] As a reference manual
[ ] As a text (instructor)
[ ] For another purpose (explain) _____

_____

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

What is your occupation? _____

Newsletter number of latest Technical Newsletter (if any) concerning this publication: _____

If you wish a reply, give your name and address: _____

_____

_____

IBM branch office serving you _____

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 title page.)

GA23-0031-4

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 52Q
Neighborhood Road
Kingston, New York 12401

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM®

Printed in U.S.A. GA23-0031-4

IBM 8100 Information System
Principles of Operation

Order No. GA23-0031-4

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.*

How did you use this publication?

[ ] As an Introduction
[ ] As a text (student)
[ ] As a reference manual
[ ] As a text (instructor)
[ ] For another purpose (explain) _____

_____

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

What is your occupation? _____

Newsletter number of latest Technical Newsletter (if any) concerning this publication: _____

If you wish a reply, give your name and address: _____

_____

_____

IBM branch office serving you _____

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 title page.)

NOTE: STAPLES CAN CAUSE PROBLEMS WITH AUTOMATIC MAIL SORTING EQUIPMENT.
PLEASE USE PRESSURE SENSITIVE OR OTHER GUMMED TAPE TO SEAL THIS FORM.

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 52Q
Neighborhood Road
Kingston, New York 12401

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

Printed in U.S.A. GA23-0031-4

IBM 8100 Information System
Principles of Operation

Order No. GA23-0031-4

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.*

How did you use this publication?

[ ] As an Introduction
[ ] As a reference manual
[ ] For another purpose (explain) _____

[ ] As a text (student)
[ ] As a text (instructor)

_____

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

What is your occupation? _____

Newsletter number of latest Technical Newsletter (if any) concerning this publication: _____

If you wish a reply, give your name and address: _____

_____

_____

IBM branch office serving you _____

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 title page.)

NOTE: STAPLES CAN CAUSE PROBLEMS WITH AUTOMATIC MAIL SORTING EQUIPMENT.
PLEASE USE PRESSURE SENSITIVE OR OTHER GUMMED TAPE TO SEAL THIS FORM.

GA23-0031-4

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 52Q
Neighborhood Road
Kingston, New York 12401

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM®

Printed in U.S.A. GA23-0031-4

Technical Newsletter

This Newsletter No.	GN31-1498
Date	25 Oct 1984
Base Publication No.	GA23-0031-4
File No.	8100-00
Previous Newsletters	None

8100 Information System
Principles of Operation

This TNL provides information about the 8150 Model A and Model B enhancements. A change to the text or to an illustration is indicated by a vertical line to the left of the change.

This TNL provides replacement pages for Appendix G of the base manual, GA23-0031-4. Remove pages from the base manual and replace with the attached TNL pages as follows:

Remove	Insert
G1 to G4	G1 to G4

Summary of Amendment

This TNL provides information about the latest 8150 models.

Note: *File this cover letter just before the back cover of the* Principles of Operation *manual. Failure to do so will prevent you from tracking changes to the manual, if the need occurs.*

IBM Corporation, Department 52Q, Kingston, New York 12401

Printed in U.S.A.

IBM

IBM 8100 Information System Principles of Operation

Printed in U.S.A.

GA23-0031-4
File No. 8100-00