LARGE SYSTEMS

ASSEMBLY
INSTRUCTIONS
DPS 8000

Bull 

GCOS 8

ASSEMBLY INSTRUCTIONS DPS 8000

SUBJECT

Description of the Assembly Instructions for the DPS 8000 Information System.

SOFTWARE SUPPORTED

GCOS 8 Software Release 2500

DATE

March 1987

ORDER NUMBER

DZ51-00

Worldwide
Information
Systems

Bull

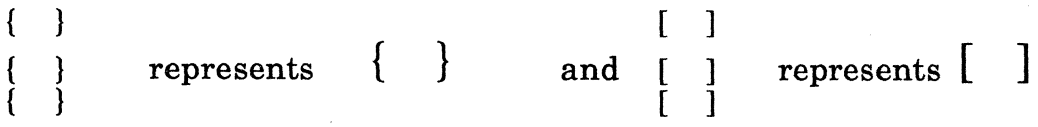
PREFACE

This manual contains information that enables the user to code programs in symbolic machine language which is then translated into binary machine instructions.

This manual is directed to users who are experienced in coding within the environment of a large-scale computer installation. Considerable knowledge and practical experience is required in the use of address modification with indirection, hardware indicators, fault interrupts and recovery routines, macro operations, pseudo-operations, and other features normally encountered in a large computer with a flexible instruction repertoire under control of a master executive program. It is assumed that the user is familiar with the two's complement number system.

This manual includes the processor capabilities, modes of operation, detailed descriptions of machine instructions, virtual memory addressing, paging, and the representation of data. It should prove useful to programmers who are responsible for analyzing conditions that cause system failures.

In this document, multiple vertical braces and brackets should be assumed to be a single brace or bracket; for example:



File No.: 1V13, 1313

DZ51-00

LISTING AND CORRECTING DOCUMENTS

The Problem Analysis Solution System (PASS) data base is an online tool that provides direct communications between Bull software development organizations and Bull customers. Documentation-related transactions available to customers via PASS include those which:

- Generate a list of all software documents published for the current Software Release.
- Prepare Software Technical Action Requests (STARs) regarding documentation discrepancies.

Logon procedures for these functions and procedures for using PASS can be obtained by contacting the Bull Technical Assistance Center (TAC).

DOCUMENT LISTING

A list of all GCOS 8 System software documents published for this Software Release and available through the Bull CSO Marketing and Sales Order Entry (telephone 1-800-343-6665) can be displayed via the NEWS facility of PASS. The document lists are available via the PASS meeting SWDOC_AVAILABILITY.

DOCUMENTATION CORRECTIONS

Customers can submit documentation error reports via the PASS online STAR Maker facility. Responses to STARs, as well as other documentation changes, also are contained on PASS. (Documentation corrections contained on PASS may apply to prior Software Releases as well as to the current Software Release.)

In addition, corrections to documents will be entered on the PASS data base. Query PASS periodically to determine if any corrections exist. Corrections documented on PASS, if applicable to the next release of the software, will be incorporated into the next update of the manual.

CONTENTS

ILLUSTRATIONS

ILLUSTRATIONS (cont)

TABLES

SECTION 1

INTRODUCTION

This manual contains a set of machine instructions used on Honeywell Bull hardware and operating systems. The systems are highly modular, allowing system configuration to be matched to the work load mix. This section describes the essential characteristics of the central processors for these systems.

Each processor module in the system has full program execution capability. The processors conduct all actual computational processing (data movement, arithmetic, logic, comparison, and control operations) within the information system. The processor communicates only with the system controller (DPS 8000: SCU, System Control Unit) and associated memory. The processors contain several special features that make significant contributions to multiprogramming, high throughput, and rapid turnaround. These features are under the control of the operating system which maintains automatic supervision and complete control of the multiprogramming/multiprocessing environment.

PROCESSOR FEATURES

A processor contains the following general features:

1. Memory protection to place access restrictions on specified segments
2. Capability to interrupt program execution in response to an external signal (e.g., I/O termination), to save processor status and to restore the status at a later time without loss of program continuity
3. Capability to fetch instructions and to buffer instructions
4. A four-stage pipelined instruction development for greater performance
5. Fully interlaced store units addressable by a given SCU
6. Ability to hold recently referenced operands and instructions in a 64K high-speed cache memory
7. An Extended (ES) mode that uses 36-bit addressing includes a set of general register-to-register instructions
8. Real memory configurations of up to 256 megawords are supported.

9. Quad-precision arithmetic operations for which the exponents are handled as powers of 16

Pipeline Architecture Of The DPS 8000

The four-stage pipeline processor consists of the following cycles:

A cycle: Effective address calculation and virtual address calculation are performed

V cycle: Virtual address to real address translation and bound checking, access checking (read, write permission, etc.) are performed

C cycle: Memory is accessed (cache) using the real memory addresss

E cycle: Instruction is executed by firmware control

One instruction execution completes via four cycles. The maximum instruction rate is attained when the processor is executing basic instructions (one memory access and one execution cycle). Because the processor operates as a four-stage pipeline, a new instruction can be issued before the prior one is completed, thereby reducing the effective execution time.

Faults And Interrupts

The processor detects illegal instruction usages, faulty communication with main memory, programmed faults, certain external events, and arithmetic faults. Many of the processor fault conditions are deliberately caused by the software and do not necessarily involve error conditions. The processor communicates with the other system modules (I/O processors and other processors) by setting and answering external interrupts. When the processor responds to a fault or interrupt, control is transferred to an operating system module via an interdomain transfer using an entry descriptor obtained from a fixed memory location.

The locations in real memory containing the entry descriptors for interrupt, fault, and system entry (PMME) are as follows:

Type	Location
Interrupt	30-31 (octal)
Fault	32-33 (octal)
System Entry	34-35 (octal)
Backup	40-41 (octal)

Interrupts and certain low-priority faults are recognized only at specific times during program execution. If, at these times, bit 28 in the instruction word is set ON, the trap is inhibited and program execution continues. The interrupt or fault signal is saved for future recognition and is reset only when the trap is recognized.

Connect/Interrupt Mechanism

On a connect to the IMX, the software points to a logical channel mailbox that resides anywhere in main memory. The mailbox is required to be 24 words, beginning at a 0-modul0-8 address. The operating system is responsible for placing specific information into the first eight words.

This mailbox serves as the primary intercommunication vehicle between the IMX and the CPU. Software specifies the (relative) starting location of the mailbox as the effective address of the connect instruction (CIOC). Normal CPU address preparation converts this to a real memory address, which is then used by the IMX.

Successive I/O operations to the same logical channel can be issued via a linked mailbox feature available through IMX's. However, once a connect has been issued by the software, it is the responsibility of the operating system to not issue another connect directed to the same logical channel until the current one is completed or a "lost interrupt" timeout has occurred.

All 128 channels (numbered 0-127) are data channels except channel numbers zero and three. Channel three is used for two-way communication between the CPU and IMX maintenance system (MCA). Channel zero is normally declared invalid, to avoid confusion that would otherwise exist in the operating system as to whether a given channel number field is zero, or the field is currently unused.

The CPU automatically directs the connect command to the "control" SCU. If the system configuration includes two SCU's (i.e., tandem), then the SCU which is designated as "control" is the one which processes all connects and interrupts for the operational system. The control SCU then adds a connect word pair to the destination port's connect queue and notifies the port that a connect is present in its queue. The IMX reads the contents of the queue with the Read Connect Words command instruction (RCW).

An interrupt queue mechanism is used in the DPS 8000 system that allows for up to 256 simultaneous entries for each of eight interrupt levels. Thus, the SCU maintains a queue for each interrupt level. Levels one and seven are for fault and special interrupts, respectively. The interrupt level for marker/terminate interrupts are specified at connect time in the mailbox (GCOS uses levels 5 and 3, respectively).

The control SCU sends an interrupt present signal to all CPU's that are unmasked for this interrupt level (each CPU initializes and modifies its own masks independently). The SCU sends an accept signal to the candidate CPU selected, and automatically shuts off all further interrupt present signals by masking a unique system-wide "all mask".

The CPU, selected by the SCU to process the interrupt, transfers to the operating system interrupt handler by executing an interdomain CALL version of the CLIMB instruction, using the entry descriptor at location 30-31 (octal). The software interrupt handler uses the RIW instruction for each pair of interrupt words (one doubleword interrupt queue entry). The next interrrupt pair is selected from the highest priority (i.e., lowest numbered), unmasked level, and inserted into the AQ register. When no more entries are available at any level that is unmasked for this CPU, then the AQ register will contain all zeros.

The operating system examines the channel mailbox for status information. On terminate or marker type interrupts, status returns are automatically stored in the channel mailbox. Up to eight words of peripheral extended status are likewise stored.

Online Processor Tests

The PATROL feature (Processor Activity Test Runs On Line) is implemented as firmware in its own unique CPU memory. PATROL runs test programs and reports status to the maintainance interface.

OPERATING MODES

Three types of modes determine the operation of the CPU.

- Privileged Master, Master, and Slave modes which determine the processor mode of operation
- NS and ES (Non-extended/Extended) modes which determine whether 18-bit or 36-bit registers are used and determine the method to be used to generate effective and virtual addresses
- Memory addressing modes

Processor Modes Of Operation

The three processor modes of operation are Privileged Master mode, Master mode, and Slave mode. The master mode bit in the indicator register, the privileged bit in the instruction segment register (ISR), and the housekeeping bit in the page table word (PTW) for the instruction define these processor modes.

The status of the determinants for each mode is shown in Table 1-1.

Table 1-1. Status Of Processor Mode Determinants

	Processor Modes [a]		
Determinants	Privileged Master	Master	Slave
Master Mode Bit in Indicator Register (bit 28)	ON	ON	OFF
Privileged Bit in Instruction Segment Register (bit 26)	ON	OFF	OFF
Housekeeping Bit in Page Table Word for the Instruction (bit 32)	ON [b]	ON/OFF	OFF

[a] All other combinations are illegal and result in a Class 1 Security Fault.

[b] When working space zero is referenced, the housekeeping bit is assumed to be ON and the processor addresses memory through absolute mode page tables.

A fault or an interrupt causes the processor to enter Privileged Master mode. If the processor is in Privileged Master mode, an instruction can change to Master mode by transferring to a segment marked non-privileged. The reverse is also true when transferring to a segment marked privileged. The use of a CLIMB instruction between Master and Privileged Master modes, like the transfer, not only allows a change of processor execution modes but also a change of domains.

The Master mode bit in the indicator register can be turned ON as follows:

1. Occurrence of an interrupt or a fault
2. Execution of the PMME version of the CLIMB instruction, which causes a system entry
3. Execution of the OCLIMB version of the CLIMB instruction where the master mode bit of the restored indicator register is ON

The following mode-dependent processor functions are listed by mode. None of these functions are permitted in Slave mode.

Functions allowed in Master and Privileged Master modes:

1. Accessing through working space register zero
2. Reading operands from a housekeeping page of segment descriptor type T = 0, 2, 4, 6, 12, or 14
3. Executing instructions from housekeeping pages of type T = 0 segments
4. Executing a CLIMB (ICLIMB or GCLIMB) not invoking a system entry option (PMME)
5. Executing a transfer to a privileged executable segment

Functions allowed only in Privileged Master mode:

1. Executing Privileged Master mode instructions (e.g., load working space registers)
2. Executing Privileged Master mode options of the LDDn, LDPn, or CLIMB instructions, such as copying the safe store register (SSR) to a descriptor register (DRn)
3. Writing on housekeeping pages of type T = 0, 2, 4, 6, 12, or 14 segments, using instructions other than CLIMB, SDRn, STDn

Non-Extended/Extended Modes

The NS (Non-extended) and ES (Extended) modes are specified with bit 24 of the Instruction Segment Register (ISR).

- When ISR bit 24 = 0 NS mode.
- When ISR bit 24 = 1 ES mode.

ISR bit 24 may be altered only with the CLIMB instruction.

Processor operations differ between NS and ES modes for the following:

- o The number of bits in the index and the address registers
- o The method used to generate effective address
- o The execution of some instructions
- o Additional register instructions available in ES mode

Memory Addressing Modes

Three types of memory addressing exist in the DPS 8000.

1. **Virtual memory** which is mapped to a **real** (physical) memory address
2. **Absolute mode** which is used only when **Working space zero** is referenced
3. **Reserved memory** which is reserved for special use

VIRTUAL MEMORY PAGING

Virtual memory paging mode is an integral part of the address translation process for mapping a virtual memory address to a real memory address. Each of the 512 working spaces (WS) is supported by one page table (PT) or by a section table (SCT) that references multiple page tables.

The location of a PT supporting a working space (WS) is indicated by a 9-bit working space number (WSN) that indexes the 512-word page table directory called the working space page table directory (WSPTD). This directory contains the real memory address of the supporting page table. Words in the WSPTD are called page table directory words (PTDW), and words on the page table are called page table words (PTW). The location of a WSPTD is indicated by the page directory base register (PDBR).

The location of the SCT supporting a given WS is indicated by a 9-bit WSN that also indexes the page table directory (WSPTD). The SCT consists of up to 4K words and includes the real memory address of the page table. The individual words in the SCT are called page table base words (PBW). The effect of SCTs is seen when paging is performed; these page tables are distributed throughout memory.

ABSOLUTE MODE

The processor utilizes the absolute addressing mode each time working space number zero is referenced. However, the virtual address is not mapped to a real address; it is used as the real address with a maximum size limitation of 2**28 words (256 megawords). Any time a working space other than zero (WSN=0) is referenced, the processor uses the paging mode.

To use the absolute addressing mode, the processor must be in Privileged Master mode. The master mode bit in the indicator register and the privileged bit in the instruction segment register must be ON. If these two conditions are not met, any attempted reference to WSN 0 results in a Command fault. The housekeeping bit is assumed ON when WSN 0 is referenced.

RESERVED MEMORY SPACE

Reserved memory space is defined by space above the Reserved Memory Base Register. This page is not represented in the Memory Utilization Table (MUT) and is addressable only in absolute mode.

INTERVAL TIMER

The processor contains a timer that provides a program interrupt (timer runout fault) at the end of a variable interval. The timer is loaded by the operating system and can be set to a maximum of approximately four minutes total elapsed time.

SECTION 2

REPRESENTATION OF DATA

FORMATS

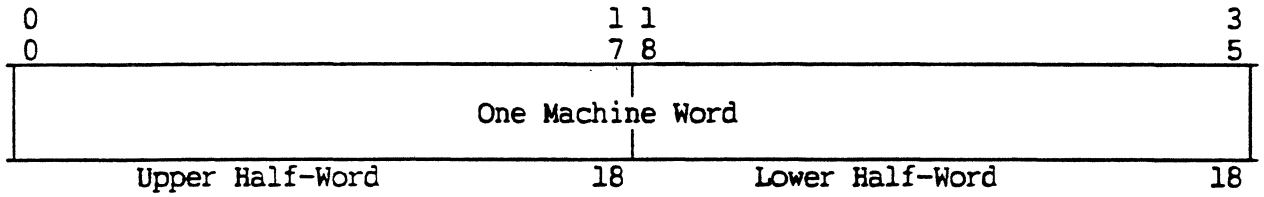
The processor is functionally organized to process 36-bit groupings of information called words. Special features are also included for ease in manipulating 4-bit groups, 6-bit groups, 9-bit groups, 18-bit groups, 72-bit double-precision, and 144-bit quad-precision groups. These bit groupings are used by the hardware and software to represent a variety of forms of information.

POSITION NUMBERING

The numbering of bit positions, character positions, words, etc., starts with zero and increases from left to right as in conventional alphanumeric text. Bit zero is the most-significant bit and the right-most bit is the least-significant bit.

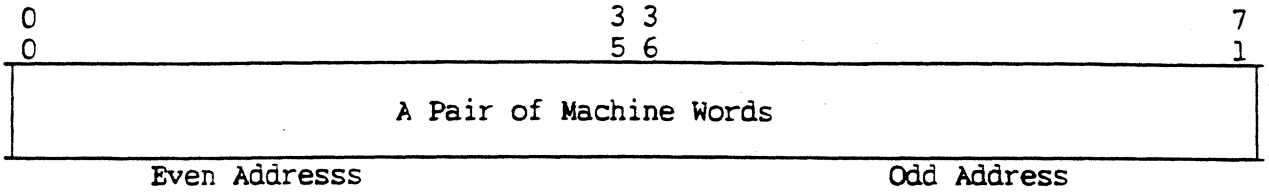
THE MACHINE WORD

The machine word consists of 36 bits arranged as follows:



Data transfers between the processor and memory are double-word-oriented; 36 bits are used at a time for single-precision data and two parallel 36-bit word are used for double-precision data. When words are transferred to a memory unit, Error Detection and Correction (EDAC) bits are added to each word pair before the words are stored. When words are requested from a memory unit, the EDAC bits are read from memory, verified, and removed before sending the word pair to the processor.

The processor has many built-in features for efficient transferring and processing of pairs of words. When a pair of words is transferred to or from memory, their addresses are an even number and the next higher odd number. A pair of words is arranged as follows.



In an instruction intended for handling pairs of machine words, either of the two addresses may be used as the effective address (Y). Thus,

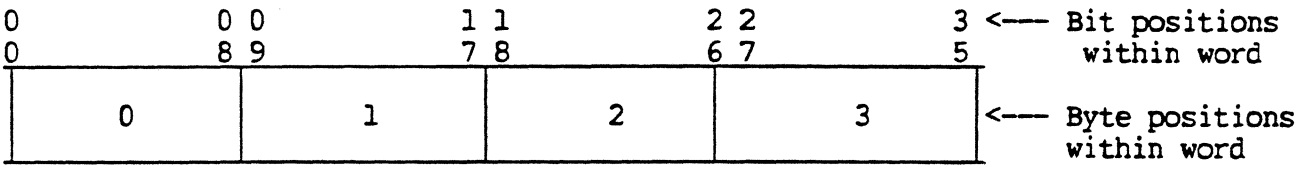
If Y is even, the pair of locations (Y, Y+1) is accessed. If Y is odd, the pair of locations (Y-1, Y) is accessed. The term "Y-pair" is used for each pair of addresses. Preferred coding practice refers to the even address; the GMAP assembler issues a warning diagnostic if Y is odd in an instruction intended for handling pairs of machine words.

CHARACTER-STRINGS

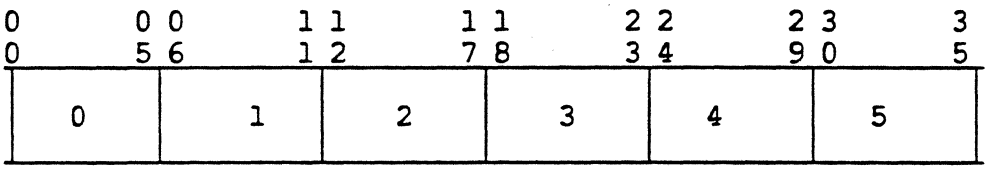
Character Positions

Alphanumeric data is represented by 9-bit, 6-bit, or 4-bit characters. A machine word contains either four, six, or eight characters, respectively. The character positions within the word are as follows:

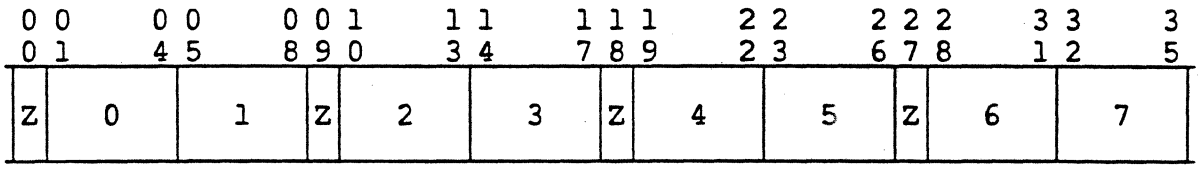
9-Bit Character (Bytes):



6-Bit Characters:



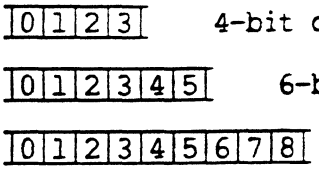
4-Bit Characters (Packed Decimal):



The Z represents the bit value 0; other numbers in the fields represent the character positions.

Bit Positions

Bit positions within a character are as follows:



Thus, both bit and character positions increase from left to right as in normal reading.

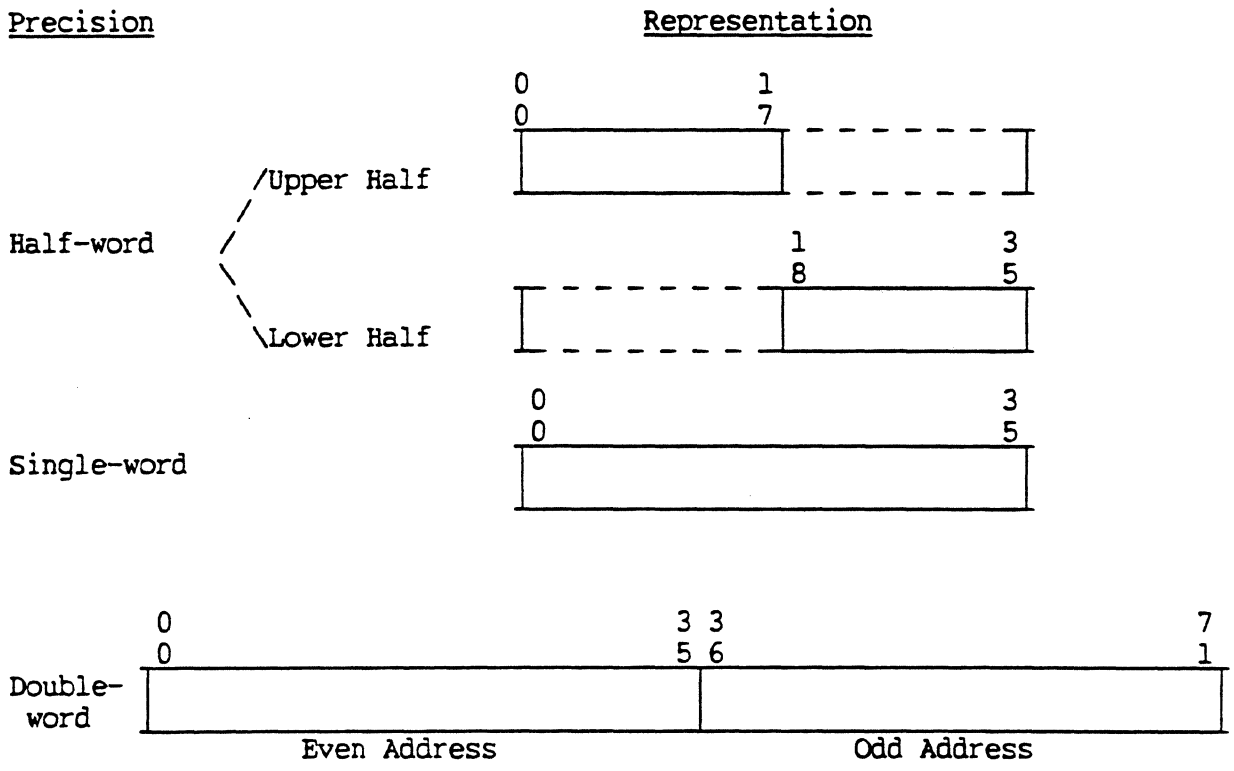
LITERALS

For information on literals refer to the GCOS 8 OS GMAP User's Guide.

BINARY NUMBERS

Fixed-Point Numbers

Binary fixed-point numbers are represented with half-word, single-word, and double-word precision as shown below.



Instructions can be divided into two groups according to the way in which the operand is interpreted: the "logic" group and the "algebraic" group.

For logic operations, operands and results are regarded as unsigned, positive binary numbers. In the case of addition and subtraction, the occurrence of an overflow is indicated by the carry out of the most significant (leftmost) bit position:

1. Addition - If the carry out of the leftmost bit position equals 1 (Carry indicator ON), the sum is above the range.

2. Subtraction - If the carry out of the leftmost bit position equals 0 (Carry indicator OFF), the difference is below the range.

In the case of comparisons, the zero and carry indicators show the relation.

For algebraic operations, operands and results are regarded as signed binary numbers, and the leftmost bit is used as a sign bit (a 0 being plus and 1 minus). When the sign is positive, all the bits represent the real value of the number; when the sign is negative, they represent the two's complement of the real value of the number.

In the case of addition and subtraction, the occurrence of an overflow is indicated by the carries into and out of the leftmost bit position (the sign position). If the carry into the leftmost bit position does not equal the carry out of that position, then overflow has occurred. If overflow has been detected and if the sign bit equals 0, the result is below range; if with overflow the sign bit equals 1, the result is above range.

In integral arithmetic, the location of the decimal point is assumed to the right of the least significant bit position; that is, depending on the precision, to the right of bit position 35 or 71 (17 for upper half-word).

The number ranges for the various cases of precision, interpretation, and arithmetic are given in Table 2-1.

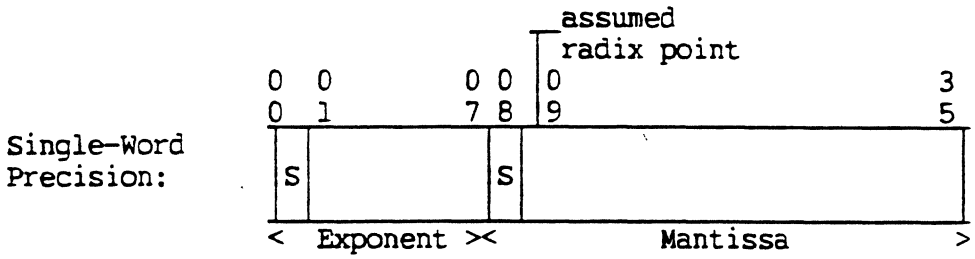
Table 2-1. Ranges Of Fixed-Point Numbers

Inter-pretation	Arithmetic	Precision		
		Half-word (X_n, $Y_{0...17}$)	Single-Word (A,Q,Y)	Double-Word (AQ,Y-pair)
Algebraic	Integral	$-2^{17} \leq N \leq (2^{17}-1)$	$-2^{35} \leq N \leq (2^{35}-1)$	$-2^{71} \leq N \leq (2^{71}-1)$
	Fractional	$-1 \leq N \leq (1-2^{-17})$	$-1 \leq N \leq (1-2^{-35})$	$-1 \leq N \leq (1-2^{-71})$
Logic	Integral	$0 \leq N \leq (2^{18}-1)$	$0 \leq N \leq (2^{36}-1)$	$0 \leq N \leq (2^{72}-1)$
	Fractional	$0 \leq N \leq (1-2^{-18})$	$0 \leq N \leq (1-2^{-36})$	$0 \leq N \leq (1-2^{-72})$

Floating-Point Numbers

Floating-point numbers are represented with single-word and double-word precision. The upper 8 bits represent the integral exponent to the base 2 in two's complement form, and the lower 28 or 64 bits represent the fractional mantissa in two's complement form.

The format for a floating-point number is:



where S = sign bit

Before performing binary floating-point additions or subtractions, the processor aligns the number that has the smaller exponent. To maintain accuracy, the lowest permissible exponent of -128, together with the mantissa of zero, has been defined as the machine representation of the number zero (which has no unique floating-point representation). Whenever a floating-point operation yields an untruncated resultant mantissa equal to zero (71 bits plus sign because of extended precision), the exponent is automatically set to -128.

Hexadecimal Floating-Point Numbers

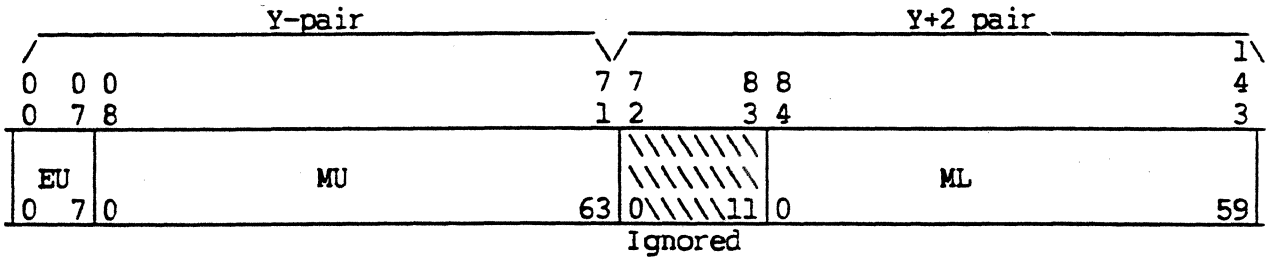
The hexadecimal option may be used in floating-point operations to declare hexadecimal constants, either explicitly or by default. The term hexadecimal refers to a floating-point format where the mantissa is a binary number, while the exponent represents a power of 16 (2**4). The mantissa is shifted by the number of places for 4-bit groups as required by the exponent.

The hexadecimal floating-point mode is enabled only when bit 32 of the Indicator Register is set to 1 and bit 33 of the mode register is set to 1. After the hexadecimal floating-point mode is requested, the user controls the floating-point mode via the Indicator Register. If the bit 32 of the Indicator Register is not set to 1, the floating-point mode will be binary.

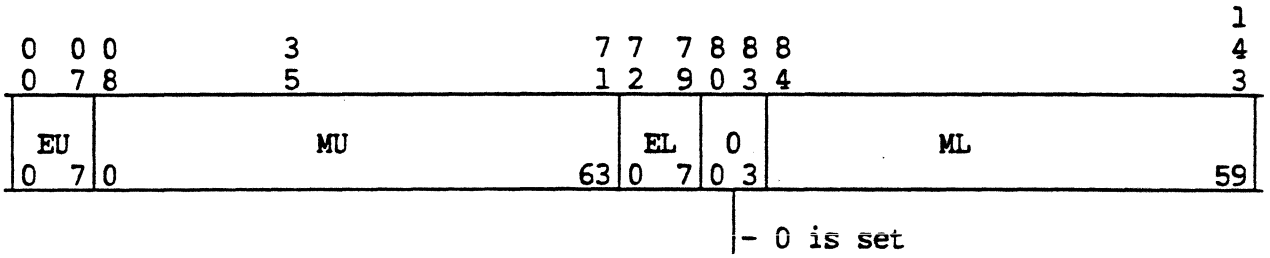
Quadruple-Precision Numbers

The data format used in quadruple-precision arithmetic is illustrated below. Notice that the format of data to be used in an operation is somewhat different from that of data to be stored after the operation.

The format for data when an operand in main memory is used as arithmetic data:

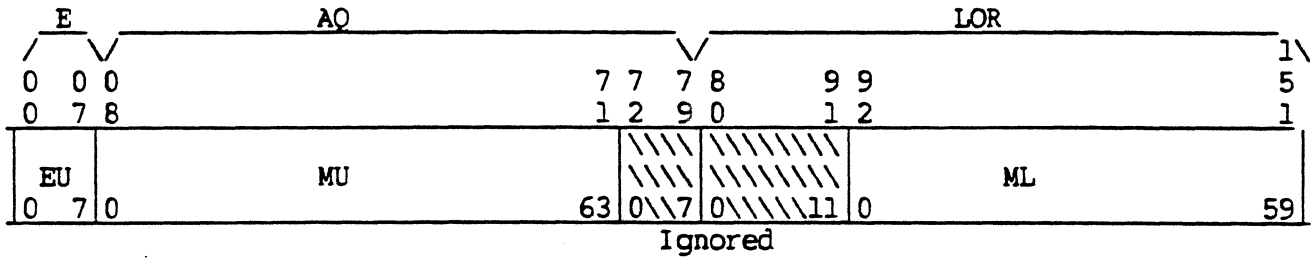


The format for data when the result is stored in main memory is as follows:

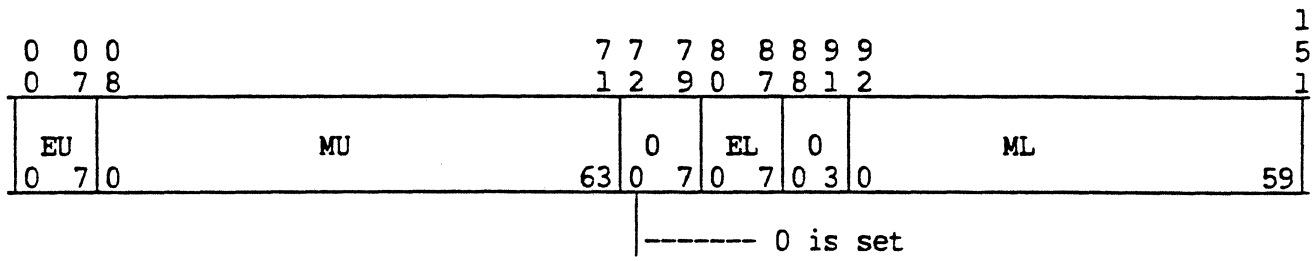


- The data in memory must reside on a double-word boundary.
- The four words of data may span two pages.

The registers E, AQ, and LOR are used for quadruple-precision arithmetic. The format for data used as operation data is as follows:



The contents of EAQ and LOR following an operation is as follows:



Field Values

EU Exponent

MU High Order Mantissa

EL EU -15 (residue)

ML Low-order mantissa

Quadruple-precision value $N = (MU + ML)16^{EU}$

The quadruple-precision instructions operate with the exponent as a hexadecimal exponent regardless of the value of bit 32 of the indicator register (IR).

Normalized Binary Floating-Point Numbers

For normalized binary floating-point numbers, the binary point is placed at the left of the most significant bit of the mantissa (to the right of the sign bit). Numbers are normalized by shifting the mantissa left (and correspondingly adjusting the exponent) until no leading zeros are present in the mantissa for positive numbers, or until no leading ones are present in the mantissa for negative numbers. The vacated bit positions on the right are zero-filled.

The number ranges resulting from the various cases of precision, normalization, and sign are given in Table 2-2.

Table 2-2. Ranges Of Binary Floating-Point Numbers

	Sign	Single Precision	Double Precision
Normalized	Positive	$-2^{-129} \leq N \leq (1-2^{-27})2^{127}$	$2^{129} \leq N \leq (1-2^{-63})2^{127}$
	Negative	$(-1+2^{-26})2^{-129} \geq N \geq -2^{127}$	$(-1+2^{-62})2^{-129} \geq N \geq -2^{127}$
Unnormalized	Positive	$2^{-155} \leq N \leq (1-2^{-27})2^{127}$	$2^{-191} \leq N \leq (1-2^{-63})2^{127}$
	Negative	$-2^{-155} \geq N \geq -2^{127}$	$-2^{-155} \geq N \geq -2^{127}$

NOTE: The floating-point number zero is not included in the table.

Binary Representation Of Fractional Values

A decimal fraction of a given number of digits cannot necessarily be represented exactly by a binary fraction of any finite number of bits. Consider, for example, the value 1/5, which is represented in decimal notation as 0.2. Trying to represent it by a four-bit binary fraction, one obtains $(.0011)_2$ or 3/16; with eight bits, one obtains $(.00110011)_2$ or 51/256. In fact, the exact value must be written as

$$(0.2)_{10} = (0.0011)_2 \ldots$$

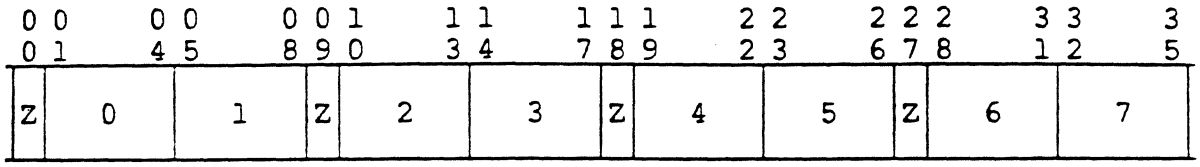
which means that the bit pattern 0011 in the binary expansion keeps repeating indefinitely. If the decimal value 0.2 is converted to a binary expansion of 71 bits and then converted back, the one-digit result would be 0.1, quite different from 0.2. The four-digit result would be 0.1999, which is almost (but not quite) equal to 0.2. If computations were involved instead of only conversions, the imprecision in the decimal result could be propagated.

Various adjustments can be made to binary fractional values to make exact decimal results highly probable. One may use binary integer notation to represent all values, whether integral or fractional, but this may make multiplication or division of an operand by a power of 10 necessary in the course of a computation.

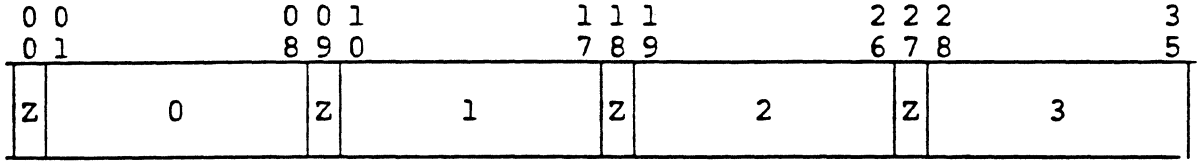
DECIMAL NUMBERS

Scaled decimal numbers that are used directly in hardware arithmetic commands are expressed as decimal digits in either the 4-bit or 9-bit character format. They are expressed as unsigned numbers or as signed numbers using a separate sign character.

Decimal data utilizes the following formats:



Packed Decimal (4-bit)



ASCII/EBCDIC (9-bit)

Z represents unused bit positions.

Decimal Data Character Codes

During arithmetic operations, decimal digits and signs are checked by the hardware as 4-bit data (the 4 least significant bits from a 9-bit numeric).

The following interpretations are made:

Bit Pattern for Character	Interpreted as	Illegal Procedure (IPR) if
0000	0	found where descriptor specifies sign
0001	1	
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
0100	8	
1001	9	
1010	+	found where descriptor specifies digits
1011	+	
1100	+	
1101	-	
1110	+	
1111	+	

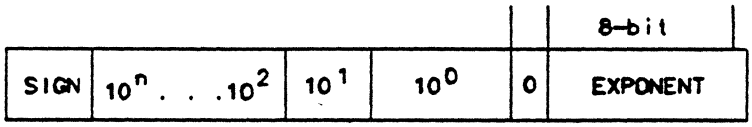
The following codes (9-bit zones are created by prefixing binary 00010) are generated for output signs; the octal values are:

	Plus	Minus
4-bit	14(13)	15
9-bit	053	055

For several numeric instructions, a sign value of 13 can be optionally generated.

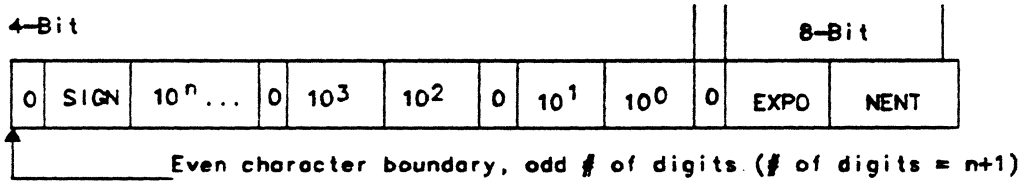
Floating-Point Decimal Numbers

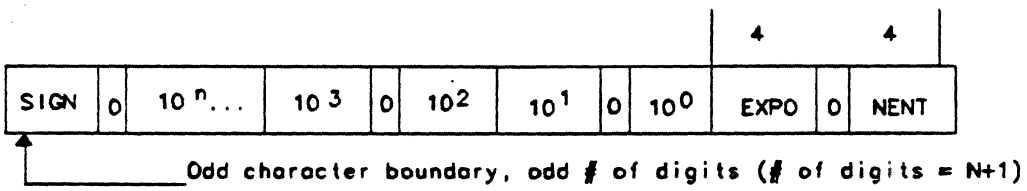
The format for a floating-point decimal number expressed in 9-bit characters is:



where: SIGN can start at any legal 9-bit character boundary

In 4-bit character notation, there are four formats for floating-point decimal numbers:





The 8-bit exponent field, which now spans two character positions, is interpreted the same as in 9-bit character mode. The other two formats are formed with n+1 even. This effectively exchanges the two exponent representations in the formats shown.

Decimal Number Ranges

The number ranges for decimal numbers are:

1. Fixed-point unsigned integer:

   Range = $0...10^{63}$

2. Fixed-point signed integer:

   Range = $\pm\ 10^{62}$

3. Floating-point (implicitly signed):

   a. 9-bit format range - $\pm\ 10^{61} * 10^{+127\ -128}$

   b. 4-bit format range - $\pm\ 10^{60} * 10^{+127\ -128}$

   c. Zero = $\pm 0 * 10^{+127\ -128}$

SECTION 3

MEMORY ORGANIZATION

The Central Processing Units (CPUs) access the main memory through the System Control Unit (SCU). Similarly, the Input/Output Multiplexer (IMX) also accesses memory through the SCU. As a component, the SCU is a passive system element, responding to requests from active units, the CPUs and the IMXs. This large, memory-oriented system architecture, permits both CPU and IMX functions to execute asynchronously and concurrently. The functions of read, store, interprocessor communication, etc., are provided by the SCU.

Increased system throughput is achieved by operating the SCU and associated memory units on a 72-bit parallel basis. This corresponds to two single-word instructions, two data words, or one double-precision fixed-point or floating-point number.

Systems with more than one system controller provide an increased effective information rate, since each system controller operates independently and its functions can be overlapped with those of other system controllers.

Additional overlap is provided by memory interlacing. Each DPS 8000 SCU operates with full memory unit interlacing, in 8-word block increments, to reduce the possibility of the same memory unit being accessed in succession.

VIRTUAL MEMORY

Virtual memory (VM) provides an extremely large, directly addressable memory space (2**43 bytes) and a complement of registers and instructions to manage virtual address space. The VM space is divided into a number of working spaces. The working spaces are further divided into variable sizes called "segments". A segment within a working space is described by a "segment descriptor", which has a base relative to the origin of the working space and a bound relative to the base, together with control information. Thus, for all memory references, virtual memory addresses are prepared relative to a particular working space and to a particular segment base within the working space. These virtual memory addresses are then mapped to real memory addresses by paging mechanisms.

To access (generate a memory address for) an area of VM, a process (used here to mean the smallest working unit of software) must have a segment descriptor that "frames" the particular segment of VM and that gives the desired permission for using this segment of VM (i.e., Read permission, Write permission, or Execute permission). A process cannot create a segment descriptor, nor change the base and bound to access an area of VM not enclosed by the area originally "framed", nor increase the permissions field. Therefore, a process is limited to accessing only those areas of VM described by segment descriptors that are available to the process.

The hardware environment for the virtual memory is composed of four elements[1]: working spaces, domains, segments, and pages. The working spaces and pages are physical elements, whereas the segments and domains are logical elements. These elements are treated as separate components of the virtual memory but must be interpreted in the context of the whole environment, since they are closely related in their interaction with each other.

Working Spaces

The virtual memory is divided into 512 (0 through 511) working spaces (WS) of 2**34 bytes, each of which is divided into fixed-length pages. These pages are used for memory management and have a fixed size of 1024 words (4096 bytes) each. Working space numbers (WSN) used to generate a particular virtual memory address are obtained from one of eight working space registers (WSR) or a segment descriptor register (DR*n*).

Page Tables

Each working space has an associated page table that identifies the real memory allocation. The page table or section table for each working space is located in real memory by a pointer that resides in the working space page table directory (WSPTD). The directory has 512 entries and the pointer to the directory is stored in the page directory base register (PDBR). Directory entries are either pointers to page tables or pointers to section tables. The section table (SCT) consists of up to 4K words called page table base words (PBW) that allow page tables to be divided and distributed throughout the memory. These pointers and tables can only be altered in the Privileged Master mode.

The virtual address has three components: a working space number (WSN), a page number, and a page byte number (commonly called an offset). The virtual address is automatically transformed to a real address by the hardware.

---

1. Historically, discussion of virtual memory included reference to working space quarters, described in this manual as working spaces. The working space quarter concept is not used by any software implementation; therefore, no further mention of working space quarters occurs in this manual. The hardware has not been changed.

Domains

Another logical element of the virtual environment is the domain. A domain is the particular subset of virtual memory that currently can be accessed by a process. It is defined initially by the collection of descriptors contained within the linkage segment (the segment described by the contents of the LSR). The domain is a flexible and temporary range of operation that may encompass several noncontiguous segments in one or more working spaces (see Figure 3-1). Two or more domains may interact by including the same segment descriptor. Each domain contains exactly one linkage segment to define the domain. A change of domain implies a change of linkage segment and vice versa. Descriptors for the domain may also be in descriptor segments described in the linkage segment, in descriptor registers, or in the parameter segment.

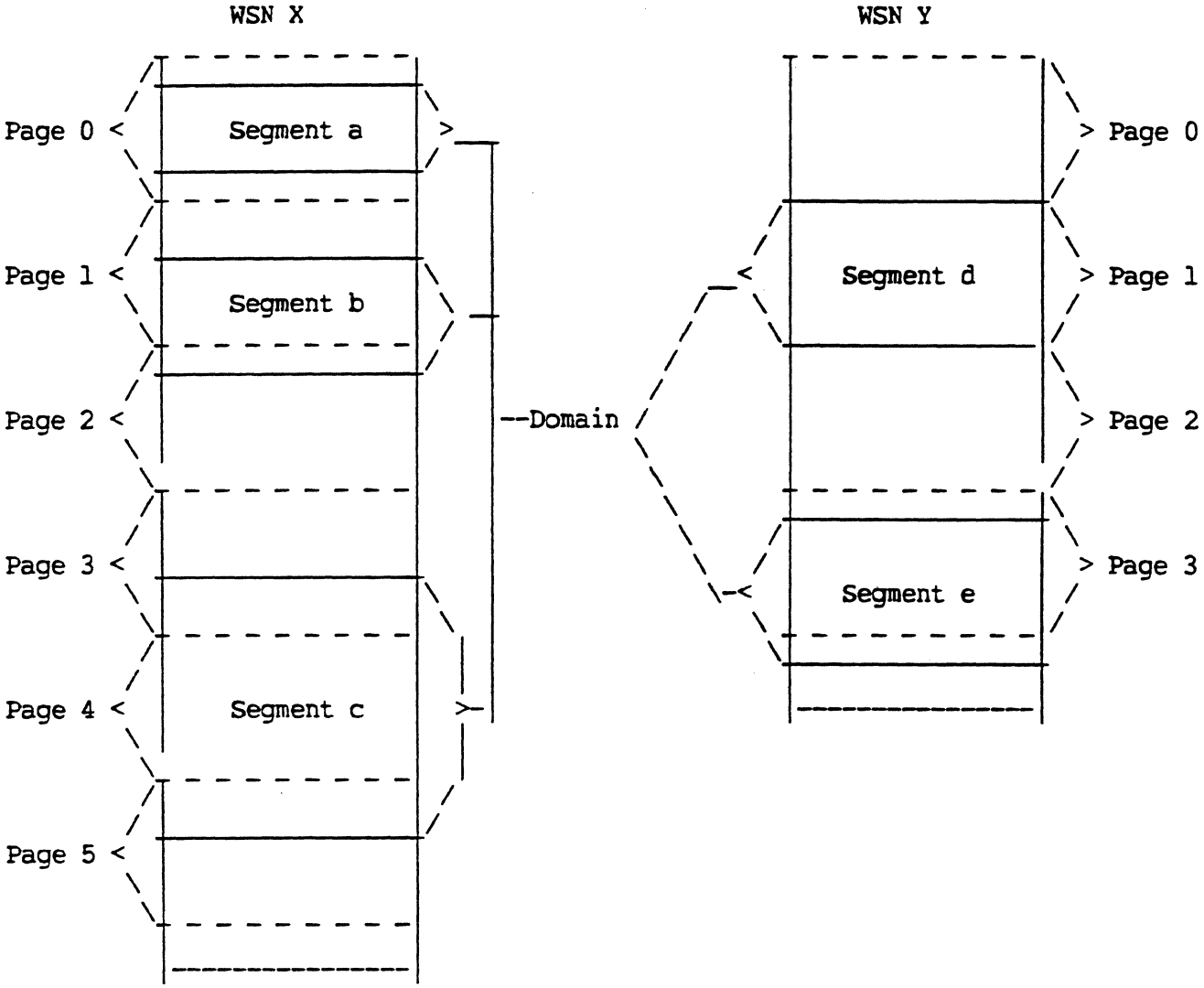


Figure 3-1. Domain Of Noncontiguous Segments

Also associated with the process are the safe store stack and the data stack segments. The safe store stack is always used (except for GCLIMB and PCLIMB) in a change of domain, but a new domain may or may not choose to access a different portion of the data stack segment. It does not have access to that portion used by the calling domain.

Normally, a change of domain is accomplished through a succession of operations that are associated with the ICLIMB instruction. Starting with two separate domains, which for convenience are referred to as calling domain and called domain, the entry descriptor accessed in the calling domain describes the called-domain linkage segment and identifies a specific initial instruction in an instruction segment described in that linkage segment. The contents of the calling domain's registers (LSR, ASR, PSR, and DSAR), as well as those of any other registers specified by the type of entry descriptor, are safe stored.

The change-of-domain CLIMB instruction indicates whether there are parameters and the number of arguments. The arguments may be either vectors or descriptors. (Refer to discussion of LDDn instruction in Section 8.) If the arguments are vectors, descriptors are prepared using the vectors and stored to form a parameter segment for the called domain.

The source of the list of vectors or descriptors is given as the contents of pointer register zero. (Descriptor register zero identifies the segment in which the list occurs and indicates whether vectors or descriptors are listed. Address register zero gives the offset in that segment of the list.) On change-of-domain return (OCLIMB), the contents of the calling-domain's domain registers and any other register contents that were safe stored are restored.

Segments

Another division of the working space is the segment. Each segment is a logical entity of variable length and may be as small as one byte or as large as 2^{32} bytes. Consequently, a segment may reside on a portion of a page or span several pages. (Refer to Figure 3-2). Segments are described with two-word (72-bit) segment descriptors. When a virtual address is generated, the segment descriptor is located in the segment descriptor register. Segments in virtual memory are specified with a base value which is relative to the origin of the WS, and a bound which is relative to the base.

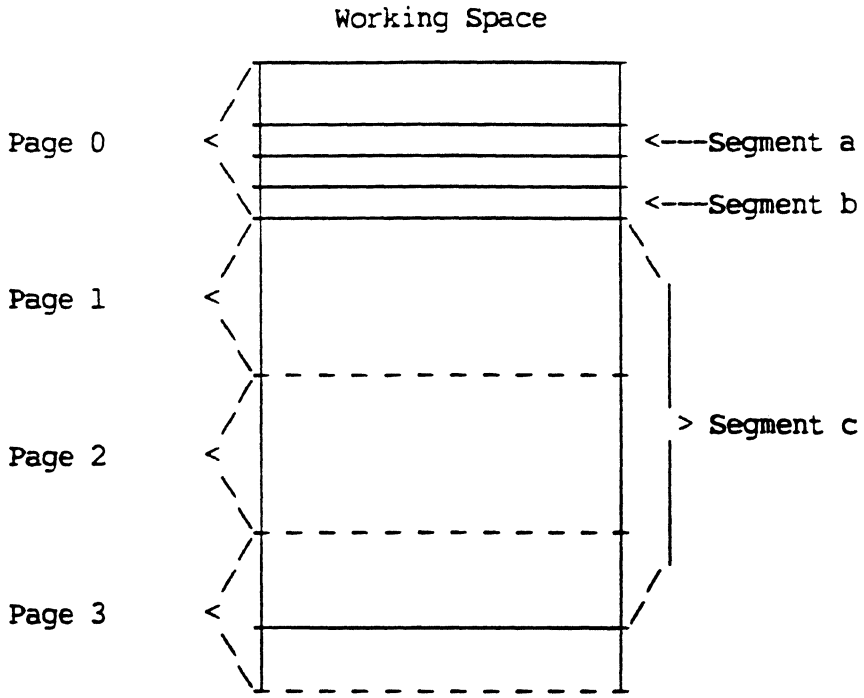
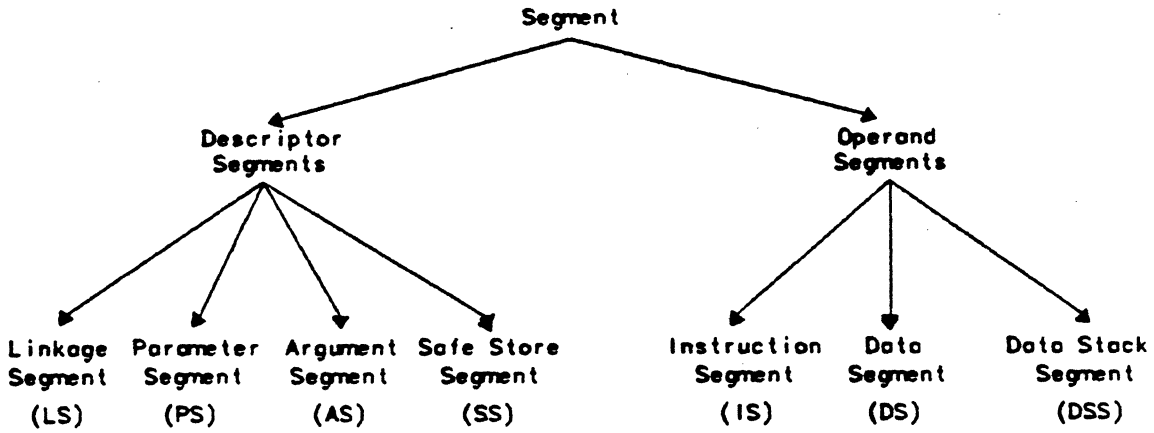


Figure 3-2. Layout Of Segments On Pages

To understand the relationship between pages and segments, it is necessary to understand the structure of a working space. The combination of a working space number and offset within the related working space is called a virtual address. Pages of 1K size are ordered sequentially by virtual page number within a working space. Each page is represented by a page table word (PTW) that points to a real page, if that page is in memory.

A segment is a logical sequence of virtual addresses, starting from a base and of a size equal to the bound of that segment. The base and bound of a segment are contained in a system protected, two-word structure called a segment descriptor. A segment may be small, contained anywhere within a page, or it may span multiple pages, irrespective of page boundaries.

A segment is characterized by its elements and the form of access to these elements, which can be Execute, Read, or Write. Segments are classified either as descriptor segments or operand segments. The descriptor segments that contain valid descriptors as part of their contents may be used as linkage, parameter, argument, or safe store segments; whereas the operand segments may be instruction-only, data-only, instruction and data segments, or data stack segments as illustrated in the following diagram.



A segment of either class may also be loaded into one of the eight operand descriptor registers (DRn).

Descriptors

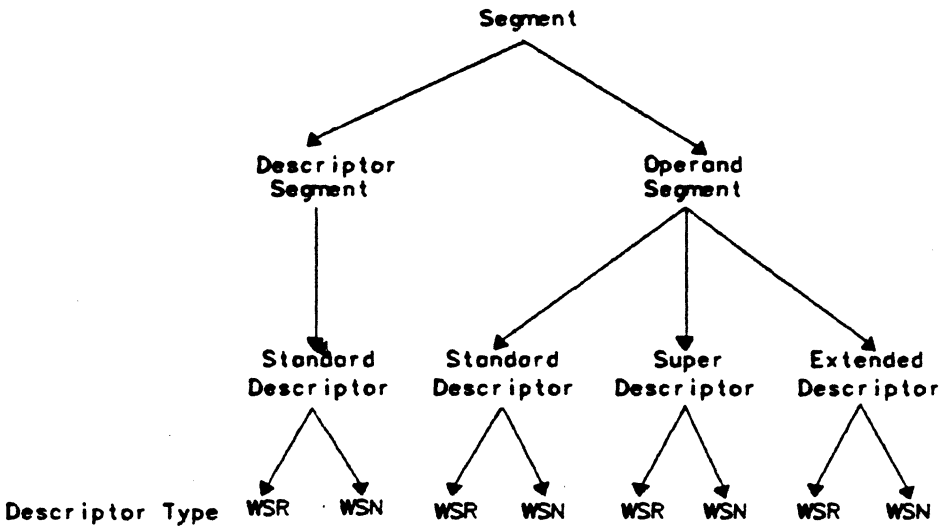
A descriptor consists of a 72-bit word-pair and locates a segment in virtual memory. When the processor hardware obtains a descriptor from memory, the processor assumes that the descriptor begins on an even-word boundary and ignores the least significant bit of the virtual word address. If a descriptor is stored from a register, the processor hardware stores on an even-word boundary.

To allow a process to have access to a segment, a copy of the descriptor must be obtained to locate the segment in virtual memory. Also, the descriptor delimits, through a set of flags, what forms of access to the segment are available.

Twelve types of descriptors are available. Those segments containing instructions, data, or a combination of both are commonly called operand segments and have descriptors that are either type 0, 2, 4, 6, 12, or 14 to indicate operand storage. The segments containing only descriptors (i.e., descriptor segments) have descriptors that are either type 1 or 3 to indicate descriptor storage. Operand memory references are always accomplished through operand segment descriptors, usually to nonhousekeeping pages, whereas descriptor references are made only through descriptor segment descriptors

to housekeeping pages. The remaining four descriptors are used only during the execution of the special transfer-of-domain (CLIMB) instruction. The list of descriptor types follows.

Type	Descriptor	Contents
0	Standard	Instructions/data
2	Standard with WSN	Data
4	Super	Data
6	Super with WSN	Data
12	Extended	Data
14	Extended with WSN	Data
1	Standard	Descriptors
3	Standard with WSN	Descriptors
5	Dynamic linking	Used only with CLIMB
8	Entry	
9	Entry	
11	Entry	



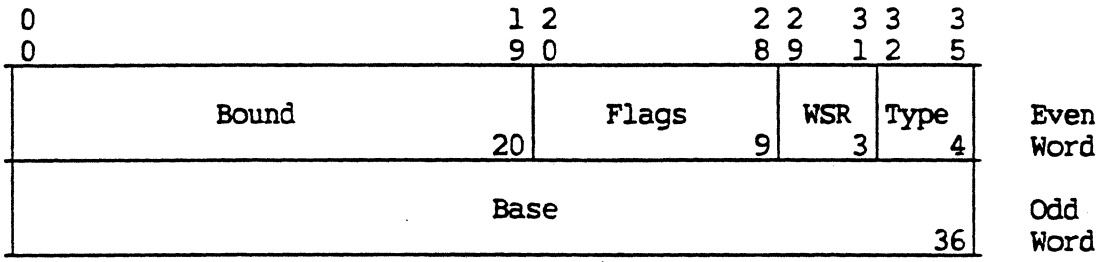
Instructions such as LDSS and LDAS that load segment descriptors from operand segments to registers and instructions such as STSS and STPS that store segment descriptors in operand memory areas access segments of type 0, 2, 4, 6, 12, or 14. In these instances, instruction operand memory addresses must specify operands in operand segments. An Illegal Procedure (IPR) fault occurs when operand or indirect word addresses are generated which specify segment descriptors of other than those types. This procedure has two exceptions:

1. Segment descriptor types 1 and 3 specify segments that include segment descriptors. The CLIMB, SDRn, LDPn, LDDn, and STDn instructions access segment descriptor segments to load or store segment descriptors. These segment descriptor segments must be located in housekeeping pages. An IPR fault occurs when either a segment descriptor is accessed with an instruction other than one of the five mentioned above, or when one of these instructions is used to access a segment descriptor in an operand segment that is not located in a housekeeping page.

2. Instructions such as LDDn can access both operand segments and segment descriptor segments because LDDn performs different operations with each access. These instructions indirectly access segment descriptors through operand segments. The safe store stack contains data other than segment descriptors. However, it is specified with type 1 or 3 segment descriptors. The safe store stack does not contain operand data and cannot be accessed except with Privileged Master Mode. Using this mode, the segment descriptor for the safe store stack can be obtained and converted to a type 0 or 2 segment descriptor. (Refer to the LDDn instruction description in Section 8.)

STANDARD DESCRIPTOR

The format of the standard descriptor is:



Bound - A 20-bit field that is the maximum valid byte address within the segment; bits 0-17 are the word address and bits 18-19 are the 9-bit byte address. The bound is relative to the base. A zero bound indicates a 1-byte segment if bit 27 is 1.

Flags - A 9-bit field that describes the access privileges as well as other control information associated with the descriptor:

Bit	Flag Code	Meaning
20	R	Read 0 Read not allowed 1 Read allowed
21	W	Write 0 Write not allowed 1 Write allowed
22	S	Store by STDn 0 Descriptor may not be stored in a type 1 or 3 segment by the STDn instruction. 1 Descriptor may be stored in a type 1 or 3 segment by the STDn instruction.
23	C	Cache Use Control Not used by DPS 8000
24	X	NS/ES Mode (when in ISR; otherwise ignored) 0 NS Mode 1 ES Mode
25	E	Execute 0 Execute not allowed 1 Execute allowed
26	P	Privilege 0 Privileged Master mode not required for execution 1 Privileged Master mode required for execution
27	B	Bound valid 0 Bound not valid; segment empty. 1 Bound field maximum valid address.
28	A	Available segment 0 Segment not available; references not allowed. 1 Segment available; references allowed.

WSR - A 3-bit field that specifies which of the eight working space registers to use with this descriptor. The working space register supplies the working space number (WSN).

Type - A 4-bit field that defines the descriptor type. The two types for standard descriptors are:

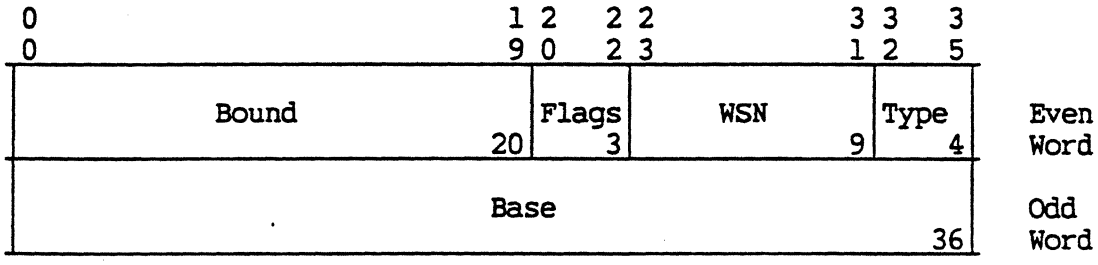
Type = 0 The descriptor "frames" instruction/operand space.

Type = 1 The descriptor "frames" an address space containing descriptors.

Base - A 36-bit virtual byte address that is relative to the working space defined in the WSR. Bits 0-33 are a 34-bit word address and bits 34-35 represent a 9-bit byte within the word.

STANDARD DESCRIPTOR WITH WORKING SPACE NUMBER

The format of the standard descriptor with working space number (WSN) is:



This format is the same as that for the standard descriptor except that the flags field has been truncated to allow the descriptor to contain the actual working space number rather than point to a working space register. The three flag bits are the same as the corresponding flag bits of the standard descriptor. The state of the truncated flags is assumed as follows:

Flags - 1. Execute not allowed (NE)

2. Not privileged (NP)

3. Bound valid (B)

4. Segment available (A)

WSN - The actual working space number.

Type - A 4-bit field that defines the descriptor type. The two types for standard descriptors witn WSN are:

Type = 2 The descriptor "frames" operand space.

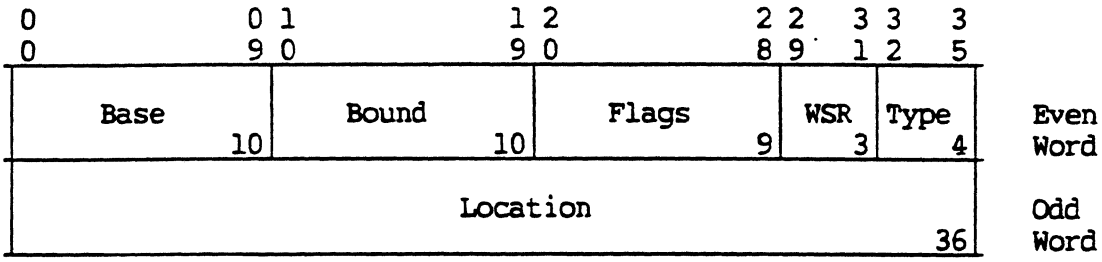
Type = 3 The descriptor "frames" an address space containing descriptors.

SUPER DESCRIPTOR

Super-descriptors may be used to define large segments. The definitions of the flags, WSR, WSN, and type fields of the super-descriptor are the same as those of the standard descriptor. The base and bound fields are automatically extended on the right to a length of 36 bits. The base is extended with zeros and the bound is extended with ones.

Therefore, a super descriptor with base, location, and bound of zero describes a segment that begins at location zero of a working space and extends 2**26 bytes (16 million words). A super descriptor with a base of 1, and location of zero, and a bound of 3 describes a segment that starts at location 2**26 and extends 2**28 bytes (64 million words).

The format of the super descriptor is:



Base - A 10-bit virtual address (unit 2**26 bytes) within a working space. The 10-bit base is converted to a 36-bit base (unit 1 byte) by extending to the right by 26 zero bits.

Bound - A 10-bit virtual address (unit 2**26 bytes) that is the maximum valid address within the segment. Conversion to a 36-bit bound (unit 1 byte) is accomplished by extending the 10-bit field to the right by 26 one bits. The bound is relative to the base.

Flags - A field that describes the access privileges associated with the descriptor (identical to the flags field for the standard descriptor).

WSR - A 3-bit field that specifies which of the eight working space registers to use with this descriptor (identical to the WSR field for the standard descriptor).

Type - A 4-bit field that defines the type for the super descriptor.

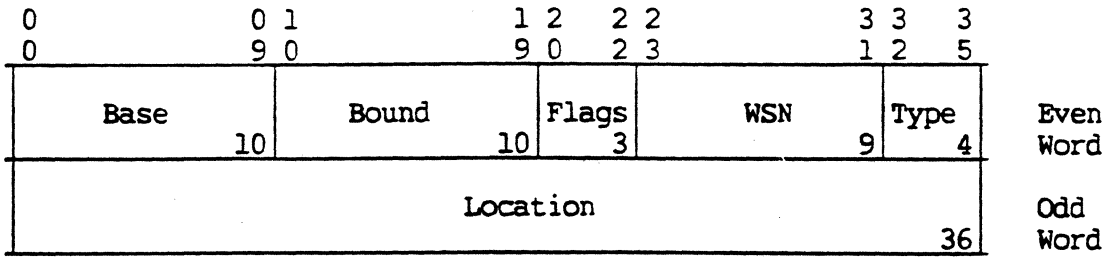
Type = 4 The descriptor "frames" operand space.

Location- A 36-bit byte virtual address relative to the base; that is, an offset from the 10-bit base. The area framed by the super descriptor extends from (Base + Location) through (Base + Bound).

NOTE: If an attempt is made to use a super descriptor in the ES mode, an IPR fault occurs.

SUPER DESCRIPTOR WITH WORKING SPACE NUMBER

The format of the super descriptor with working space number (WSN) is:



This format is the same as that for the super descriptor with the exception that the truncated flags field contains three bits that are defined identically as the corresponding three bits of the standard descriptor. The state of the truncated flags is assumed as follows:

Flags - 1. Execute not allowed (NE)

2. Not privileged (NP)

3. Bound valid (B)

4. Segment available (A)

WSN - The actual working space number

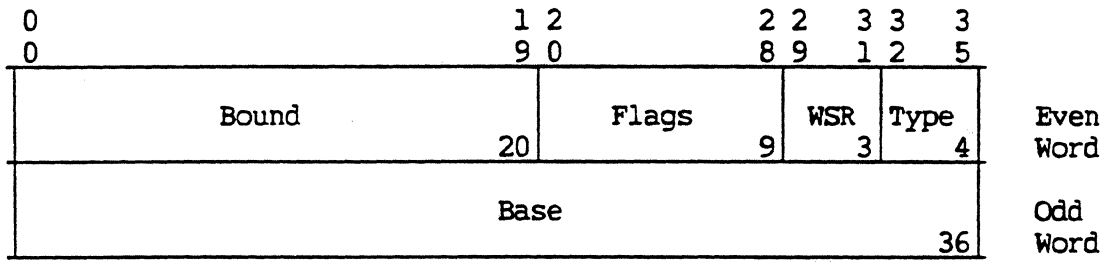
Type - A 4-bit field that defines the descriptor type as "super with WSN".

Type = 6 The descriptor "frames" operand space.

NOTE: If an attempt is made to use a super descriptor with WSN in the ES mode, an IPR fault occurs.

EXTENDED DESCRIPTOR

The format of the extended descriptor is:



Bound - A 20-bit field that is the maximum valid byte address within the segment, modulo 2^{12} bytes (2^{10} words). In other words, the bound is in terms of 4096-byte pages. It is converted to a 36-bit byte bound by extending to the right of the 20-bit field by 12 1-bits and adding four zero-bits in the high-order. The bound is relative to the base.

Flags - The same as in the standard descriptor

WSR - The same as in the standard descriptor

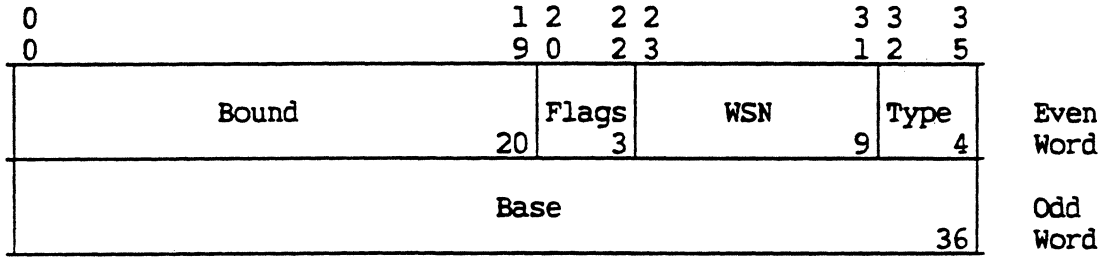
Type - The type for the descriptor

Type = 12_{10} for the extended descriptor

Base - The same as in the standard descriptor

EXTENDED DESCRIPTOR WITH WORKING SPACE NUMBER

The format of the standard descriptor with working space number (WSN) is:



This format is nearly the same as for the Extended Descriptor (T = 12_{10}), except that the flag field is shorter and a working space number (WSN) is specified.

Flags - The three bits of the flag field are the same as the corresponding standard descriptor flag bits. The state of the truncated flags is assumed as follows:

1. Execute bit allowed
2. Not privileged (NP)
3. Bound valid (B)
4. Segment available (A)

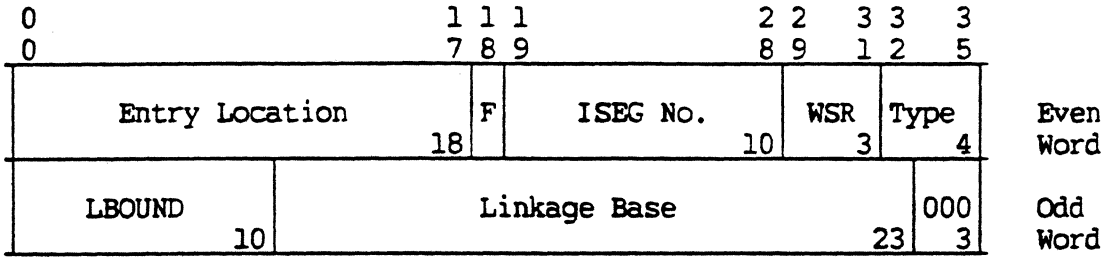
WSN - The actual working space number

Type - The type of the descriptor

T = 14_{10} indicates an Extended descriptor with WSN

ENTRY DESCRIPTOR

An entry descriptor is required to call a new domain. The entry descriptor describes the linkage segment that defines the new domain, a segment containing instructions to be initially executed in the domain, and an offset relative to the origin of that segment to which control is transferred. The entry descriptor is used with the CLIMB instruction and has the following format:



Entry Location - An 18-bit word address that is loaded into the instruction counter when the entry descriptor is used as an argument of the CLIMB instruction. The entry location is relative to the base of the new instruction segment.

F - Bit 18 is the "store" permission flag is interpreted the same as flag bit 22 of the other descriptor types.

ISEG No. - The number of the descriptor to be loaded into the instruction segment register (ISR). The ISEG number is expressed in units of descriptors and is an index relative to the new linkage segment base. The ISEG number is extended with three zeros to be expressed in bytes and is also used in loading the SEGID (IS) register as follows:

Bits 0 - 1 = 11
Bits 2 -11 = ISEG No.

WSR - The working space register containing the number of the working space to which the linkage base is relative.

Type - A 4-bit field that defines the entry descriptor type.

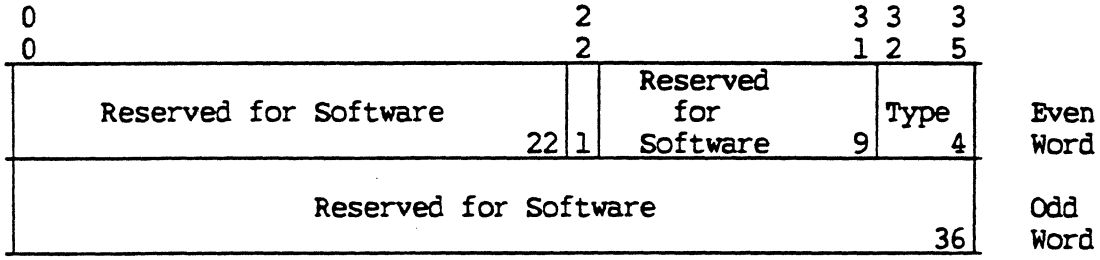
Type = 8, 9, or 11 Each number has a special meaning for the CLIMB instruction (determining the registers to be saved in the safe store stack upon change of domain).

LBOUND - The bound of the linkage segment expressed in units of descriptors. To form a standard descriptor bound, bound = 0000000||LBOUND||111.

Linkage base - The virtual starting address of the linkage segment relative to the working space defined by the working space register pointed to by the WSR field. When an entry descriptor is utilized, the associated linkage segment must be contained in the first 2**26 bytes of the working space. The last three bits of the linkage base are shown as zeros since the linkage segment must start on a double-word boundary; in actual practice, the hardware ignores the contents of these three bits.

DYNAMIC LINKING DESCRIPTOR

The dynamic linking descriptor has a double-word format with a type field of T=5 entered in bits 32-35 of the even word. Bits 0-21, 23-31, and 36-71 are used to define how the linkage is to be resolved. Bit 22 indicates store permission. A dynamic linking fault occurs when the CLIMB instruction attempts to address through a dynamic linking descriptor. Any attempt by the STDn instruction to store a dynamic linking descriptor with the store permission bit (bit 22) of word 1 equal to zero in a type T=1 or 3 segment causes an SCL2 fault. The dynamic linking descriptor has the following format:



Type - A 4-bit field that defines the dynamic linking descriptor

Type = 5

NOTE: The software usually replaces this descriptor with a Type = 11 entry descriptor while processing a dynamic linking fault.

SHRINKING

Shrinking provides descriptor access control. This is the only means available to the Slave mode for the creation of descriptors. In this process a new descriptor of decreased scope is formed in one of the descriptor registers from a descriptor already available. In essence a new subordinate segment identified by the shrunken descriptor is formed as shown in Figure 3-3.

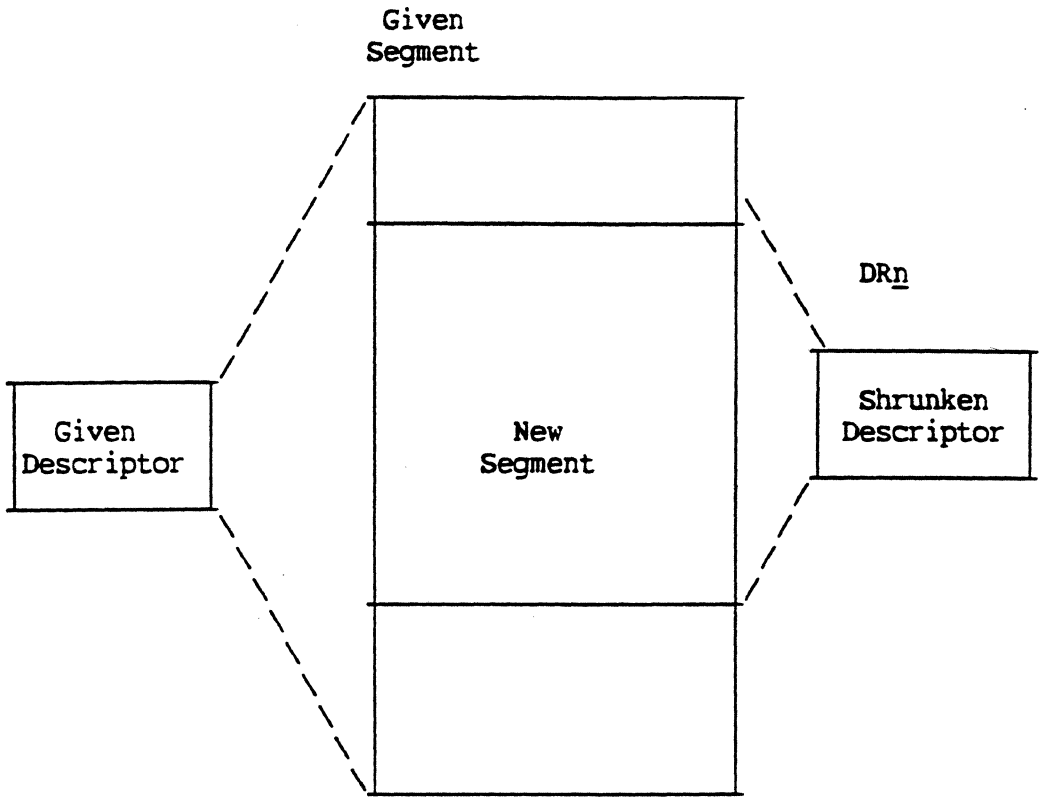


Figure 3-3. Shrunken Descriptor For Corresponding New Segment

Shrinking is used to prepare parameter descriptors for another domain, to facilitate access to portions of the domain, and to restrict access to specific shared portions of the domain. Shrinking operations may be performed on both standard and super descriptors, but the result is always a standard descriptor. A shrunken descriptor may be stored in a descriptor segment on a housekeeping page or in the descriptor stack addressable by the Argument Stack Register (ASR). Storing requires that the descriptor to be stored have store permission.

Shrinking is done using Load Descriptor Register n (LDDn) instruction, or a domain call, or the transfer version of the CLIMB instruction (ICLIMB or PCLIMB). In each instance, operands are used to define the shrinking operation in terms of a base address, size, and segment. The operands are called vectors and each is composed of two or four contiguous words. Each vector specifies one of the following functions to be performed by the instruction: copy descriptor, normal shrink, or data stack shrink. An operand of a LDDn instruction may be in the same segment as the LDDn instruction or in another segment. If the operand is in a descriptor segment, it is a descriptor, not a vector, and replacement occurs rather than shrinking.

A companion of the vector is an internal offset (a combination of a segment identifier (SEGID) and an address value) called a pointer. A pointer, in NS mode, is a 36-bit operand with sufficient information to identify an operand within a domain. Since a pointer is relative to a domain, it can be used only to address operands within its domain. Pointers for one domain cannot be used in another domain; however, pointers can be exchanged and used by several instruction segments within a domain.

A pointer in ES mode is a 2-word construct containing the same information of segment identifier (SEGID) and address offset value.

SECTION 4

PROCESSOR ACCESSIBLE REGISTERS

A processor register is a hardware assembly that holds information for use in some specified manner. An accessible register is a register whose contents are available to the user. Some accessible registers are explicitly addressed by particular instructions, some are implicitly referenced during the execution of instructions, and some are used in both ways. The accessible registers are listed in Table 4-1. Refer to the Section 8, "Machine Instruction Descriptions" for a discussion of each instruction to determine the way in which the registers are used.

Table 4-1. Processor Accessible Registers

Register Name	Mnemonic	Length (bits)	Quantity
Accumulator Register	A	36	1
Quotient Register	Q	36	1
Accumulator-Quotient Register(1)	AQ	72	1
Exponent Register	E	8	1
Exponent-Accumulator-Quotient Register(1)	EAQ	80	1
Low Operand Register	LOR	72	1
Index Registers	Xn	18	8
General Index Register	GXn	36	8
Indicator Register	IR	18	1
Timer Register	TR	27	1
Instruction Counter	IC	18	1
Address Registers	ARn	24/36	8
Linkage Segment Register	LSR	72	1
Instruction Segment Register	ISR	72	1
Segment Descriptor Registers	DRn	72	8
Segment Identity Registers	SEGIDn	12	8
Instruction Segment Identity Register	SEGID(IS)	12	1
Pointer Registers(2)	PRn	108	8
Option Register	OR	2	1
Calendar Clock(3)	CCL	52	1
Working Space Registers	WSRn	9	8
Safe Store Register	SSR	72	1
Stack Control Register	SCR	2	1
Argument Stack Register	ASR	72	1
Parameter Segment Register	PSR	72	1
High Water Mark Register	HWMR	20	1
Data Stack Descriptor Register	DSDR	72	1
Data Stack Address Register	DSAR	17	1
Page Directory Base Register	PDBR	19	1
CPU Mode Register	MR	36	1
Cache Mode Register, Lockup Fault Reg.	CMR/LFR	34/2	1
Configuration Register	CR	18	1
Address Trap Register	ATR	72	1
Virtual Address Trap Register	VATR	72	1
CPU Number Register	NR	72	1
Interrupt Mask Register(3)	IMR	36	1
CPU Fault Register	FR	72	1
Extended Fault Register	EFR	72	1
History Registers	HR	144	64
Reserve Memory Base Register	RMBR	36	1
SCU Fault Register(3)	SCUFR	72	1
Syndrome Register(3)	SYR	72	1
SCU Configuration Register(3)	SCUCR	72	1
SCU History Register(3)	SCHR	144	64
Memory Error Status Register (3)	MSR	72	1
Memory Identification Register(3)	MID	72	1

(1) These registers are not separate physical assemblies but are combinations of their constituent registers.

(2) The pointer registers are not distinct physical registers but are a collective group of registers (DRn, ARn, SEGIDn).

(3) These registers exist in the system controller. However, because they may be read and/or written with processor instructions, they have been included in this table.

In the descriptions that follow, the diagrams given for register formats do not imply that a physical assembly possessing the pictured bit pattern actually exists. The diagram is a graphic representation of the form of the register data as it appears in memory when the register contents are stored or how data bits must be assembled for loading into the register.

If the diagrams contain the character "x" or "0", the value of the bit in the position shown is irrelevant to the register. Bits pictured as "x" are not changed in the receiving cell when the register is stored. Bits pictured as "0" are set to 0 in the receiving cell when the register is stored. Neither "x" bits nor "0" bits are loaded into the register. If fields contain the "/" character, the field is not used.

NOTE: Following descriptions of all of the programmable registers, the registers used only in Privileged Master Mode are described.

ACCUMULATOR REGISTER (A)

Format: 36 bits

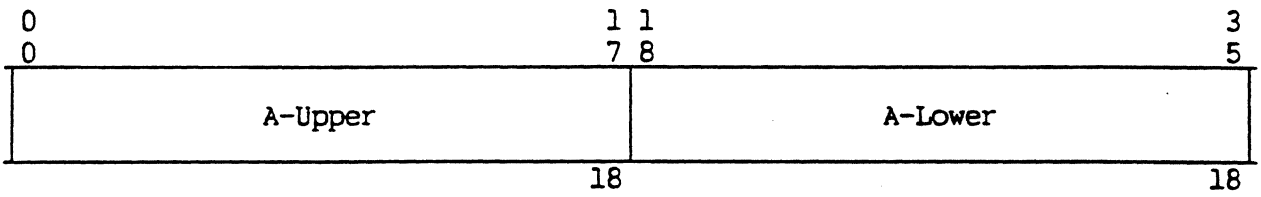


Figure 4-1. Accumulator Register (A) Format

Description:

A 36-bit physical register

Function:

In fixed-point instructions, holds operands and results.

In floating-point instructions, holds the most significant part of the mantissa and the result.

In shifting instructions, holds original data and shifted results.

In address preparation, may hold two logically independent offsets, A-upper and A-lower, or an extended range bit- or character-string length.

QUOTIENT REGISTER (Q)

Format: 36 bits

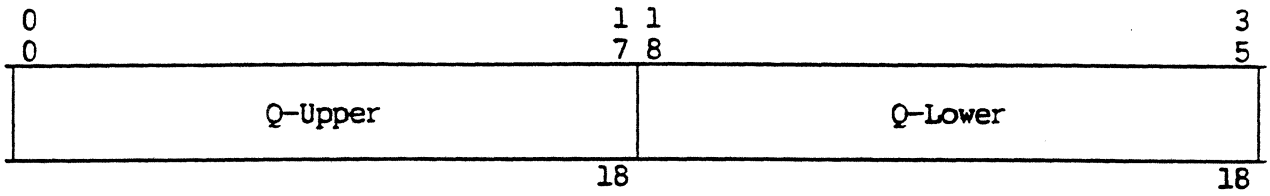


Figure 4-2. Quotient Register (Q) Format

Description:

A 36-bit physical register

Function:

In fixed-point binary instructions, holds operands and results.

In floating-point instructions, holds the least significant part of the mantissa.

In shifting instructions, holds original data and shifted results.

In address preparation, may hold two logically independent offsets, Q-upper and Q-lower, or an extended range bit- or character-string length.

ACCUMULATOR-QUOTIENT REGISTER (AQ)

Format: 72 bits

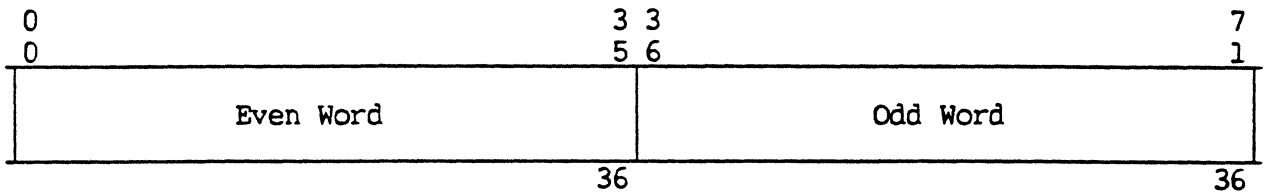


Figure 4-3. Accumulator-Quotient Register (AQ) Format

Description:

A combination of the accumulator (A) and quotient (Q) registers

Function:

In fixed-point binary instructions, holds double-precision operands and results.

In floating-point instructions, holds the mantissa and the result.

In shifting instructions, holds original data and shifted results.

EXPONENT REGISTER (E)

Format: 8 bits

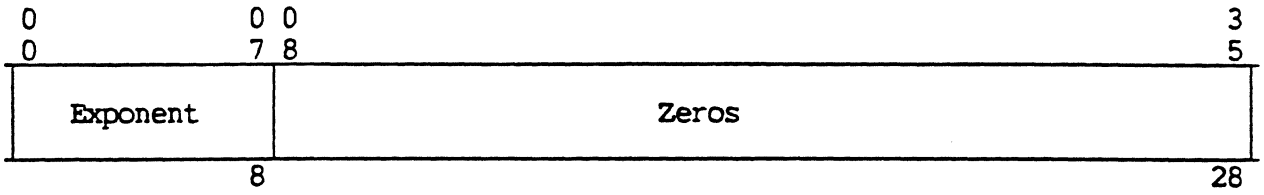


Figure 4-4. Exponent Register (E) Format

Description:

An 8-bit physical register

Function:

In floating-point instructions, holds the exponent.

EXPONENT-ACCUMULATOR-QUOTIENT REGISTER (EAQ)

Format: 80 bits

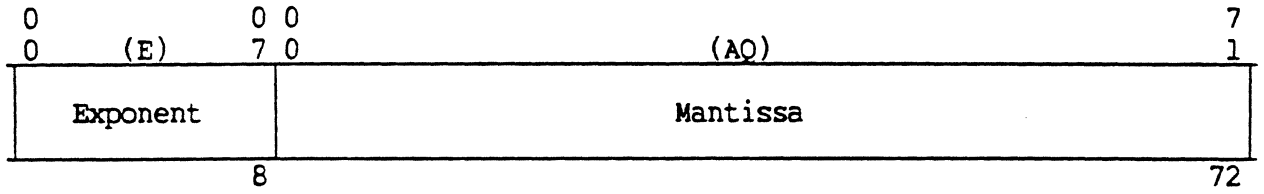


Figure 4-5. Exponent-Accumulator-Quotient Register (EAQ) Format

Description:

A combination of the exponent (E), accumulator (A), and quotient (Q) registers. Although the combined register has a total of 80 bits, only 72 are involved in transfers to and from main memory. The low-order 8 bits are discarded on store and zero-filled on load (that is, Q-register bits 28-35 are zero on load; bits 64-71 of the AQ Register are ignored). See "Floating-Point Arithmetic Instructions" in Section 7.

Function:

In floating-point instructions, holds operands and results.

LOW OPERAND REGISTER (LOR)

Format: 72 bits

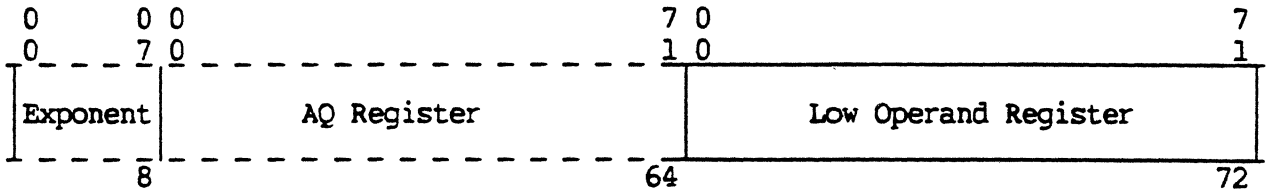


Figure 4-6. Low Operand Register Format

Description:

The lower operand register (LOR) functions in combination with the exponent (E), accumulator (A), and quotient (Q) registers in quadruple-precision floating-point operations.

Function:

The 72-bit lower operand register is used for the lower mantissa of quadruple-precision (four words) with floating-point operations.

INDEX REGISTERS (Xn)

Format: 18 bits each (NS Mode)

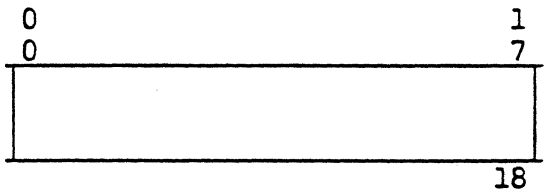


Figure 4-7. Index Register (Xn) Format

Description:

Eight 18-bit physical registers numbered 0 through 7. Index register data may occupy the position of either an upper or lower 18-bit half-word operand.

Function:

In fixed-point binary instructions, hold half-word operands and results.

In address preparation, hold bit, character, or word offsets or hold extended range bit- or character-string lengths.

GENERAL INDEX REGISTERS (GXn)

Format: 36 bits (ES Mode)

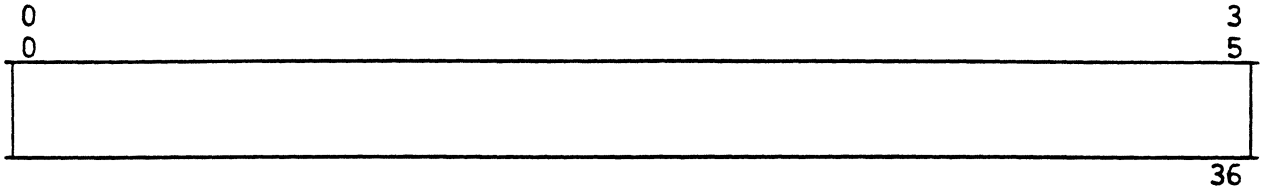


Figure 4-8. General Index Registers (GXn) Format

Description:

Eight 36-bit physical registers numbered 0 through 7 used in ES mode only. General register data may occupy the entire 36-bit operand.

Function:

May be used as a data operand register with fixed-point operations; however, in the ES mode, GXn registers may be used as the single-precision operand register.

In address preparation, hold bit, character, or word offsets or hold extended range bit- or character-string lengths.

INDICATOR REGISTER (IR)

Format: 18 bits

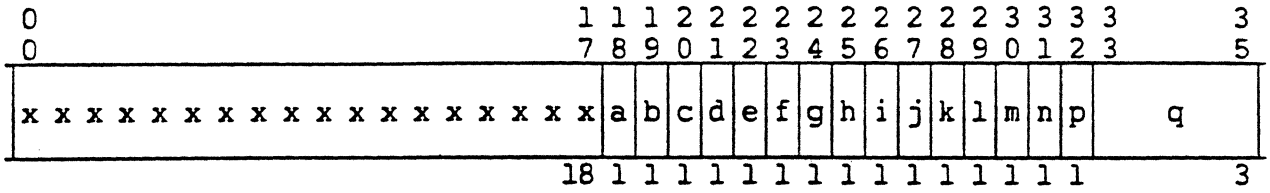


Figure 4-9. Indicator Register (IR) Format

Description:

An assemblage of 15 indicator flags from various units of the processor. The data occupies the position of a lower 18-bit half-word operand. When interpreted as data, a bit value of 1 corresponds to the ON state of the indicator; a bit value of 0 corresponds to the OFF state.

Function:

The functions of the individual indicator bits follow.

Key	Indicator name	Action
a	Zero	This indicator is set ON whenever the output of the main binary adder consists entirely of zero bits for binary or shifting instructions or the output of the decimal adder consists entirely of zero digits for decimal instructions; otherwise, it is set OFF.
b	Negative	This indicator is set ON whenever the output of bit 0 of the main binary adder has value 1 for binary or shifting instructions or the sign character of the result of a decimal instruction is the negative sign character; otherwise, it is set OFF.
c	Carry	This indicator is set ON for any of the following conditions; otherwise, it is set OFF. (1) If a bit propagates leftward out of bit 0 of the main binary adder for any binary or left-shifting instruction.

Key	Indicator name	Action
		(2) If |value1| <= |value2| for a decimal numeric comparison instruction.
		(3) If char1 <= char2 for a decimal alphanumeric comparison instruction.
d	Overflow	This indicator is set ON if the arithmetic range of a register is exceeded in a fixed-point binary instruction or if the target string of a decimal numeric instruction is too small to hold the integral part of the result. It remains ON until reset by the Transfer On Overflow (TOV) instruction or reset by some other instruction that loads the IR. The event that sets this indicator ON may also cause an overflow fault. (See overflow mask indicator below.)
e	Exponent overflow	This indicator is set ON if the exponent of the result of a floating-point binary or decimal numeric instruction is greater than +127. It remains ON until reset by the Transfer On Exponent Overflow (TEO) instruction or reset by some other instruction that loads the IR. The event that sets this indicator ON may also cause an overflow fault. (See overflow mask indicator below.)
f	Exponent underflow	This indicator is set ON if the exponent of the result of a floating-point binary or decimal numeric instruction is less than -128. It remains ON until reset by the Transfer On Exponent Underflow (TEU) instruction or reset by some other instruction that loads the IR. The event that sets this indicator ON may also cause an overflow fault. (See overflow mask indicator.)

Key	Indicator name	Action
g	Overflow mask	This indicator is set to ON or OFF only by the LDI, RET, and CLIMB instructions. When set ON, it inhibits the generation of the fault for those events that normally cause an overflow fault. When the overflow mask is ON, no overflow fault is generated if either the overflow or the exponent overflow indicator is set to ON status. When the overflow mask is set OFF, an overflow fault is generated if either the overflow or the exponent overflow indicator is set to ON status. If the overflow mask indicator is set OFF after an overflow event, an overflow fault does not occur even though the indicator for that event is still set ON. The state of the overflow mask indicator does not affect the setting, testing, or storing of any other indicator, nor does it affect the overflow fault caused by the truncation indicator.
h	Tally runout	This indicator is set OFF at initialization of any tallying operation. It is then set ON for any of the following conditions: (1) If any Repeat instruction terminates because of tally runout. (2) If a Repeat Link (RPL) instruction terminates because of a zero link address (NS mode only). (3) If a tally exhaust is detected for an Indirect then Tally modifier. The instruction is executed whether or not tally runout occurs. (4) If a string scanning instruction reaches the end of the string without finding a match condition.
i	Parity error	This indicator is set by the hardware when a parity error occurs on an access to memory. It can be set with the LDI and STI instructions. The indicator is set OFF only by instructions that load the IR.

Key	Indicator name	Action
j	Parity mask	This indicator is set ON or OFF only by the LDI, RET, and CLIMB instructions. When it is set ON, it inhibits the generation of the parity fault for all events that set the parity error indicator even when a MEMSYS fault condition is detected. If the parity mask indicator is set OFF after a parity error event, a parity fault does not occur even though the parity error indicator may still be set ON. The state of the parity mask indicator does not affect the loading, testing, or storing of any other indicator.
k	Master mode	This indicator is set ON for an interrupt acceptance, a fault acceptance, a PMME instruction execution, and the execution of an OCLIMB instruction (when the master mode bit of the indicator register to be restored is ON). This indicator is reset to OFF following the execution of a TSS, RET (with operand bit 28=0), OCLIMB (when the master mode bit of the IR to be restored is OFF), or an ICLIMB instruction (when the second word bit 19=0).
l	Truncation	This indicator is affected only by multiword instructions. It is set to ON during string instructions when the source string length is greater than the destination string length, and set to OFF when the reverse is true. For decimal arithmetic instructions, it is set to ON when there are no rounding specifications and the lowest digit, or more of the result is truncated, and set to OFF when the reverse is true. When the highest nonzero digit is lost, the Overflow Indicator is set ON.
m	Multiword instruction interrupt	This indicator is set OFF by the execution of the SPL instruction and by the end of execution of all multiword instructions, and is set ON by the events described below. The indicator has meaning only when determining the proper restart sequence for an interrupted multiword instruction. This indicator is set: When any fault or interrupt occurs during the execution of a multiword instruction (except CLIMB);

Key	Indicator name	Action
		The ON state of this indicator is used during the CLIMB (after a fault or interrupt) instruction, for example, to save the pointers and lengths data in order to resume the instruction.
n		Reserved for future use
p	Hex mode	This indicator is set ON or OFF only by the instructions that load the IR. NOTE: When set ON with bit 33 of the CPU mode register set ON, the floating-point instructions are executed in the hexadecimal exponent mode.
q		Reserved for future use

TIMER REGISTER (TR)

Format: 27 bits

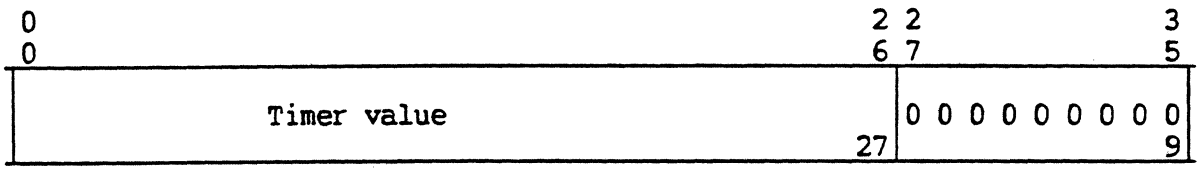


Figure 4-10. Timer Register (TR) Format

Description:

A 27-bit settable, free-running clock. The value decrements at a rate of 512 kHz. Its range is 1.953125 microseconds to approximately 4.37 minutes.

Function:

The TR may be loaded with any convenient value with the Load Timer Register (LDT) instruction. When the value next passes through zero, a timer runout fault is signalled. If the processor is in Slave mode with interrupts not inhibited or is stopped at an uninhibited Delay Until Interrupt Signal (DIS) instruction, the fault occurs immediately. If the processor is in Master or Privileged Master mode or has interrupts inhibited, the fault is delayed until the processor returns to Slave mode or stops at an uninhibited DIS instruction.

INSTRUCTION COUNTER (IC)

Format: 18 bits

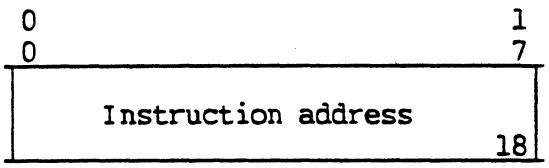


Figure 4-11. Instruction Counter (IC) Format

Description:

An 18-bit physical register

Function:

Holds the address of the current instruction being executed. The IC is incremented by 1 by the control unit for the sequential execution of single-word instructions or by the appropriate amount (2, 3, or 4) for multiword instructions. The content of the IC is changed by a transfer-of-control instruction or by a fault or interrupt.

A description of faults and interrupts is contained in Section 6.

ADDRESS REGISTERS (ARn)

Format: 24 bits each (NS Mode)

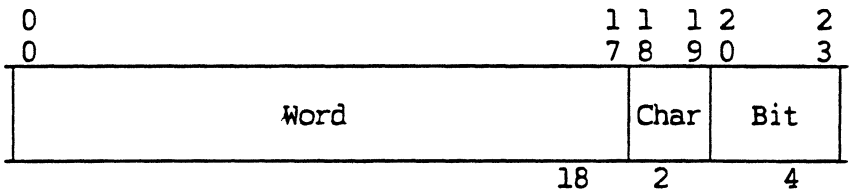


Figure 4-12. Address Register (ARn) Format (NS Mode)

Description:

Eight 24-bit physical registers numbered 0 through 7 that are associated with the segment descriptor registers (DRn) and that allow address modification on a word, character, or bit basis

Function:

The address registers provide address modification to the word, byte, and bit level:

Word - 18 bits (0-17); a word offset within the segment described by the associated segment descriptor register

Char - 2 bits; designates one of the four 9-bit characters (bytes) of which the word is composed

Bit - 4 bits; designates one of the 9 bits within the character

Format: 36 bits each (ES Mode)

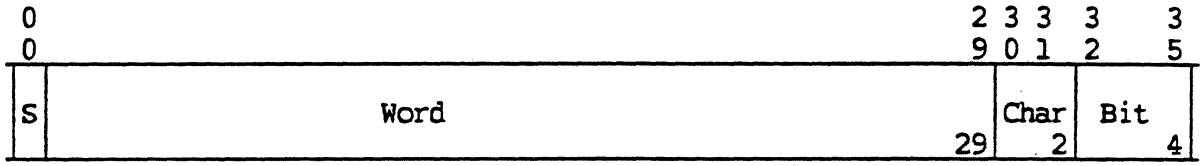


Figure 4-13. Address Register (ARn) Format (ES Mode)

Description:

Eight 36-bit physical registers numbered 0 through 7 that are associated with the segment descriptor registers (DRn) and that allow addressing on a word, character, or bit basis

Function:

In ES mode, each address register is extended to 36 bits. The ARn is as given in two's complement form, with bit 0 as sign bit. In the effective address generation, bit 0 is extended 4 bits to the left.

Word - 29 bits (1-29); a word offset within the segment described by the associated segment descriptor register

Char - 2 bits; designates one of the four 9-bit characters (bytes) of which the word is composed

Bit - 4 bits; designates one of the 9 bits within the character

LINKAGE SEGMENT REGISTER (LSR)

Format: 72 bits

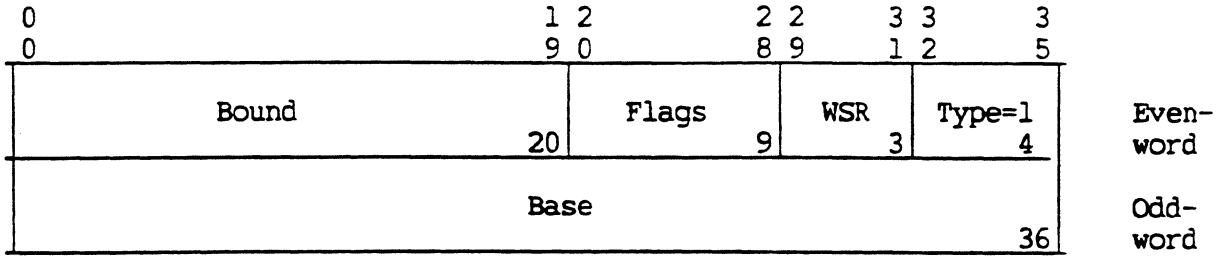


Figure 4-14. Linkage Segment Register (LSR) Format

Description:

A 72-bit register that holds a type 1 standard descriptor that describes the linkage segment of the current domain of the currently executing process

Function:

The linkage segment register is loaded only by executing a CLIMB instruction. The linkage segment register may be stored by transferring the contents of the LSR to an segment descriptor register (DRn) and then storing DRn. When the bound field of the LSR is loaded, bits 0-6 are forced to zero and bits 17-19 are forced to 111. Thus, the size of the linkage segment is effectively limited to 1024 descriptors.

INSTRUCTION SEGMENT REGISTER (ISR)

Format: 72 bits

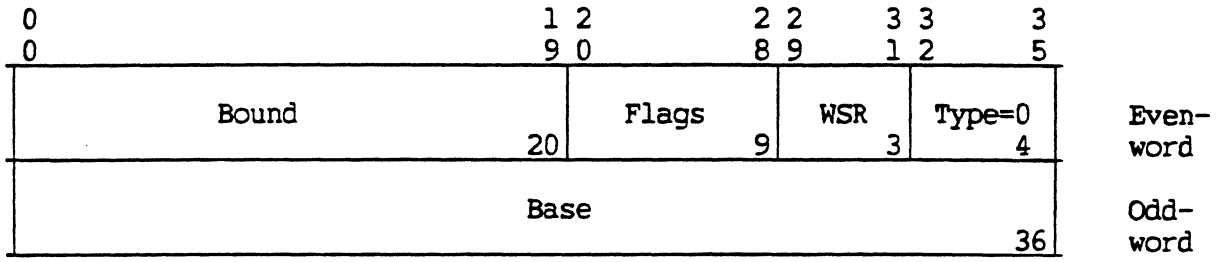


Figure 4-15. Instruction Segment Register (ISR) Format

Description:

A 72-bit register that holds a type 0 standard descriptor that describes the current instruction segment for the current domain of the currently executing process.

Function:

The instruction segment register may not be loaded or stored directly. The register is loaded during the execution of a CLIMB or transfer instruction with bit 29 ON. The ISR may be stored indirectly by moving its contents to an segment descriptor register (DRn) and then storing DRn. If bit 29 of an instruction word is zero or the AR bit in the MF field of a multiword instruction is zero, the instruction segment register is used in forming the virtual address of the operand. The base and bound values placed in the ISR are constrained; the 5 least-significant bits of the base field must be zero and the 5 least-significant bits of the bound field must be ones.

SEGMENT DESCRIPTOR REGISTERS (DRn)

Format: 72 bits each

Description:

Eight 72-bit registers that hold segment descriptors that describe address space contained within the current domain of the currently executing process. The format of the descriptors is in accordance with the content of the type fields; type fields 0, 2, 4, 6, 12, and 14 are used for operand segments and type fields 1 and 3 are used for descriptor segments.

Function:

Instructions are available for loading and storing the segment descriptor registers and for modifying their contents. A segment descriptor register is invoked for virtual operand address development when bit 29 of the instruction is 1; address bits 0, 1, and 2 specify which of the combined segment descriptor register (DRn) and address register n (ARn) is to be used. Each of these eight segment descriptor registers is associated with a corresponding address register. For example, an AR3 modification refers to the segment whose descriptor is the contents of DR3. For multiword instructions, the use of ARn and the associated DRn is specified by the AR bit in the MF field. Refer to "Multiword Modification Field" in Section 5.

SEGMENT IDENTITY REGISTERS (SEGIDn)

Format: 12 bits each

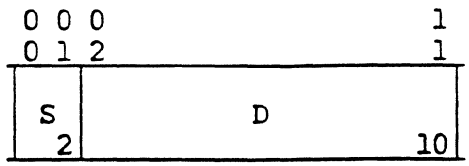


Figure 4-16. Segment Identity Register (SEGIDn) Format

Description:

Eight 12-bit registers that have a one-to-one correspondence with the segment descriptor registers (DRn). The segment identity registers point to the source of the descriptor in the DRn.

Function:

The SEGIDn registers are loaded concurrently with the related descriptor registers (DRn). The S and D field codes used in these registers indicate the origin of the descriptor (S = segment, D = descriptor offset).

When S = 0:

The D field indicates the location of the segment descriptor loaded into the DRn.

For D = 1760 through 1777 (octal), the selected register is copied into the DRn.

D = 1760	Undefined
D = 1761	The segment descriptor type field is changed. *
D = 1762	Instruction Segment Register (ISR)
D = 1763	Data Stack Descriptor Register (DSDR)
D = 1764	Safe Store Register (SSR)
D = 1765	Linkage Segment Register (LSR)
D = 1766	Argument Stack Register (ASR)
D = 1767	Parameter Segment Register (PSR)
D = 1770	DR0, Descriptor Register 0 }
D = 1771	DR1, Descriptor Register 1 }
D = 1772	DR2, Descriptor Register 2 }
D = 1773	DR3, Descriptor Register 3 } Self-Identifying
D = 1774	DR4, Descriptor Register 4 }
D = 1775	DR5, Descriptor Register 5 }
D = 1776	DR6, Descriptor Register 6 }
D = 1777	DR7, Descriptor Register 7 }

* When S = 0 with D = 1761, 1763, and 1764, a Command fault occurs unless the CPU is in the Privileged Master mode.

When S = 0 with D = 1761 in the Privileged Master Mode and the type of the segment descriptor in the DRn is T = 1 or 3, this segment descriptor type is changed to 0 or 2, respectively. SEGIDn is set to be self-identifying. No fault occurs and no operation is performed with the LDDn instruction, when the type in the DRn is not T = 1 or 3.

For D = 0000 through 1757 (octal), the descriptor in DRn was loaded from the parameter segment and D was the index to the desired descriptor.

When S = 2

The descriptor DRn was loaded from the argument stack using D as the index to the descriptor.

When S = 1 or 3

The descriptor in DRn was loaded from the linkage segment using D as the index to the descriptor.

INSTRUCTION SEGMENT IDENTITY REGISTER - SEGID(IS)

Format: 12 bits

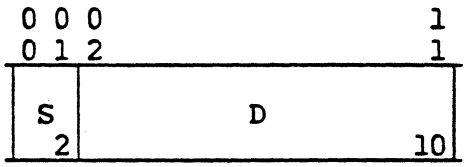


Figure 4-17. Instruction Segment Identity Register - SEGID(IS) Format

Description:

A 12-bit register that is associated with the instruction segment register (ISR) in the same manner that a SEGIDn register is associated with an segment descriptor register (DRn). This register points to the source of the descriptor in the ISR.

Function:

The instruction segment identity register may not be loaded or stored directly; it is loaded with the identity of the source of the descriptor when a transfer or CLIMB instruction loads the Instruction Segment Register (ISR). The S and D field codes used in these registers indicate the origin of the descriptor. See SEGIDn description.

POINTER REGISTERS (PR)

Format: A collective grouping of registers

Description:

Eight "convenience" logical combinations of registers

Function:

The pointer registers are not physical registers but are convenient terms used to refer to segment descriptor register (DRn), segment identity register (SEGIDn), and address register (ARn) utilized as a collective register.

OPTION REGISTER (OR)

Format: 2 bits

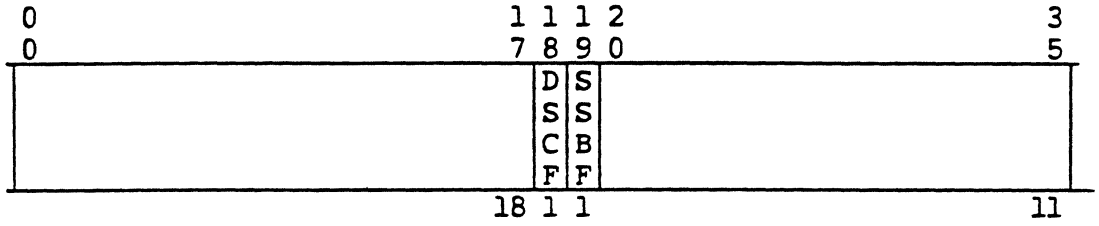


Figure 4-18. Option Register (OR) Format

Description:

A 2-bit register that controls the clearing of data stack space and bypassing the safe store portion of an inward CLIMB (ICLIMB) instruction. Bit 18 is the Data Stack Clear Flag (DSCF) and bit 19 is the Safe Store Bypass Flag (SSBF).

Function:

The option register is loaded with the Load Option Register (LDO) instruction and stored with the Store Option Register (STO) instruction.

CALENDAR CLOCK REGISTER (CCR)

Format: 52 bits

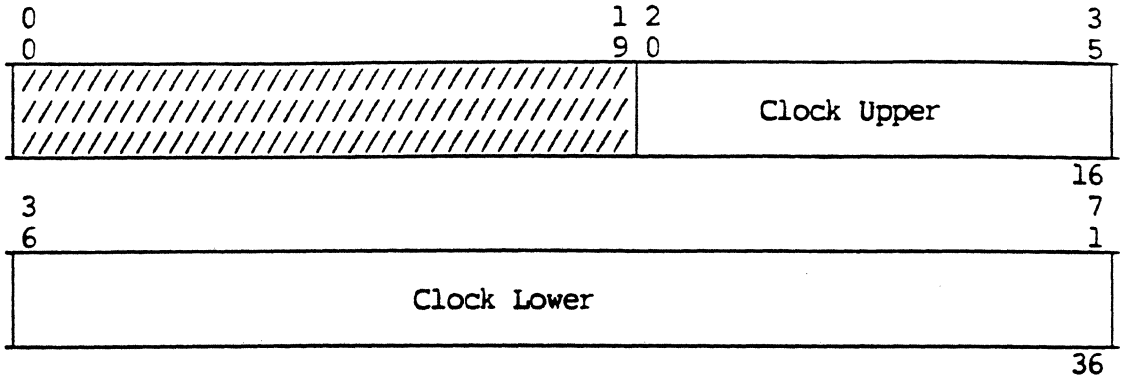


Figure 4-19. Calendar Clock Register (CCR) Format

Description:

A 52-bit register that holds a calendar clock with a resolution of one micro second

Function:

The CCR register provides a means for setting and reading the calendar clock. The CCR is set by using the SSCR 04 instruction and read by using the RSCR 04 instruction. (Refer to the individual descriptions of these instructions in Section 8).

NOTE: THE FOLLOWING REGISTERS CAN BE ACCESSED ONLY IN PRIVILEGED MASTER MODE.

WORKING SPACE REGISTERS (WSRn)

Format: 9 bits each

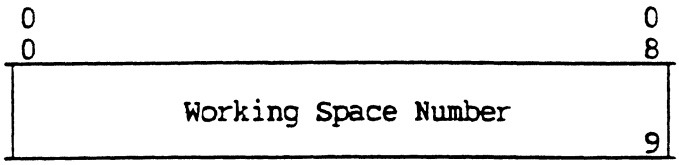


Figure 4-20. Working Space Register (WSRn) Format

Description:

Eight 9-bit registers located in the virtual unit, each of which holds a working space (WS) number that is used to form a virtual address

Function:

A working space register is referred to by the WSR field of a descriptor. The LDWS and STWS instructions are used to load and store the working space registers, respectively. To execute these two instructions, the processor must be in Privileged Master mode. When the processor is initialized and cleared, working space register 0 is set to all zeros. The working space registers provide the means for sharing and isolating working spaces.

SAFE STORE REGISTER (SSR)

Format: 72 bits

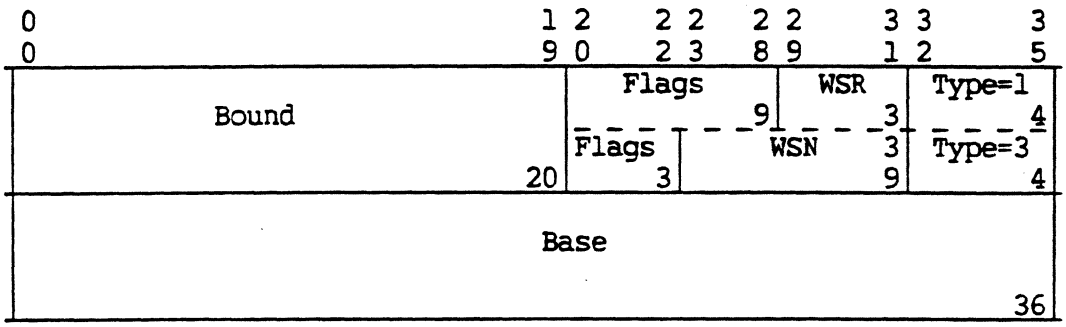


Figure 4-21. Safe Store Register (SSR) Format

Description:

A 72-bit register located in the virtual unit that holds either a Type 1 or 3 standard descriptor that describes the safe store stack of the current process. Note that the format for a Type 3 descriptor differs in that the Flags field is truncated at bit 22 to allow the descriptor to contain the actual working space number (WSN) rather than point to a Working Space Register (WSR).

Function:

The safe store register describes the safe store stack of the current process. The safe store register is loaded and stored with the Privileged Master mode instructions LDSS and STSS. A 2-bit hardware stack control register (SCR) is associated with the safe store register. The Stack Control Register (SCR) content determines the size of the safe store frame. (Refer to SCR below.)

STACK CONTROL REGISTER (SCR)

Format: 2 bits (internal)

Description:

An internal register that controls the size of the safe store frame

Function:

The SCR is initialized by execution of the Privileged Master mode instruction LDSS. This register contains the code indicating the size of the last safe store frame as shown in the table below. (Refer to the discussion of the Safe Store Register (SSR).)

SCR	Safe Store Stack Size
00	16 words
01	24 words
11	64 words
10	80 words

(Bit values are binary.)

ARGUMENT STACK REGISTER (ASR)

Format: 72 bits

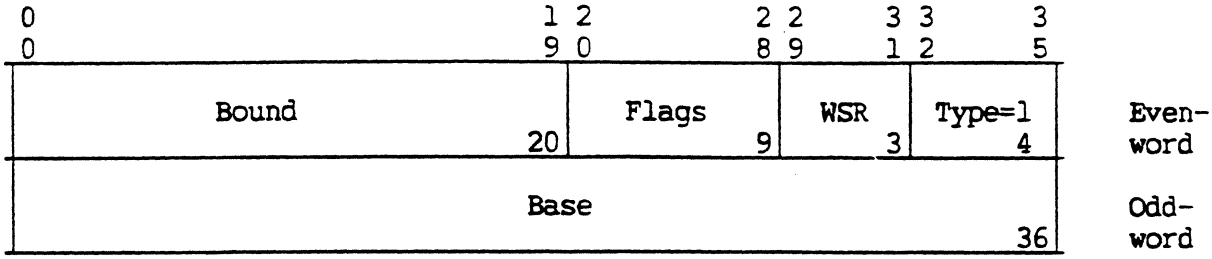


Figure 4-22. Argument Stack Register (ASR) Format

Description:

A 72-bit register that holds a type 1 standard descriptor that describes (or frames) the argument stack of the current domain of the currently executing process

Function:

Instructions are provided for loading (Privileged Master mode) and storing the argument stack register. The argument stack register is utilized by and may have its contents changed by the hardware during the execution of a Save Descriptor Register (SDRn) or CLIMB instruction. When the bound field of the ASR is loaded, bits 0-6 are forced to zero; if flag-bit 27 = 1 (bound valid), bits 17-19 are forced to 111. Thus, the size of the argument stack is effectively limited to 1024 descriptors.

PARAMETER SEGMENT REGISTER (PSR)

Format: 72 bits

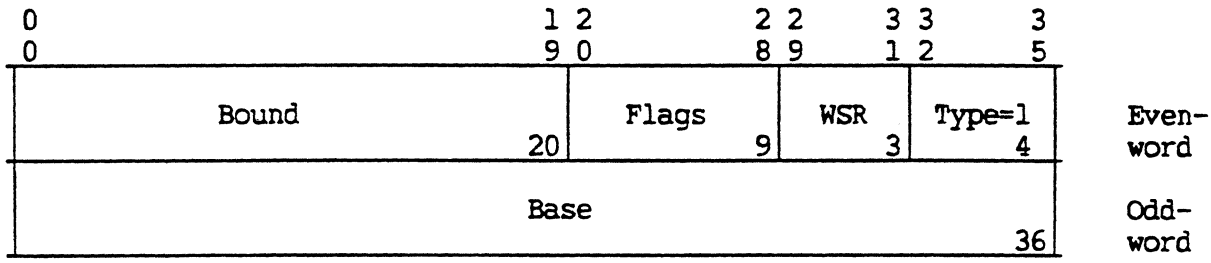


Figure 4-23. Parameter Segment Register (PSR) Format

Description:

A 72-bit register that holds a type 1 standard descriptor that frames the parameter segment of the current domain of the currently executing process

Function:

Instructions are provided for loading (Privileged Master mode) and storing the parameter segment register. The parameter stack register is utilized by and may have its contents changed by the hardware during the execution of the CLIMB instruction. When the bound field of the PSR is loaded, bits 0-6 are forced to zero; if flag-bit 27 = 1 (bound valid), bits 17-19 are forced to 111. Thus, the size of the parameter segment is effectively limited to 1024 descriptors.

HIGH WATER MARK REGISTER (HWMR)

Format: 20 bits

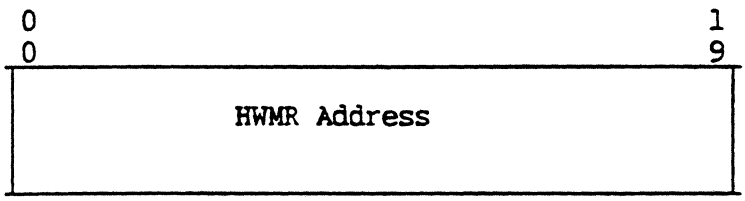


Figure 4-24. High Water Mark Register (HWMR) Format

Description:

A 20-bit register containing the maximum bound reached relative to the current ASR base.

Function:

The bound defined by the address contained in the register prevents one program from gaining access to any portion of another program's descriptors that were stored on the argument stack. The HWMR allows the PAS instruction to be executed in the slave mode. Instructions which affect the HWMR are LDAS, SDRn, and CLIMB. (Refer to the individual descriptions of these instructions in Section 8.)

DATA STACK DESCRIPTOR REGISTER (DSDR)

Format: 72 bits

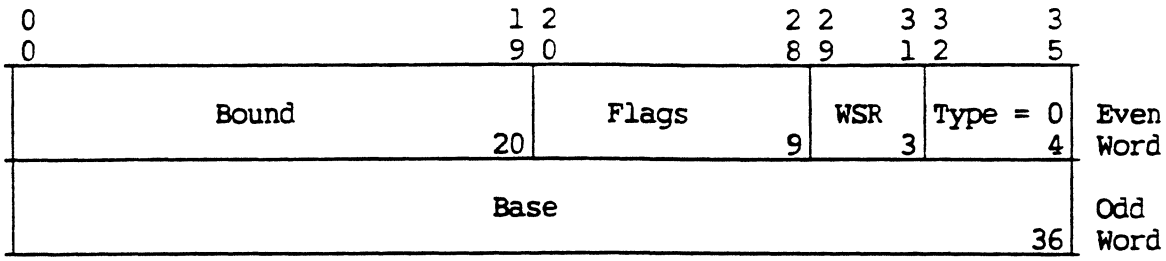


Figure 4-25. Data Stack Descriptor Register (DSDR) Format

Description

A 72-bit register located in the virtual unit that holds a type 0 standard descriptor that frames the data stack area of memory for the current process

Function:

Privileged Master mode instructions (LDDSD and STDSD) are available for loading and storing the data stack descriptor register. The contents of the data stack descriptor register are utilized by the hardware when the vector of the Load Descriptor Register (LDDn) or CLIMB instruction indicates that a working data stack descriptor is to be generated.

DATA STACK ADDRESS REGISTER (DSAR)

Format: 17 bits

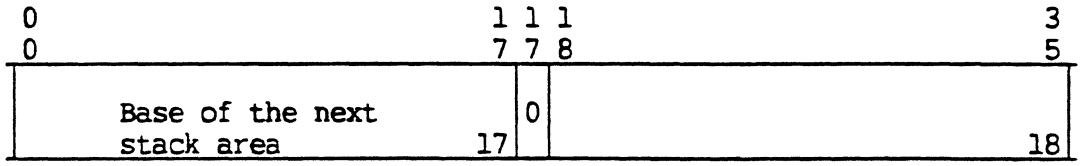


Figure 4-26. Data Stack Address Register (DSAR) Format

Description:

A 17-bit special-purpose index register that points to the next available double-word location within the data stack area of memory framed by the data stack descriptor register (DSDR). Bit 17 is always zero.

Function:

Privileged Master mode instructions (LDDSA and STDSA) are available for loading and storing the Data Stack Address Register. The contents of the DSAR may be altered during the execution of the Load Descriptor Register (LDDn) instruction, Load Data stack Address Register (LDDSA) instruction, or CLIMB instruction.

PAGE DIRECTORY BASE REGISTER (PDBR)

Format: 19 bits

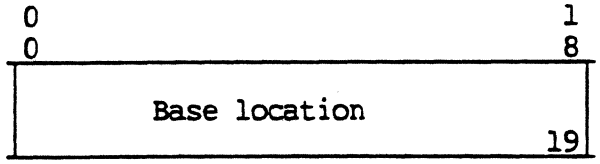


Figure 4-27. Page Directory Base Register (PDBR) Format

Description:

A 19-bit, modulo 512 word register that contains the base location of the working space page table directory

Function:

Privileged Master mode instructions (LPDBR, SPDBR) are available for loading and storing the page directory base register.

CPU MODE REGISTER (MR)

Format: 36 bits

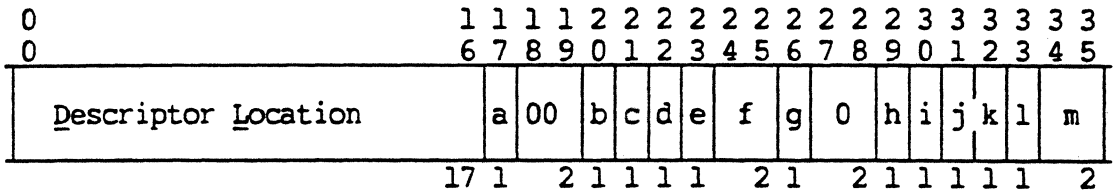


Figure 4-28. CPU Mode Register (MR) Format

Description:

An assemblage of flags and indicators from the CPU. The mode register is stored into the even word of a Y-pair by an SCPR instruction with tag = 6. The mode register is loaded by an LCPR instruction with tag = 4. These instructions may be executed in Privileged Master mode only.

On a SCPR tag 06, the second word contains the cache mode register and lockup fault register.

Function:

The CPU mode register controls the operation of those features of the processor capable of being enabled and disabled.

The functions of the constituent flags and indicators are as follows:

Key	Bits	Function
DL	0-16	Bits 10-26 of address trap match entry descriptor location; bits 0-9, 27 = 0.
a	17	When set ON, enables a trap on addess match. A fault or machine stop occurs.
	18-19	Not used
b	20*	When set ON, indicates generation of incorrect data parity. Flag is reset by return of an SCU activity status.
c	21	When set ON, indicates generation of incorrect ZAC parity. Flag is reset by return of SCU actiity status.
d	22	Control SCU 0 = Lower memory port 1 = Port High memory port
e	23	Not used
f	24-25	SEGID compare for LDPn Bit 24 - Slave mode Bit 25 - Master and Privileged Master mode 1 = enable compare 0 = disable compare
		NOTE: Disabled by GCOS
g	26**	Reset Backup fault flag
	27-28	Not used
h	29	When set ON, enables history register transfer trace mode
i	30***	When set ON, enables history register strobe

Key	Bits	Function
j	31	When set ON, resets bit 30 on fault
	32	Not used
k	33	Set ON, enables hexadecimal exponent mode
l	34	Inhibit PATROL
m	35	Set ON, enables CPU mode register

* If bit 20 is set:

1. On a store into cache, bad parity exists in the data.
2. On a store to the SCU, bad parity exists in the data.
3. On a block load into cache, bad parity exists in the data placed into cache, on the entry in cache directory, and on the data to the register defined in the instruction.

** The LCPR tag 04 instruction resets the Backup fault flag regardless of the value in C(Y); this bit is set by hardware to indicate the occurrence of a backup fault. SCPR tag 06 stores the Backup fault flag as bit 26 of the CPU mode register.

*** If bit 31 is on, then bit 30 is reset OFF (locks history registers) for the following faults:

LUF, PAR, CMD, BND, IPR,
Shutdown, SCL1, SCL2, SSSF, MPG, MSG, MWS,
Dynamic Linking

Bit 30 is set to OFF for ONC fault regardless of the bit 31 setting.

CACHE MODE REGISTER (CMR), LOCKUP FAULT REGISTER (LUF)

Format: 34/2 bits

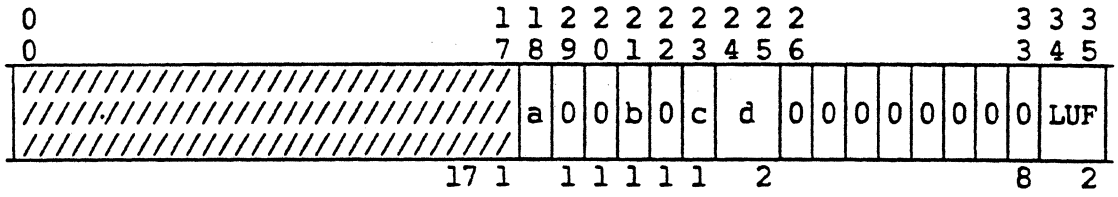


Figure 4-29. Cache Mode Register (CMR), Lockup Fault Register Format (LUF)

Description:

A 34/2-bit register holding an assemblage of bits that provide information concerning cache mode and lockup faults.

Function:

The CMR/LUF register is used to engage and disengage control of cache memory and to determine the existence of any lockup fault. This register is accessed only through Privileged Master mode. It is loaded by an LCPR instruction with tag = 02 and stored by an SCPR instruction tag = 6.

The functions of the constituent bits are as follows:

Key	Bits	Function
	0-17	Ignored
a	18*	Cache enabled, 1 = enable; reset to zero by ONC
	19	Zero
	20	Zero
b	21	Cache enabled for instruction fetch; 1 = enable
	22	Zero
c	23**	Cache to register; 1 = ON
d	24-25	Level 0,1, ON; 1 = ON
	26-33	Zero
LUF	34-35	Lockup Fault register

NOTE: Word 0 of the double-precision store contains CPU mode register information. (Refer to CPU Mode Register for definition of these bits.)

Settings of the Lockup fault register are as follows:

Bits 34-35	Milliseconds
00	8.0
01	16.0
10	32.0
11	64.0

These values are applicable in Slave mode. In Master or Privileged Master mode, the Lockup fault register is set to 128 milliseconds.

* Cache is cleared when enabled if the previous cache state was OFF. The CCAC instruction acts as a NOP.

** When the cache to register flag is ON, all double-precision instructions obtain operands from the normally selected double-word and column cache location determined by address bits Y25-26 and Y13-24, respectively. The address match in the cache directory is ignored (correct match is assumed). The cache level is selected by address bit Y12. All other instructions execute normally. If the use of the flag is to dump cache contents, the cache memory should be disabled to avoid being changed by the non-double-word instructions.

When cache is used for PATROL, only level 0 is used. The normal full/empty (F/E) bits of cache blocks used by PATROL are set to empty. PATROL operation always assumes hits in cache, independent of the state of the F/E bit and the address match. Cache flushes (e.g., due to write/notify buffer overflow) do not affect PATROL operation.

CONFIGURATION REGISTER (PORT ASSIGNMENT) (CR)

Format: 18 bits

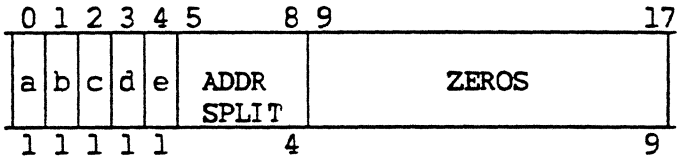


Figure 4-30. Configuration Register (Port Assignment) (CR)

Description:

An 18-bit register providing configuration information.

Function:

The CR register is used to determine the port assignment and to determine the address split. This register can be used in the Privileged Master mode only. It is stored by the SCPR instruction with tag = 11 and loaded by the LCPR instruction with tag = 11.

The functions of the constituent fields are as follows:

Key	Bits	Function
a	0	Bit zero is not loaded by software 0 = Port A accesses lower memory 1 = Port B accesses lower memory
b	1	Port A Enabled
c	2	Port B Enabled
d	3	Port A Initialize from SCU ON
e	4	Port B Initialize from SCU ON
f	5-8	Address Split 0000 = 256MW 1000 = 128MW 1100 = 64MW 1110 = 32MW 1111 = 16MW

NOTES:
1. Bits 0-4 are initialized by the Service Processor (SP)) in accordance with the designation of the lower memory port.
2. If only one port is enabled, the address split is not used. All memory accesses are directed to the lower memory port. The lower memory port must always be enabled.

ADDRESS TRAP REGISTER (ATR)

Format: 72 bits

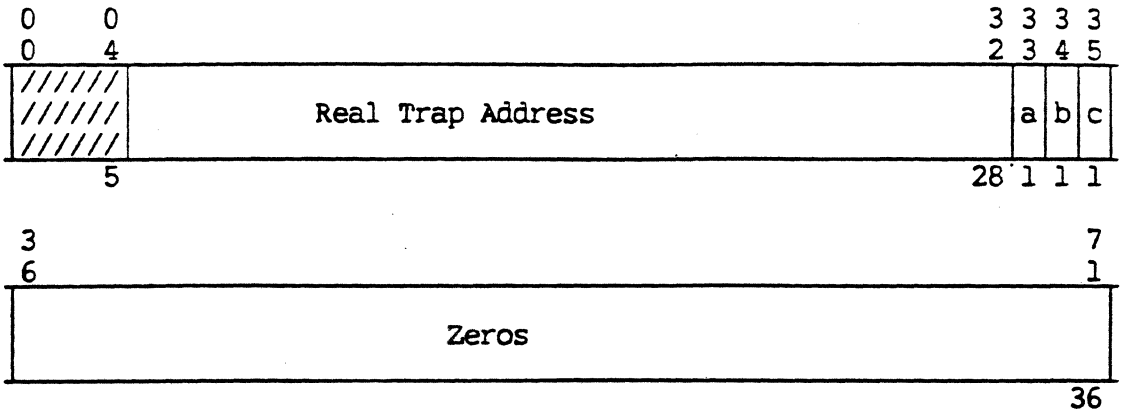


Figure 4-31. Address Trap Register (ATR) Format

Description:

A 72-bit register containing an address trap address and information relating to it.

Function:

The ATR register is used to establish the absolute word address of a trap and to indicate the conditions and status of the trap. This register can be used in Privileged Master mode only. In order for the address trap to be enabled, bit 17 in the CPU mode register must be set ON. The ATR is stored using the SCPR instruction with tag = 12 and loaded with the LCPR instruction with tag = 12.

The contents of the register fields are as follows:

Key	Bits	Function
	0-4	Ignored
	5-32	Real word address

Key	Bits	Function
a	33	0 = trap on instruction fetch or operand fetch 1 = trap on instruction fetch
b	34	0 = trap on load or store 1 = trap on operand store or indirect store
c	35	If ON, trap enabled on a real address
	36-71	Zeros

VIRTUAL ADDRESS TRAP REGISTER (VATR)

Format: 72 bits

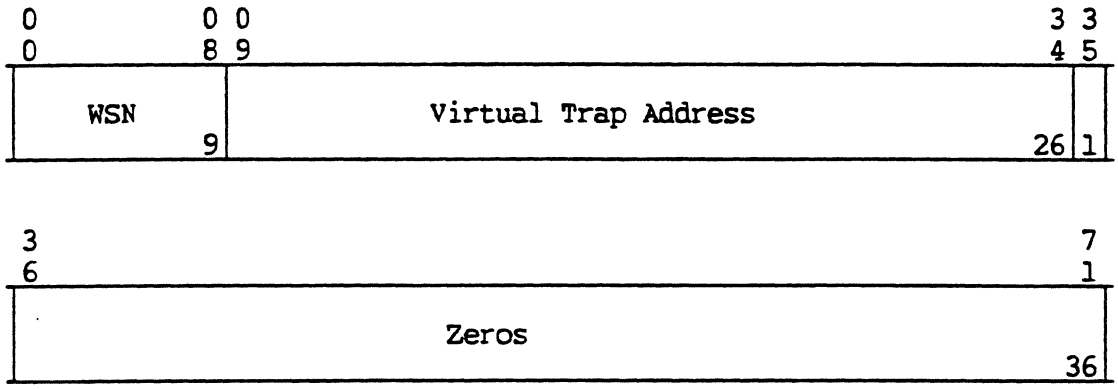


Figure 4-32. Virtual Address Trap Register (VATR) Format

Description:

A 72-bit register containing a virtual address trap address and information relating to it.

Function:

This 72-bit register is used to establish the working space number and virtual address of a virtual address trap. This register can be used in Privileged Master mode only. In order for the address trap to be enabled, bit 17 in the CPU mode register must be set ON. It is stored with the SCPR instruction with tag = 14 and loaded with the LCPR instruction with tag = 14.

The functions of the constituent fields are as follows:

Bits	Function
0-8	Working Space Number
9-34	Bits 15-40 of the virtual address
33,34	Bits 33 and 34 of ATR apply to VATR operation. Therefore, the trap conditions are common for ATR and VATR operation.
35	When set ON, enables a trap on a virtual address

CPU NUMBER REGISTER (NR)

Format: 72 bits

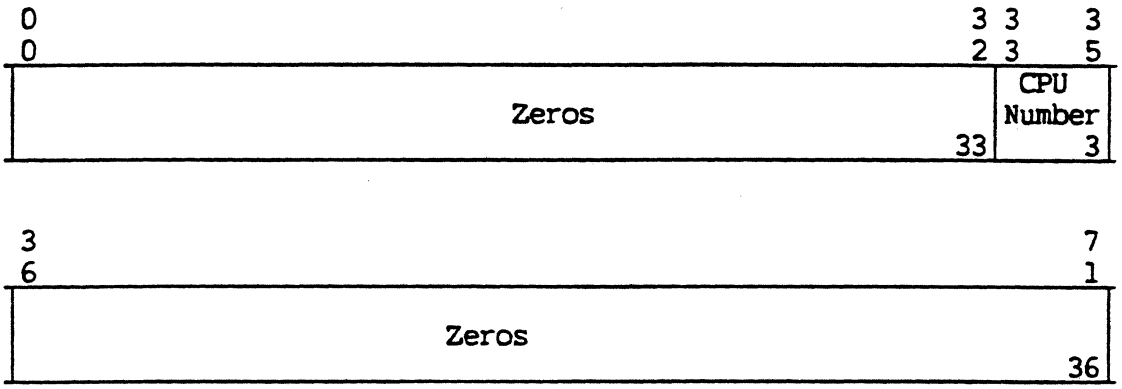


Figure 4-33. CPU Number Register (NR) Format

Description:

A 72-bit register that holds the CPU number

Function:

The NR register is used to establish the CPU number. The NR register can only be used in Privileged Master mode. It is stored by the SCPR instruction with tag = 13 and loaded by the LCPR instruction with tag = 13.

Only three bits of the two-word register are used as shown below:

Bits	Function
0-32	Zeros
33-35	CPU Number
36-71	Zeros

INTERRUPT MASK REGISTER (IMR)

Format: 36 bits

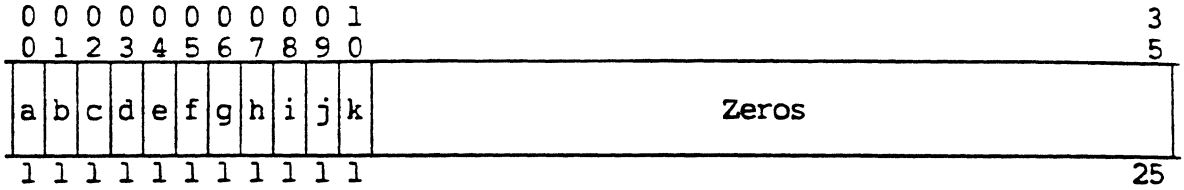


Figure 4-34. Interrupt Mask Register (IMR) Format

Description:

A 36-bit register that contains a mask for interrupts.

Function:

The IMR is used to enable or disable the interrupt levels from the CPU. The CPU can set the IMR with the Load Interrupt Mask Register (LIMR) instruction and can read the IMR with the Read Interrupt Mask Register (RIMR) instruction. Both of these instructions execute in Privileged Master Mode only. (Refer to discriptions of LIMR and RIMR in Section 8.)

An IMR per port exists in the SCU to inform the CPU of a particular event. (Refer to Interrupt Procedures in Section 6.)

The contents of the constituent bits of the IMR are as follows:

Key	Bits	Function
	0-7	Interrupt levels (functions listed are a software convention)
a		0 not used by GCOS
b		1 when ON = fault channel interrupt
c		2 not used by GCOS
d		3 when ON = terminate interrupt
e		4 not used by GCOS
f		5 when ON = marker interrupt
g		6 not used by GCOS
h		7 when ON = special interrupt

Key	Bits	Function
i	8	All Mask, conditionally (see "k" below) When ON enables interrupt present signals (XIP) to all ports
j	9	Port connect mask. When ON enables connect faults
k	10	Functions as indicated below:

Bit 10 contents	Bit 8 contents	All Mask contents
x	1	1
0	0	Unchanged
1	0	0

CPU FAULT REGISTER (FR)

Format: 72 bits

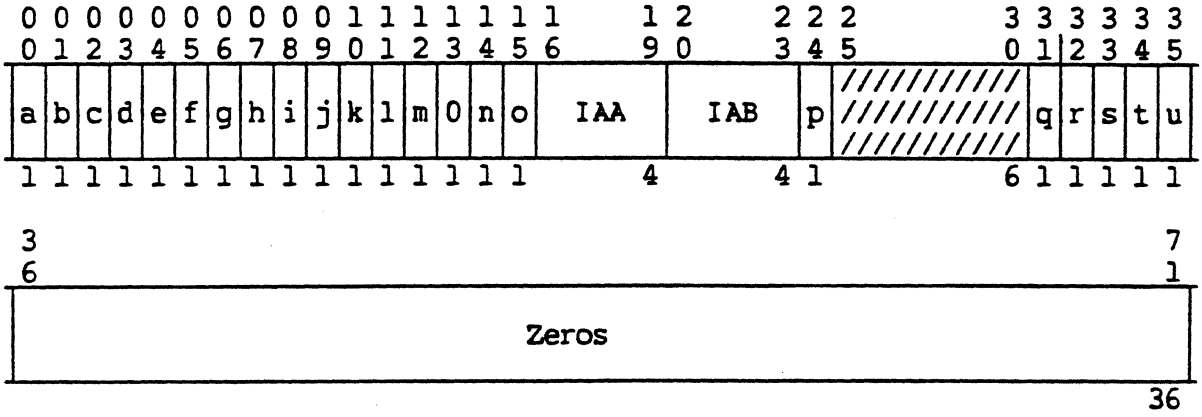


Figure 4-35. Fault Register (FR) Format

Description:

A combination of flags and registers located in the system control unit (SCU). The fault register contains the conditions in the processor for several of the hardware faults.

Function:

The FR register is stored and cleared by an SCPR instruction with the TAG = 1. The data is stored into the word pair at location Y and that bits 36-71 (Y+1) are cleared. The fault register cannot be loaded. Data accumulates in the fault register during a fault until the register is stored and cleared. The data is not overwritten during subsequent fault events.

An explanation of the constituent bits and their functions follows:

Key	Bits	Function
a	0	When ON, a firmware-detected opcode, repeat, or modify Illegal Procedure fault (IPR) in MVE
b	1	When ON, an IPR in MVE
c	2	When ON, an illegal EIS descriptor: REG code for AR displacement, DU/DL, Repeat, Modify, Register length code, IPR
d	3	When ON, an A-cycle or V-cycle, IPR
e	4	When ON, an illegal descriptor, IPR
f	5	When ON, indicates parity error in CA or CB chips
g	6	When ON, illegal EIS data, IPR
h	7	When ON, parity error on even word from the SCU port
i	8	When ON, parity error on odd word from the SCU port
j	9	When ON, cache directory multiple match
k	10	When ON, that the processor has attempted a retry to the SCU; an error on the retry causes a CPU Command fault
l	11	When ON, indicates parity error in CN or CP chips
m	12	When ON, execution unit (EU) scratch pad parity error
	13	Not used
n	14	When ON, indicates parity error in EA chips
o	15	When ON, indicates parity error in CQ chips
IAA	16-19	Illegal action code from SCU on Port A. (See Table 4-2.)
IAB	20-23	Illegal action code from SCU on Port B. (See Table 4-2.)
p	24	When ON, a write-notify receiving buffer overflow (causes cache to automatically be cleared).
	25-30	Not used
q	31	When ON, parity error on write-notify at receiving port (causes cache to automatically be cleared).

Key	Bits	Function
r	32	When ON, a cache directory parity error
s	33	When ON, a cache storage parity error
t	34	When ON, illegal action on store
u	35	When ON, that parity error occurred on other than the target pair of words. (Cache is always loaded 8 words at a time, but only two of these words represent the target pair.)

NOTES: 1. Bits 01-04 added for additional fault resolution

2. Bits 05, 11, 12, 14, 15 added to locate parity error checker

System Controller Illegal Action Codes:

The errors reported by the System Control Unit (SCU) cause illegal action codes resulting in CPU faults. The activities causing these faults, the faults, and the results are displayed in Tables 4-2 and 4-3.

Table 4-2. System Controller Illegal Action Codes

Code (Binary)	Activity	CPU Fault Type	Result
0xxx	Good memory activity	None	
1xxx	Memory error detected	Parity	
x000	Good SCU activity	None	
x001	Uncorrected read/alter/ rewrite (RAR) error	Parity	Uncorrected data rewritten to memory
x010	Bound check error	Bound	
x011	Parity error on write	Parity	Write aborted; if multiple writes, all aborted
0100	CONNECT to disabled or halted port	Command	
x101	Uncorrected read error	Parity	Incorrect data transmitted
x110	Internal SCU address/zone error	Parity	
x111	SCU multi-error detection	Parity	

Table 4-3. Source Of Fault Register Errors

		Source Of Error		
		CPU H/W	SCU H/W	GCOS S/W
Fault Register				
0				x
1				x
2				x
3				x
4				x
5		x		
6				x
7		x	x	
8		x	x	
9		x		
10		x	x	
11		x		
12		x		
13		x		
14		x		
15		x		
16-19		x	x	
20-23		x	x	
24		x		
25-30	(unused)	-	-	-
31		x	x	
32		x		
33		x		
34		x	x	
35		x	x	
Extended Fault Register				
0	(unused)	-	-	-
1	(unused)	-	-	-
2		x		
3		x		
4	(unused)	-	-	-
5		x		
6		x		

EXTENDED FAULT REGISTER (EFR)

Format: 72 bits

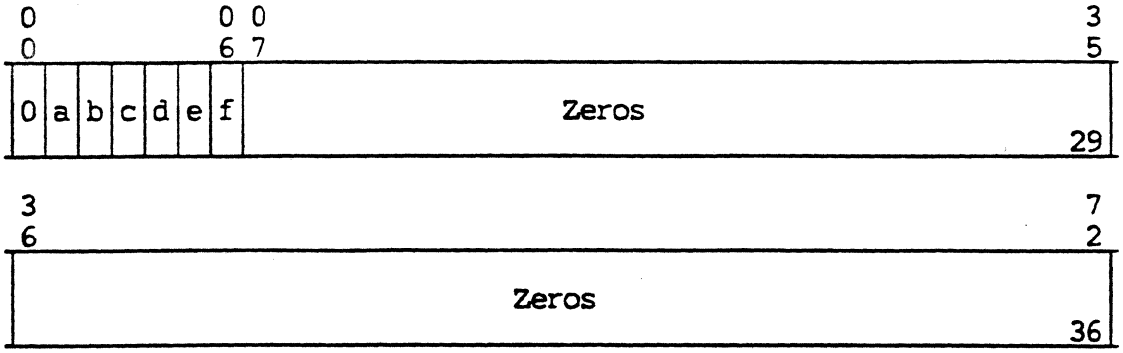


Figure 4-36. Extended Fault Register (EFR) Format

Description:

The 72-bit EFR register containing PATROL information obtained from the DI status register (RDS).

Function:

The EFR is used to determine diagnostic and error conditions not contained in the FR. The EFR can only be used in the Privileged Master mode. It is stored by the SCPR instruction with tag = 3. The EFR register cannot be loaded.

The functions of the constituent bits are as follows:

Key	Bits	Function Indicated
	0	Always zero
a	1	When ON, PATROL cycle completed.
b	2	When ON, PATROL detected error.
c	3	When ON, a CPU firmware single error corrected
d	4	When ON, connect from diagnostic unit
e	5	When ON, a parity error in page table word associative memory (PTWAM) directory
f	6	When ON, a parity error in page table word storage
	07-71	Always zero

HISTORY REGISTER (HR)

Format: 144 bits

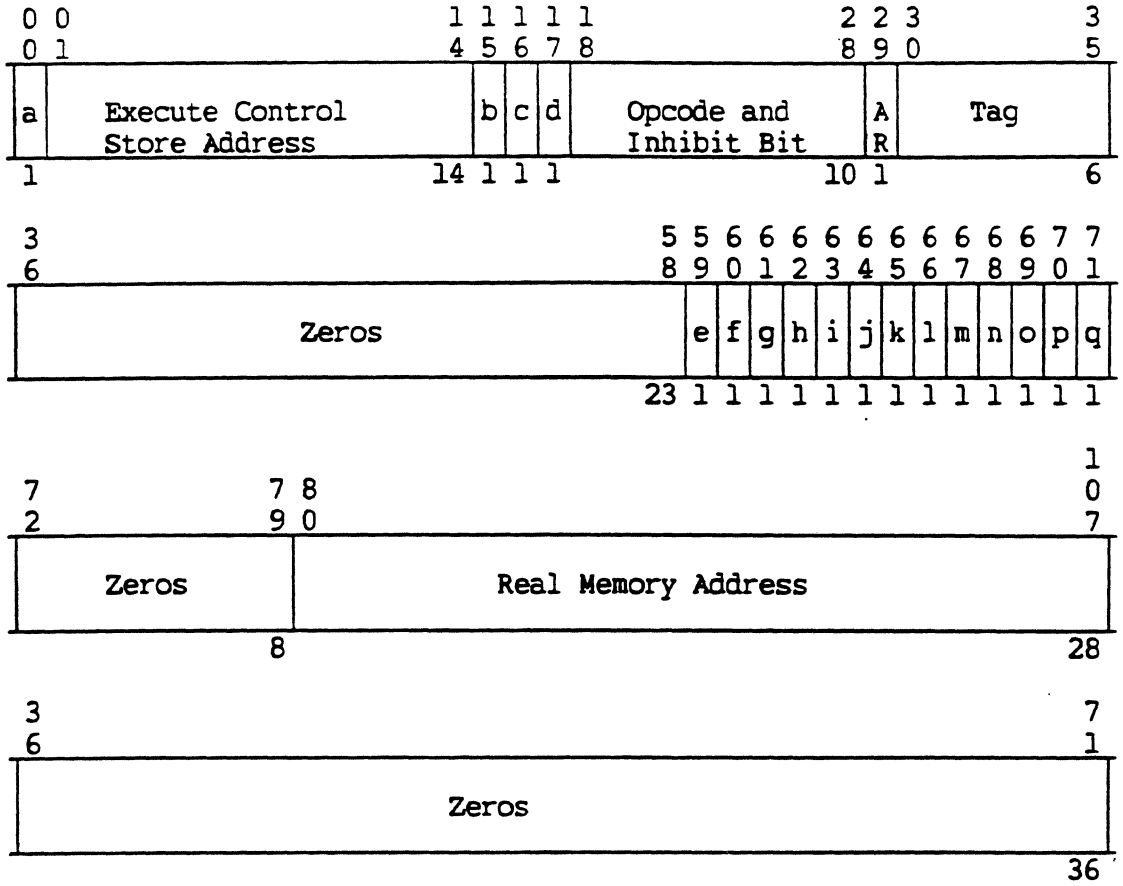


Figure 4-37. History Register (HR) Format

Description:

The history registers record information about the 64 micro steps preceding the current step. Each history register entry is four words long; the depth of the history registers is 64 entries. The history registers are implemented as two independent groups. Each group has its own address pointer. Word 0 is in the first group; words 1 and 2 are in the second group. The first group of history registers receives an entry on every regular clock (micro instruction cycle). The second group receives an entry on every C cycle. If the history register mode is set to transfer trace (by Test Mode Register bit 13), the second group is entered only on transfer-go cycles.

Function:

A history register is loaded by the LCPR instruction with tag = 03 or 07 and is stored by the SCPR instruction with tag = 20. (Refer to LCPR and SCPR instruction descriptions in Section 8.) Entries are made according to controls set in the mode register.

The meaning of the constituent flags and registers are as follows:

Key	Bits	Flag Name	Function
Word 0:			
a	00	DIDL	Execution cycle in the idle cycle
	01-14	ECS	Execution control store address (address of next micro instruction)
b	15	CEND	Last micro instruction of the instruction
c	16	DPOA	Current "A" cycle for the operand
d	17	FPIA	Current "A" cycle for the instruction
	18-28	RBIR	Opcode and inhibit bit of the instruction
	29	AR	Address register bit
	30-35	RSIR	Tag field of the instruction
Word 1			
	36-58	Zeros	
e	59	FSTRC	Store cycle
f	60	FDBLC	Double-word memory access
g	61	FDIRC	Direct operand
h	62	INSFCH	Instruction fetch
i	63	FIC17C	Bit 17 of the instruction counter (IC)
j	64	DPOAC	Operand first read or write cycle
k	65	DPGF	Paging cycle
l	66	PTW	PTW rewrite cycle
m	67	DPPG	Prepage cycle
n	68	Retry disable bit. (Instruction being executed is not directly retryable if set.)	

Key	Bits	Flag Name	Function
o	69	PTBUSY	Port busy
p	70	FBLKLD	Block load request to cache
q	71	FBYRD	Cache bypass read
Word 2			
	72-79		Zeros
	80-107		Real memory address
Word 3			
	108-144		Zeros

RESERVE MEMORY BASE REGISTER (RMBR)

Format: 36 bits

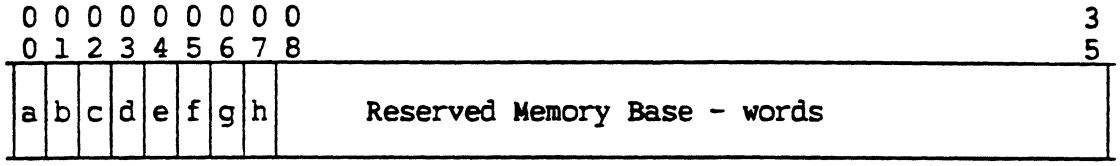


Figure 4-38. Reserve Memory Base Register (RMBR) Format

Description:

A 36-bit register designating the active processors and the reserve memory base. The bit setting, of the individual bits in bits 0-7, indicates an active processor when set = 1.

Function:

The RMBR is loaded by the Privileged Master mode instruction Load Reserve Memory Base (LRMB) and stored by SCPR tag 10.

The meaning of the constituent bits are as follows when set = 1.

Key	Bits	Function
a	0	When ON - processor #0 active
b	1	When ON - processor #1 active
c	2	When ON - processor #2 active

Key	Bits	Function
d	3	When ON - processor #3 active
e	4	When ON - processor #4 active
f	5	When ON - processor #5 active
g	6	When ON - processor #6 active
h	7	When ON - processor #7 active
	8-35	Reserved memory base -- Real memory address pointing to a real memory reserved exclusively for the CPU firmware

SCU FAULT REGISTER (SCUFR)

Format: 72 bits

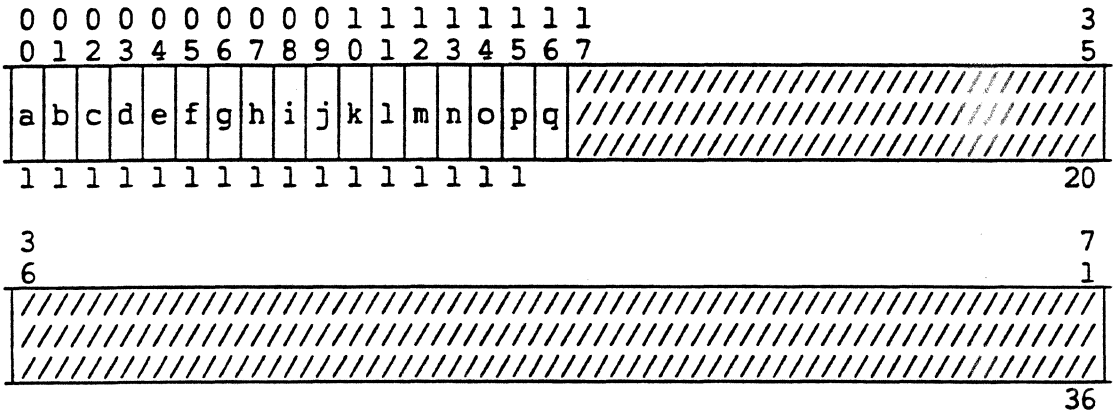


Figure 4-39. System Control Unit Fault Register (SCUFR) Format

Description:

The first 18 bits of the 72-bit SCU fault register contain an accumulation of flags indicating errors occurring in the SCU.

Function:

The SCU fault register is read and reset by the Read System Controller Register (RSCR) instruction. The SCU selection is based upon the control SCU mode bit (22) in the CPU mode register.

The contents of the constituent bits are as follows:

Key	Bit	Error Indicated
a	0	Write data parity error
b	1	Read data parity error on C board

The contents of the constituent bits are as follows:

Key	Bit	Error Indicated
a	0	Write data parity error
b	1	Read data parity error on C board
c	2	Bound check error
d	3	Non-correctable EDAC error
e	4	Port hold request
f	5	Backpanel address/zone bus parity error
g	6	Port zone address/zone bus parity error
h	7	Memory error
i	8	Memory lock timeout
j	9	Connect queue overflow
k	10	Interrupt queue overflow
l	11	Connect to a disabled port
m	12	Connect to a halted port
n	13	Correctable EDAC error
o	14	Read/clear parity error
p	15	SCU/port bus parity error
q	16	Interrupt/connect queue data parity error (This shows up as a parity error in the IA field of the CPU fault register. The data read in is not reliable.)
	17-71	Unused

SYNDROME REGISTER (SYR)

Format: 72 bits

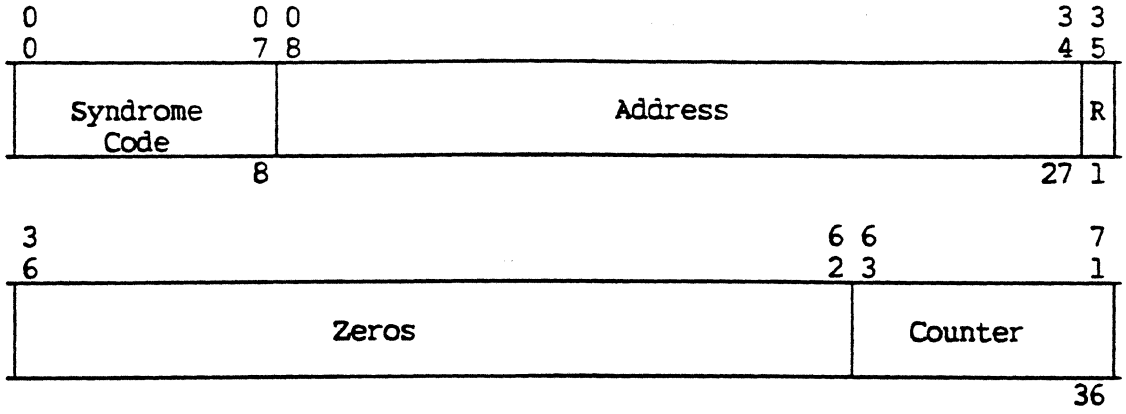


Figure 4-40. Syndrome Register (SYR) Format

Description:

An 8-bit syndrome code with a corresponding real memory address, a read alter rewrite (RAR) bit, and a counter that counts the number of EDAC errors.

Function:

The first word of the syndrome register is locked when a non-correctable EDAC error occurs. The counter in the second word operates continuously. In the unlocked state, an entry is made in the first word when a memory read operation produces a non-zero EDAC syndrome and the counter is incremented. The counter is incremented for each additional error and wraps around when it reaches the maximum count that it can hold. The syndrome register is read by the RSCR instruction with bits 22-24 = 6. SCU selection is based on the control SCU bit in the CPU mode register. When the syndrome register is read, it is unlocked and the counter is reset to zero.

The contents of the constituent bits are as follows:

Bit	Function
0-7	A code that specifies either the position of the bit in error, or whether it is a single bit error, or if not single, the number of bits in error.
8-34	Bits 0-26 of the real memory address (double word) of detected syndrome

Bit	Function
35	Memory operation type 0 = read 1 = RAR
36-62	Zero
63-71	Counter

SCU CONFIGURATION REGISTER (SCUCR)

Format: 72 bits

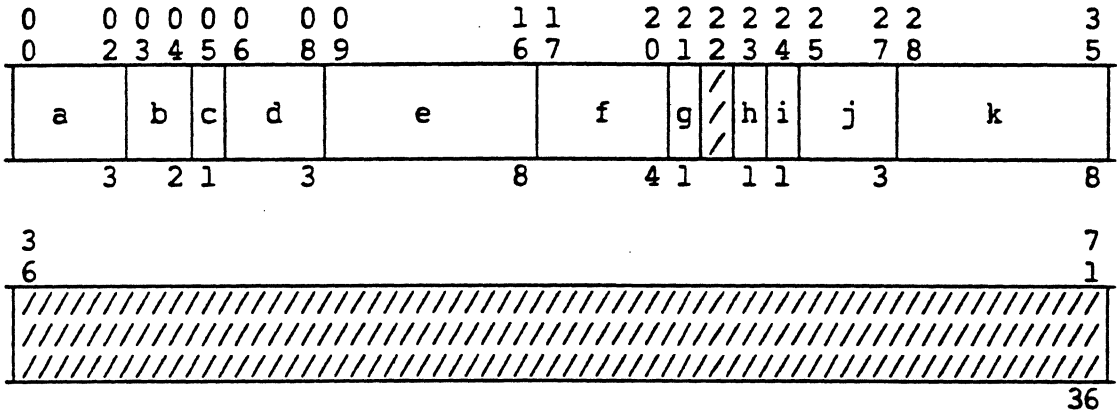


Figure 4-41. SCU Configuration Register (SCUCR) Format

Description:

A 72-bit SCU register that controls configuration and operation

Function:

The SCUCR is read and set in the Privileged Master mode by instructions RSCR and SSCR. (Refer to individual descriptions of these instructions in Section 8.)

The functions of the constituent bits are as follows.

Key	Bits	Function
a	0-2	The number of memory units attached to an SCU Interlace configuration 000 = 1 001 = 2 010 = 4 011 = 8 100 = 16 101 = 1 110 = 1 Non-interlace configuration 111 = 16
b	3-4	History Register Control: Recording Mode 00 = OFF, inhibit entry 01 = ON, record all selected activities continuously 10 = ON, record all selected activities, stop on fault and reset bits to 00 11 = ON, record start of selected activities, stop on fault and reset bits to 00
c	5	History Register Control: Port Select 0 = Record only for designated port 1 = Record for all ports
d	6-8	History register Control: Designated Port
e	9-16	Upper bound modulo 1 megawords (corresponds to minimum memory size)
f	17-20	Lower bound modulo 16 megawords (corresponds to port address split)
g	21	Used for hardware test
	22	Not used
h	23	Used for hardware test
i	24	ID definer 1 = logical ID select 0 - Physical ID select

Key	Bits	Function
j	25-27	Requesting port number (read only)
k	28-35	Port enable indicator for ports 0-7 1 = enable 0 = disable
	36-71	Unused

SCU HISTORY REGISTER (SCUHR)

Format: 144 bits

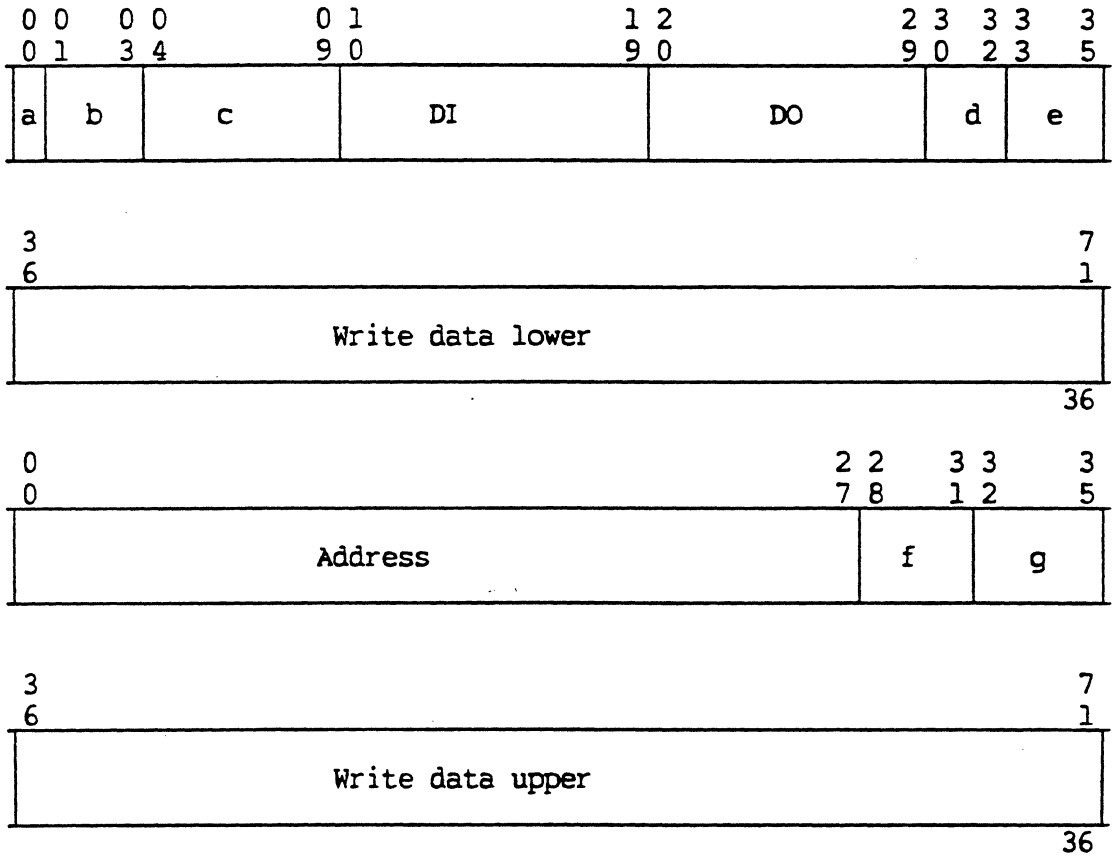


Figure 4-42. SCU History Register (SCUHR) Format

Description:

The four-word SCU history register records activity status, activity flags, and command flags. A circular storage is maintained for 1024 activity cycles. If no activity occurs during a clock period, no entry is written by the SCU.

Function:

This register is read using the Privileged Master mode instruction Read System Control Register (RSCR) . A single two-word pointer is maintained. This pointer is incremented twice on each four-word SCU entry and once on each two-word read. If the history register is locked, it is necessary to reset the configuration register to the correct recording mode in order to turn the history register on.

The contents of the constituent fields of the register are as follows:

Key	Bits	Function
Word Pair 0		
a	0	Start of activity
b	1-3	Port
c	4-9	Command
DI	10-19	Data-in activity shift register summation
DO	20-29	Data-out activity shift register summation
d	30-32	Port priority
e	33-35	Activity number
	36-71	Write data, lower (previous cycle)
Word Pair 1		
	0-27	Real memory address
f	28-31	Zone
g	32-35	Memory select
	36-71	Write data, upper

MEMORY ERROR STATUS REGISTER (MSR)

Format: 72 bits

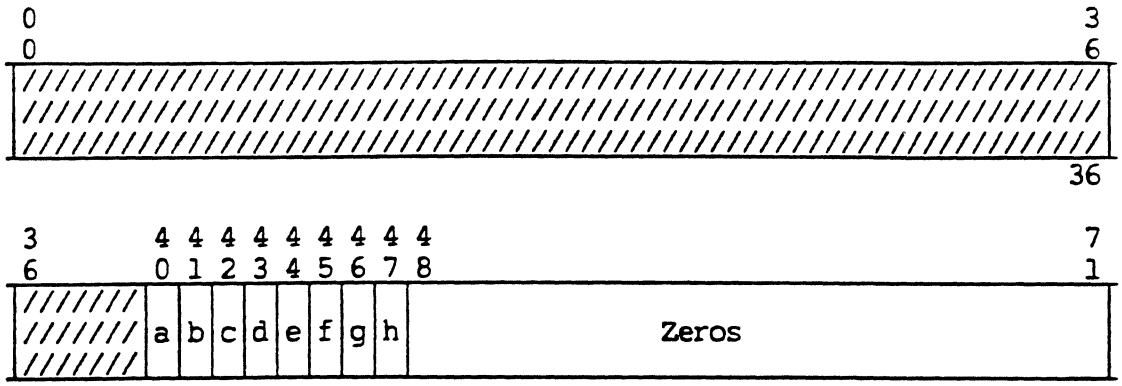


Figure 4-43. Memory Error Status Register Format

Description:

Eight bits in a 72-bit register hold the error status of each memory board. The error conditions occurring on each active board memory cycle are entered in the error status register. Indication of the error is given on the error output line.

Function:

An error output is issued when any error occurs on the current cycle or when the error-register refresh-fault bit was set on an earlier cycle. The memory-error-status register is read and set by the Privileged Master Mode instructions, RMR and SMR, respectively. The memory error status register is reset when a read or write status command cycle occurs, or when memory is initialized.

The contents of the eight status bits is as follows:

Key	Bits	Function
a	40	A15-A22 address parity error
b	41	A7-A14 address parity error
c	42	A0-A6 address parity error
d	43	CM0-CM3 command parity error
e	44	Refresh fault
f	45	Timing generator parity error
g	46	Unit selected during a busy error
h	47	Illegal command or write in logical mode (WMID) error or select parity error
	48-71	All other bits are zero.

MEMORY IDENTIFICATION REGISTER (MID)

Format: 72 bits

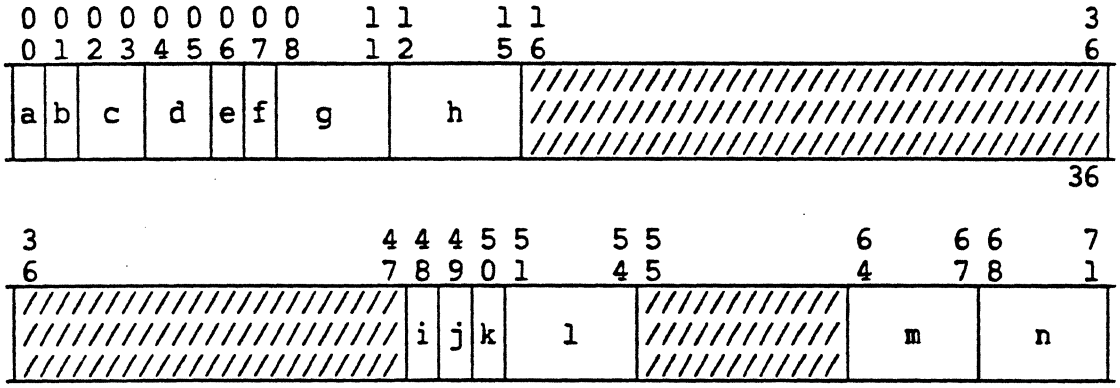


Figure 4-44. Memory Identification Register (MID)

Description:

A 72-bit memory identification (MID) register is located on each memory board to indicate whether or not the board is present, to reflect status, and define physical characteristics of the board.

Function:

The MID register is read and set by the Privileged Master mode instructions RMID and SMID, respectively.

The contents of the constituent fields are as follows:

Key	Bits	Function
a	0	Memory board present 0 = not present 1 = present
b	1	Memory clear status 0 = complete 1 = clear is active
c	2-3	Number of memory units per board. 00 = 1 01 = 4 10 = 2 11 = 8
d	4-5	Size of memory unit 00 = 1M 01 = 4M 10 = 2M 11 = 8M
e	6	ID select 0 = physical ID select 1 = logical ID select These bits reflect the memory select ID definer of the configuration register.
f	7	Memory unit 0 enable 0 = enable 1 = mask
g	8-11	Memory unit 0 logical ID code
h	12-15	Physical ID. This value is equal to the slot number.
	16-35	Unused
	36-47	Unused

Key	Bits	Function
i	48	Memory unit 1 enable
j	49	Memory unit 2 enable
k	50	Memory unit 3 enable
l	51-54	Memory unit 1 logical ID code
m	64-67	Memory unit 2 logical ID code
n	68-71	Memory unit 3 logical ID code

Bits 7-11, 48-54, 64-71 are set by the SMID instruction. The enable bits apply only for a logical ID select.

SECTION 5

ADDRESS MODIFICATION AND DEVELOPMENT

ADDRESS MODIFICATION FEATURES

Address modification features permit the user to alter an address contained in an instruction (or in an indirect word referenced by an instruction). The address modification procedure is generally directed by the tag field of the instruction or indirect word. Address generation differs between the Non-extended (NS) and Extended (ES) modes depending upon the setting of ISR bit 24. (0 = NS; 1 = ES).

ADDRESS GENERATION IN THE NS MODE

Basic Modification

Address modification is performed in four basic ways: Register (R), Register Then Indirect (RI), Indirect Then Register (IR), Indirect Then Tally (IT). A fifth way, address register modification, is discussed later in this section under "Address Modification With Address Registers". Each of these basic types has variations in which selectable registers can be substituted for R in R, RI, and IR and in which various tallying or other substitutions can be made for T in IT. I indicates indirect address modification and is represented by the asterisk placed in the variable field of the program statement as *R or R* when IR or RI is specified. To indicate IT modification, only the substitution for T appears in the variable field; the asterisk is not used.

Indirect Addressing

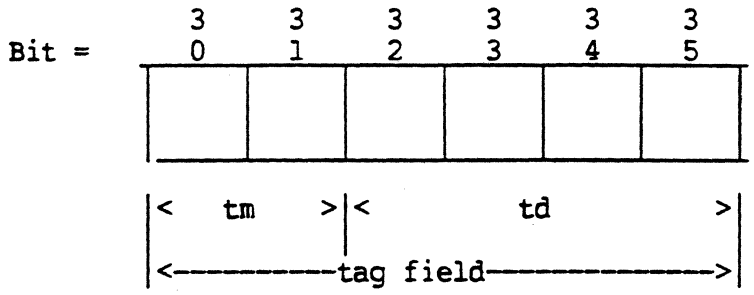
Generally, in indirect addressing, the content of bits 0-17 in the word addressed by the instruction address (y) is treated as another address, rather than as the operand of the instruction. Indirect address modification is performed by the hardware whenever called for by a program instruction. When I modification is called for by a program instruction, an indirect word is always obtained from memory. This indirect word may call for I modification again, or it may specify the effective address (Y) to be used for the original instruction. Indirect addressing for RI, IR, and IT modification is indicated by a binary 1 in either position of the tag modifier field (bit positions 30 and 31) of an instruction or indirect word.

NOTE: A "1" in bit position 30 or 31 of an indirect word does not necessarily mean further indirection.

Tag Field

An address modification procedure generally takes place as directed by the tag field of an instruction and the tag field of an indirect word. Repeat mode instructions and character store instructions do not provide for address modification.

The tag field consists of two parts, tag modifier (tm) and tag designator (td), as follows:



where:

tm specifies one of four possible modification types: Register (R), Register Then Indirect (RI), Indirect Then Register (IR), and Indirect Then Tally (IT).

td specifies the activity for each modification type:

1. When tm = R, RI, or IR, td is called the register designator and generally specifies the register to be used in indexing.

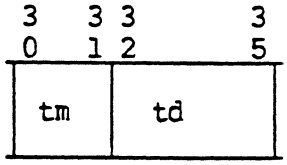
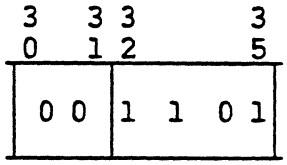
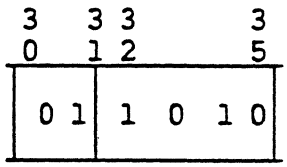
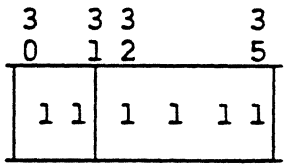
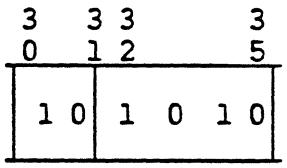
2. When tm = IT, td is called the tally designator and specifies the tallying in detail.

The following table shows the valid assembler mnemonics for address modification and their relationship to the classes R, RI, IR, and IT.

td	tm=00 R	tm=01 RI	tm=11 IR	tm=10 IT
00	Blank	*		
00	N	N*	*N	F
01	AU	AU*	*AU	--
02	QU	QU*	*QU	--
03	DU	--	*DU	--
04	IC	IC*	*IC	SD
05	AL	AL*	*AL	SCR
06	QL	QL*	*QL	--
07	DL	--	*DL	--
10	0	0*	*0	CI
11	1	1*	*1	I
12	2	2*	*2	SC
13	3	3*	*3	AD
14	4	4*	*4	DI
15	5	5*	*5	DIC
16	6	6*	*6	ID
17	7	7*	*7	IDC

Types Of Address Modification

The four basic modification types, their mnemonic substitutions as used in the variable field of the program statement, and their binary forms are as follows:

Modification Type	Variable Field	Binary Forms	Example
			
R	BETA,(R)		BETA,5
RI	BETA,(R)*		BETA,2*
IR	BETA,*(R)		BETA,*7
IT	BETA,(T)		BETA,SC

The parentheses enclosing R and T indicate that substitutions should be made by the user for R and T as explained under the separate discussions of R, IR, RI, and IT modification below. Binary equivalents of the substitution are used in the tm subfield.

REGISTER (R)

The processor performs register address modification whenever an R-type variation is coded. The assembler places binary zeros in both positions of the tm subfield of the instruction. Accordingly, 1 of 16 variations under R are performed by the processor, depending upon bit configurations generated by the assembler, and placed in the designator subfield (td) of the general instruction. The 16 variations, their mnemonic substitutions used on the assembler coding sheet, the td field binary forms presented to the processor, and the effective address Y generated by the processor are indicated below.

R modification allows for the use of the instruction address field as the operand. This is called direct operand address modification, of which there are two types: Direct Upper (DU) and Direct Lower (DL). With the DU variation, the address field of the instruction serves as bit positions 0-17 of the operand and zeros serve as bit positions 18-35 of the operand. With the DL variation, the address field of the instruction serves as bit positions 18-35 of the operand and zeros serve as bit positions 0-17 of the operand.

IC modification should only be used with an absolute operand. A relative operand that has IC modification is flagged with a possible relocation error (R) by the assembler.

Modification Variation	Mnemonic Substitution	Binary Form (td field)	Effective Address
(R)=X0	0	1000	Y=y+C(X0)
=X1	1	1001	Y=y+C(X1)
=X2	2	1010	Y=y+C(X2)
=X3	3	1011	Y=y+C(X3)
=X4	4	1100	Y=y+C(X4)
=X5	5	1101	Y=y+C(X5)
=X6	6	1110	Y=y+C(X6)
=X7	7	1111	Y=y+C(X7)
=A 0-17	AU	0001	Y=y+C(A) 0-17
=A 18-35	AL	0101	Y=y+C(A) 18-35
=Q 0-17	QU	0010	Y=y+C(Q) 0-17
=Q 18-35	QL	0110	Y=y+C(Q) 18-35
=IC	IC	0100	Y=y+C(IC)
direct upper	DU	0011	Bits 0-17 of operand = y; bits 18-35 of operand = 0
direct lower	DL	0111	Bits 0-17 of operand = 0; bits 18-35 of operand = y
=None	Blank or N	0000	Y=y
=Any symbolic index register	Any defined symbol [1]		

1. Symbol must be defined as one of the index registers by using an applicable pseudo-operation (EQU or BOOL).

The following examples show how R-type modification variations are entered and how they affect effective addresses.

EXAMPLES:

	1	8	16	Effective Address
(1)		EAX0	1	
		LDA	B,0	Y=B+1
(2)		LDA	=2,DL	
		LDA	C,AL	Y=C+2
(3)		EAQ	3	
		LDA	M,QU	Y=M+3

	1	8	16	Address
(4)	ABC	LDA	-2,IC	Y=ABC-2
(5)	XYZ	LDA	*,DU	operand =XYZ, operand =0 0-17 18-35
(6)		EAX7	ABC	
		LDA	1,7	Y=ABC+1
(7)		LDA	2,DL	operand =0,operand =2 0-17 18-35
(8)		LDA	B	Y=B
(9)		LDA	B,N	Y=B
(10)		EAX	ALPHA,10	
		LDA	C,ALPHA	
	ALPHA	EQU	2	Y=C+10

Coding examples of R-type modification follow:

- (R) = N

```
ALPHA  LDA     ADRES1,N
```

is equivalent to

```
ALPHA  LDA     ADRES1
```

No address modification results; ADRES1 is the effective operand.

o (R) = Xn where n = 0 to 7

```
ALPHA  LDA     ADRES2,5
```

X5 contains the value 2.

```
ADRES2 DEC     12
       OCT     7777
       OCT     123456765432
```

ADRES2+2 becomes the effective address and its contents (octal 123456765432) are loaded into the A-register.

	A-register	X5
Before	773412315026	000002
After	123456765432	000002

o (R) = AU, AL, QU, QL

```
ALPHA  LDA     ADRES3,QU
```

Bits 0-17 of the Q-register contain the value 3.

```
ADRES3 DEC     10
       OCT     12
       OCT     14
       OCT     16
```

ADRES3+3 becomes the effective address and its contents (octal 16) are loaded into the A-register.

	A-register	Q-register	
Before	123456765432	000003	123456
After	000000000016	000003	123456

- (R) = DU,DL

```
ALPHA  LDA     ADRES4,DU
```

There is no memory access to obtain modification of ADRES4. The address represented by the symbol ADRES4 is placed in bits 0-17 of the A-register; bits 18-35 are filled with zeros.

```
ADRES4  OCT    10   (assume ADRES4 is at location 001002 octal)
```

Before	0 0 0 0 0 0 0 0 0 0 1 6
After	0 0 1 0 0 2 0 0 0 0 0 0

A simple program segment, the movement of 50 words from ABC to XYZ, may help illustrate the power of address modification.

Without Address Modification

```
1       8       16
START   LDX1    =0B17,DU
        LDA     ABC
        STA     XYZ
        LDA     =1B17
        ASA     START+1
        ASA     START+2
        ADLX1   =1B17
        CMPX1   =50B17
        TNC     START+1
```

With Address Modification

```
1       8       16
START   LDX1    0,DU
        LDA     ABC,1
        STA     XYZ,1
        ADLX1   1,DU
        CMPX1   50,DU
        TNC     START+1
```

REGISTER THEN INDIRECT (RI)

Register Then Indirect address modification is a combination in which both indexing (register modification) and indirect addressing are performed. For indexing modification under RI, the mnemonic substitutions for R are the same as those given under the discussion of register (R) modification with the exception that DU and DL are invalid for RI usage. For indirect addressing (I), the processor interprets the contents of the operand address associated with the original instruction or with an indirect word.

Under RI modification, the effective address Y is found by first performing the specified register modification on the operand address of the instruction; the result of this R modification under RI is the address of an indirect word which is then retrieved. (Refer to Figure 5-1.)

After the indirect word has been accessed from memory and decoded, the processor carries out the address modification specified by this indirect word. If the indirect word specifies RI, IR, or IT modification (any type specifying indirection), the indirect sequence is continued. When an indirect word is found that specifies R modification, the processor performs R modification, using the register specified by the td field of this last-encountered indirect word and the address field of the same word, to form the effective address Y.

The variations DU and DL of register modification (R), when used with Register Then Indirect modification (RI), cause an Illegal Procedure (IPR) fault.

To refer to an indirect word from the instruction itself without including register modification of the operand address, the "no modification" variation should be specified; under RI modification, this is indicated by placing only an asterisk (*) in the tag position.

The following examples illustrate the use of RI modification, including the use of (R) = N (no register modification). The asterisk appearing in the modifier subfield is the assembler symbol for I (Indirect). The address-subfield, single-symbol expressions shown are not intended as realistic coding examples, but to show the relation between operand addresses, indirect addressing, and register modification.

EXAMPLES:

```
                                                    Modification    Effective
      1         8         16                          Type          Address
(1)             EAA       1
                EAX1      2
                STA       Z,AU*                     (RI)            Y=B+2
          Z     ORG       Z+1
                ARG       B,1                       (R)

(2)             EAQ       3
                MPY       Z,*                       (RI)            Y=B+3
                .
          Z     ARG       B,QU                      (R)

(3)             EAX3      3
                EAX5      5
                STQ       Z,*                       (RI)            Y=M
                .
          Z     ARG       B,5*                      (RI)
                ORG       B+5
                ARG       C,3*                      (RI)
                ORG       C+3
                ZERO      M                         (R)
```

Coding examples of RI modification follow:

- (RI) = N*

```
ALPHA  LDA     ADRES1,N*
```

is equivalent to

```
ALPHA  LDA     ADRES1,*
```

The indirect word at ADRES1 is obtained; if this indirect word specifies further indirect modification, the process continues until an indirect word is obtained with (R) modification.

- (RI) = (Xn)* where n = 0 to 7

```
       EAX5    5
       EAX2    2
ALPHA  LDA     ADRES2,5*
```

The indirect word at ADRES2+5 is obtained. If the indirect word at this location is

```
       LDQ     ADRES3,2
```

the effective address is ADRES3+2.

INDIRECT THEN REGISTER (IR)

Indirect Then Register address modification is a combination in which both indirect addressing and indexing (register modification) are performed. IR modification is not a simple inverse of RI; several important differences exist.

Under IR modification, the processor first fetches an indirect word from the memory location specified by the address field y of the machine instruction; the C(R) of IR are safe stored for use in making the final index modification to develop the effective address Y.

Next, the address modification, if any, specified by this first indirect word is examined. If this modification is again IR, another indirect word is retrieved from storage immediately; and the new C(R) are safe stored, replacing the previously safe stored C(R). If an IR loop develops, the above process continues, each new C(R) replacing the previously safe stored C(R), until a type other than IR is encountered in the sequence.

If the indirect sequence produces an RI indirect word, the R-type modification is performed immediately to form another address; but the I of this RI treats the contents of the address as an indirect word. The chain then continues with the C(R) of the last IR still safe stored, awaiting final use. At this point the new indirect word might specify IR-type modification, possibly renewing the IR loop noted above; or it might initiate an RI loop. In the latter case, when this loop is broken, the remaining modification type is R or IT.

When either R or IT is encountered, it is treated as type R, where R is the last safe stored C(R) of an IR modification. At this point the safe stored C(R) is combined with the y of the indirect word that produced R or IT, and the effective address Y is developed.

If an indirect modification without register modification is desired, the "no modification" variation (N) of register modification should be specified in the instruction. This normally will be entered on coding sheets as *N in the modifier part of the variable field. (The entry * alone is equivalent to N* under RI modification and must be used in that way.)

EXAMPLE 1:

(IR) = *N

```
ALPHA   LDA     ADRES1,*N
```

The indirect word at ADRES1 is obtained. If the indirect word at this location is:

```
ADRES1  LDQ     ADRES2
```

the effective address is ADRES2

EXAMPLE 2:

IR and then R or IT

(IR) = *(X<u>n</u>) where <u>n</u> = 0 to 7

```
        EAX5    15

ALPHA   LDA     ADRES1,*5
```

The indirect word at ADRES1 is obtained. If the indirect word is:

```
ADRES1  LDQ     ADRES2,(R)
```

or

```
ADRES1  LDQ     ADRES2,(T)
```

the effective address is ADRES2+15

EXAMPLE 3:

IR and then RI

(IR) = *(Xn) where n = 0 to 7

```
        EAX5    16
        EAX2    17
ALPHA   LDA     ADRES1,*5
ADRES1  LDQ     ADRES2,2*
          .
          .
          .
        LDA     ADRES4    ( in ADRES2+17)
```

the effective address is ADRES4+16

EXAMPLE 4:

IR and then IR

(IR) = *(Xn) where n = 0 to 7

```
        EAX5    18
        EAX3    19
ALPHA   LDA     ADRES1,*5
ADRES1  LDA     ADRES2,*3
ADRES2  LDA     ADRES3
```

the effective address is ADRES3+19

The following examples illustrate the use of IR-type modification, intermixed with R and RI types, under the several conditions noted above.

EXAMPLES:

	1	8	16	Modification Type	Effective Address
(1)		LDQ	1,DL		
		LDA	Z,*QL	(IR)	Y=M+1
	Z	ARG	M	(R)	

	1	8	16	Modification Type	Effective Address
(2)		EAX3	2		
		EAX5	3		
	ABC	LDA	Z,*3	(IR)	Y=C+2
	Z	ARG	B,5*	(RI)	
		ORG	B+3		
		ARG	C,IC	(R)	
(3)		EAX3	4		
		EAX5	5		
		EAQ	6		
		EAX7	7		
		LDA	Z,*3	(IR)	Y=M+6
	Z	ARG	B,*5	(IR)	
	B	ARG	C,*QU	(IR)	
	C	ARG	M,7	(R)	
(4)		EAX3	8		
		LDQ	9,DL		
		LDA	Z,*DL	(IR)	
					$C(A)_{18-35}=M$
	Z	ARG	B,3*	(RI)	
		ORG	B+8		
		ARG	M,QL	(R)	
(5)		LDA	10,DL		
		LDA	Z,*AL	(IR)	Y=B+10
	Z	ARG	B,AD	(IT)	
(6)		EAX3	11		
		LDA	Z,*N	(IR)	Y=B
	Z	ARG	B,3	(R)	

	1	8	16	Modification Type	Effective Address
(7)		EAX5	12		
		LDA	Z,*N	(IR)	
	Z	ARG	B,*5	(IR)	
	B	ARG	M,DU	(R)	
(8)		EAX5	13		
		LDA	Z,*	(RI)	Y=M+13
	Z	ARG	B,*5	(IR)	
	B	ARG	M,DU	(R)	
(9)		EAX1	14		
		LDA	X,*	(RI)	Y=Z+14
	X	ARG	B,*1	(IR)	
	B	ARG	Z,ID	(IT)	
	Z	TALLY	A,10	(IT)	

INDIRECT THEN TALLY (IT)

Indirect Then Tally address modification is a combination in which both indirect addressing and automatic incrementing/decrementing of fields in the indirect word are performed as hardware features, thus relieving the user of these responsibilities. The automatic tallying and other functions of IT modification allow processing of tabular data in memory, provide a means for working upon character data, and allow termination on user-selectable numeric tally conditions. When tally runout occurs, bit 25 in the indicator register is set. If an unassigned IT tag is used, an Illegal Procedure (IPR) fault occurs.

The variations under IT modification are summarized below. The mnemonic substitution for IT is (T); the designator I for indirect addressing in IT is not represented. (Note that one of the substitutions for T is I.)

Variation	Mnemonic Substitution	Binary Form (td Field)	Effect on Processor and Indirect (Tally) Word for Each Reference
Fault	F	0000	None. A Fault Tag fault is generated. The indirect word is not examined.
Character indirect	CI	1000	None. Applies to TALLY, TALLYB.
Sequence character	SC	1010	Obtain the operand address from the tally word; then add 1 to the character position value in the tag field and subtract 1 from the tally count field; add 1 to the address field and set the character position value to zero when the character position crosses a word boundary. Applies to TALLY, TALLYB.
Sequence character reversed	SCR	0101	Subtract 1 from the character position value in the tag field and add 1 to the tally count field; subtract 1 from the address field and set the character position value to 3 (TALLYB) or 5 (TALLY) when the character position crosses a word boundary. Then obtain the operand address from the tally word. Applies to TALLY, TALLYB.
Indirect	I	1001	None. The operand address is the word to which the tally word address field refers. Applies to all tally pseudo-operations.
Increment address, decrement tally	ID	1110	Obtain the operand address from the tally word; add 1 to the address field and subtract 1 from the tally count field. Applies to all tally pseudo-operations.
Decrement address, increment tally	DI	1100	Subtract 1 from the address field, add 1 to the tally count field, and then obtain the operand address from the tally word. Applies to all tally pseudo-operations.

Variation	Mnemonic Substitution	Binary Form (td Field)	Effect on Processor and Indirect (Tally) Word for Each Reference
Increment address, decrement tally, and continue	IDC	1111	Obtain the operand address from the tally word, add 1 to the address field, and subtract 1 from the tally count field. Additional address modification will be performed as specified by the tag field. Applies to TALLYC. Results in IPR fault in ES mode.
Decrement address, increment tally, and continue	DIC	1101	Subtract 1 from the address field, add 1 to the tally count field, and then obtain the operand address from the tally word. Additional address modification will be performed as specified by the tag field. Applies to TALLYC. Results in IPR fault in ES mode.
Add delta	AD	1011	Obtain the operand address from the tally word, add an increment to the address field, and subtract 1 from the tally count field. Applies to TALLYD.
Subtract delta	SD	0100	Subtract an increment from the address field, add 1 to the tally count field, and then obtain the operand address from the tally word. Applies to TALLYD.

Indirect Word Format

The location of the indirect word is specified by the address field (y) of the instruction or previous indirect word (IDC or DIC). IT modification causes the indirect word to be fetched and interpreted as specified by the td subfield of the instruction or previous indirect word that referred to the indirect word.

The format of the indirect word is shown in Figure 5-1.

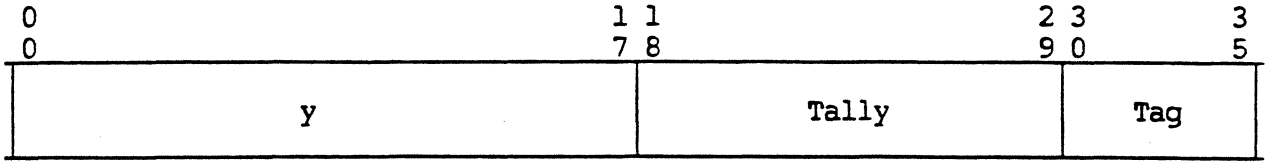


Figure 5-1. Indirect Word Format

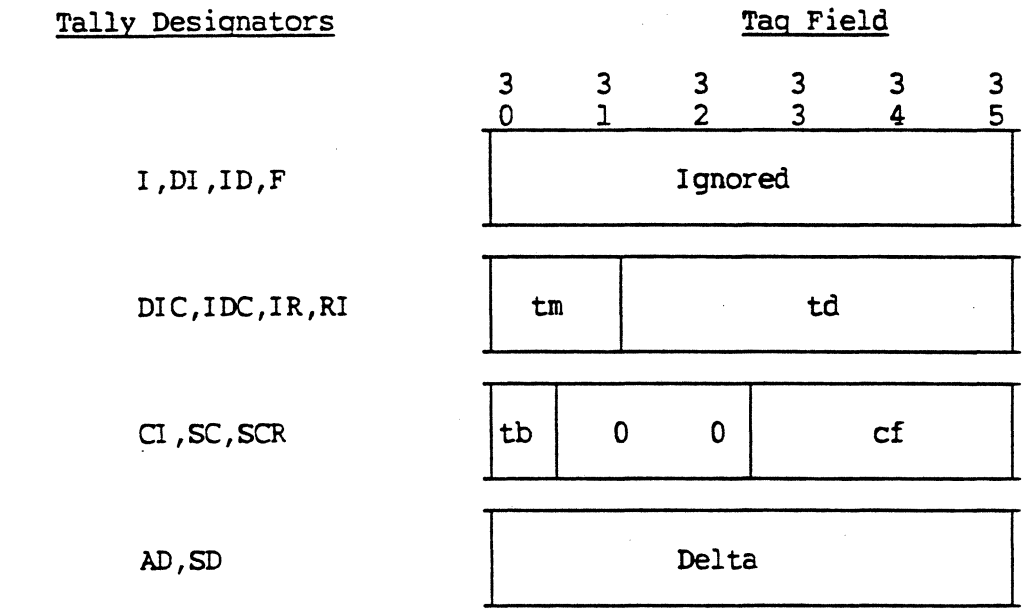
where:

y - address field

Tally - tally field (ignored except for tally modification)

Tag - tag field

Depending upon the prior tally designator, the tag field for the indirect word is used in one of the following ways:



where:

tm - tag modifier

td - tag designator

tb - character size indicator (0=6-bit, 1=9-bit)

cf - character position field

Delta - delta field (Size of increment)

Variations Under IT Modification

Fault (T) = F Variation. The Fault variation enables the user to force program transfers to operating system routines or to corrective routines during the execution of an address modification sequence by causing a Fault Tag fault. (This will usually indicate some abnormal condition for which the user desires protection.)

Character Indirect (T) = CI Variation. The Character Indirect (CI) variation allows operations on the A register or Q register where repeated reference to a single character in memory is required. The character size field (tb) of the indirect word specifies the character size.

For this variation, the effective address is the address field of the CI indirect word obtained via the tentative operand address of the instruction or preceding indirect word that specified the CI variation. The character position field (cf) of the indirect word is used to specify the character to be involved in the operation.

This variation is similar to the SC variation except that no incrementing or decrementing of the address, tally, or character position is performed.

EXAMPLES:

```
        Z       TALLY   B,,4        6-bit char. addressing
        1       8       16
(2)             LDA     ADDR,CI
                .
        ADDR    TALLY   ADD,,3      6-bit char. addressing
                or
        ADDR    TALLYB  ADD,,3      9-bit char. addressing
```

The effective address is ADD. The character in character position 3 is loaded into the A-register in character position 5 for 6-bit characters or into position 3 for 9-bit characters. The remainder of the A-register is loaded with all zero bits.

Sequence Character (T) = SC Variation. The Sequence Character (SC) variation is provided for sequential access to 6-bit or 9-bit characters. The character size field (tb) of the indirect word is used to specify the character size. Processor instructions that do not allow SC operations are so indicated in the individual instruction descriptions. The operand address is obtained from the address field of the indirect word referenced by the word containing the SC tag.

Characters are operated on in sequence from left to right within the machine word. The character position field (cf) of the indirect word is used to specify the character position to be involved in the operation. The Tally Runout indicator is set when the tally field of the indirect word reaches 0.

EXAMPLE:

```
1       8       16                          32
        LDA     A,SC
A       TALLY   TABLE,70,4                  6-bit char. addressing
TABLE   BSS     13
```

in which 70 is the count and 4 designates the character position of the tally start.

For register loads using the SC variation, a character is fetched from the indicated position of the memory location and is written into the lower end of the register; the remaining bits of the register are set to zero. For stores under the SC variation, a character is fetched from the lower end of the register and written into the indicated position in the memory location; the remaining character positions in the memory location remain unchanged.

The tally field of the indirect word is used to count the number of times a reference is made to a character. Each time an SC reference is made to the indirect word, the tally is decremented by 1, and the character position is incremented by 1 to specify the next character position. The tally runout indicator is set when the tally reaches 0. When character position 5 (for 6-bit characters) or 3 (for 9-bit characters) is incremented, it is changed to position 0 and the address field of the indirect word is incremented by 1. All incrementing and decrementing are done after the effective address has been provided for the current instruction execution. The effect of successive references using SC modification is shown in the following examples.

EXAMPLES:

```
1       8       16                  Effective   Character
                                    Address     Position    Reference

        LDA     Z,SC                B           0           1

Z       TALLY   B,80,0              B           1           2
B       BSS     14                  .           .           .
                                    .           .           .
                                    B           5           6
                                    B+1         0           7
                                    .           .           .
The Tally Runout indicator          .           .           .
is set on the 80th reference.
                                    B+n         0           6n+1
                                    .           .           .
                                    .           .           .
```

```
1       8       16
ADD1    LDA     ADDR,SC

        TTF     ADD1
        .
        .
ADDR    TALLY   ADD,12,3  (6-bit characters)

          or

ADDR    TALLYB  ADD,12,3  (9-bit characters)

ADD     BSS     4
```

The first effective address is ADD. The character in character position 3 is loaded into the A-register in position 5 (for 6-bit characters) or into position 3 (for 9-bit characters). The second reference will load ADD character 4 (if 6-bit) or ADD+1 character 0 (if 9-bit), etc. The tally is decremented from 12 to 0. The destination in the A-register does not change.

Sequence Character Reverse (T) = SCR Variation. The SCR variation is the reverse of SC. The character position is decremented by 1 and the tally is incremented by 1 before the indirect word address field and character position are used as the operand character address. When the character position attempts to go negative, it is set to the maximum value (3 or 5) and the address is decremented by 1.

Indirect (T) =I Variation . The Indirect (I) variation of IT modification is, in effect, a subset of the ID and DI variations described below in that all three -- I, ID, and DI -- make use of one indirect word in order to refer to the operand. The I variation is functionally unique, however, in that the indirect word accessed by an instruction remains unaltered; no

incrementing/decrementing of the address field or tallyoccurs. Since the tag field of the indirect word under I is not interrogated, this word will always terminate the indirect chain.

The following differences in the coding and the effects of *, *N, and I should be observed:

1. RI modification is coded as R* for all cases, excluding R=N.

   For R=N under RI, the modifier subfield can be written as N* or as * alone, according to preference.

   When N* or just * is coded, the assembler generates a machine word with octal 20 in bit positions 30-35; octal 20 causes the processor to add 0 to the address field y of the word containing the N* or * and then to access the indirect word at memory location y.

2. IR modification is coded as *R for all cases, including R=N.

   For R=N under IR, the modifier subfield must be written as *N.

   When *N is coded, the assembler generates octal 60 in bit positions 30-35 of the associated machine word; octal 60 causes the processor to (1) retrieve the indirect word at the location (y) specified by the machine word, and (2) effectively safe store zeros (for possible final index modification of the last indirect word).

3. IT modification is coded using only a variation designator (I, ID, DI, SC, SCR, CI, AD, SD, F, IDC, or DIC); that is, no asterisk (*) is written. Thus, a written IT address modification appears as ALPH,DI; BETA,AD; etc.

   For the variation I under IT, the assembler generates a machine word with octal 51 in bit positions 30-35; 51 causes the processor to examine one, and only one, indirect word to be retrieved from memory to obtain the effective address Y.

EXAMPLE:

```
1       8       16                      Modification    Effective
                                            Type         Address
        EAX5    1
        LDA     Z,I             (IT)                    Y=B

Z       ARG     B,*5            (IR)
```

Increment Address, Decrement Tally (T) = ID Variation. The ID variation under IT modification provides automatic (hardware) incrementing or decrementing of an indirect word that is best used for processing tabular operands (data located at consecutive memory addresses). The indirect word always terminates the indirect chain.

In the ID variation, the effective address is the address field of the indirect word obtained via the tentative operand address of the instruction or preceding indirect word, whichever specified the ID variation. Each time such a reference is made to the indirect word, the address field of the indirect word is incremented by 1 and the tally portion of the indirect word is decremented by 1. The incrementing and decrementing are performed after the effective address is provided for the instruction operation. When the tally reaches zero, the Tally Runout indicator is set.

EXAMPLES:

```
1       8       16          Modification Type      Effective Address      Reference

        LDA     Z,ID        (IT)                   B                      1

 Z      TALLY   B,12         word addressing        B+1                    2
 B      BSS     12                                   .                      .
                                                     .                      .
                                                     B+n                    n+1
 The Tally Runout indicator is                       .                      .
 set on the 12th reference.                          .                      .
```

```
1       8       16
ADRES1  LDA     ADRES2,ID

        TTF     ADRES1
          .
          .
ADRES2  TALLY   ADRES3,10   word addressing

ADRES3  BSS     10
```

The first effective address is ADRES3; the second is ADRES3 plus 1, etc. The tally is decremented from 10 to zero. The TTF instruction checks the Tally Runout indicator. If the tally is not zero, transfer is made to ADRES1. If the tally is zero, processing continues with the instruction following TTF. Without the TTF instruction, only one effective address is obtained.

Decrement Address, Increment Tally (T) + DI Variation. The DI variation under IT modification provides automatic (hardware) incrementing and decrementing of an indirect word that is best used for processing tabular operands (data located at consecutive memory addresses). The indirect word always terminates the indirect chain.

In the DI variation, the effective address is the modified address field (1 less than the value before modification) of the indirect word obtained via the tentative operand address of the instruction or preceding indirect word, whichever specified the DI variation. Each time a DI reference is made to the indirect word, the address field of the indirect word is decremented by 1 and the tally portion is incremented by 1. When the tally is incremented from 7777 to 0, the tally runout indicator is set. The incrementing and decrementing are performed prior to providing the effective address for the current instruction operation.

EXAMPLES:

1	8	16	Modification Type	Effective Address	Reference
	LDA	Z,DI	(IT)	B-1	1
Z	TALLY	B,-18	word addressing	B-2	2
	.	.		.	.
B	BFS	18		.	.
				B-n	n
The Tally Runout indicator is set on the 18th reference; there, the 12-bit tally field in the indirect word overflows and becomes all zeros.				.	.
				.	.

1	8	16	Modification Type	Effective Address	Reference
ADRES1	LDA	ADRES2,DI			
.					
	TTF	ADRES1			
.					
.					
ADRES2	TALLY	ADRES3,-10	word addressing		
ADRES3	BFS	10			

The first effective address is ADRES3 -1; the second is ADRES3 -2; etc. The tally increases from -10 to 0.

Increment Address, Decrement Tally, and Continue (T) = IDC Variation. The IDC variation under IT modification functions in a manner similar to the ID variation except that, in addition to automatic incrementing/decrementing, it permits the user to continue the indirect chain in obtaining the instruction operand. Where the ID variation is useful for processing tabular data, the IDC variation permits processing of scattered data by a table of indirect pointers. More specifically, the ID portion of this variation provides the ability to sequentially step through a table and the C portion (continuation) allows indirection through the tabular items. The tabular items may be data pointers, subroutine pointers, or a transfer vector.

The address and tally fields are used as described under the ID variation. The tag field uses the set of instruction address modification variations under the following restrictions: no variation is permitted that requires an indexing modification in the IDC cycle since the indexing adder is in use by the tally phase of the operation. Thus, permissible variations are any allowable form of IT or IR; but if RI or R is used, R must equal N.

EXAMPLES:

1	8	16	Modification Type	Effective Address	Reference
	LDA	Z,IDC	X	1	
Z	TALLYC	B,10,I	Y	2	
B	ARG	X	Z	3	
	ARG	Y	.	.	
	ARG	Z	.	.	

The Tally Runout indicator is set on the 10th reference.

```
1       8       16              32
ADRES1  LDA     ADRES2,IDC
        TTF     ADRES1

ADRES2  TALLYC  ADRES3,4,*      word addressing and indirect
ADRES3  ARG     AD1
        ARG     AD2
        ARG     AD3
        ARG     AD4
```

AD1 is the first effective address, AD2 is the second, AD3 is the third, and AD4 is the fourth.

Decrement Address, Increment Tally, and Continue (T) = DIC Variation. The DIC variation under IT modification performs in much the same way as the DI variation except that, in addition to automatic decrementing or incrementing, it permits the user to continue the indirect chain in obtaining an instruction operand. The continuation function of DIC operates in the same manner and under the same restrictions as IDC except that (1) it increments in the reverse direction, and (2) decrementing/incrementing is performed prior to obtaining the effective address from the tally word. (Refer to the first example under IDC; work from the bottom of the table to the top.) DIC is especially useful in processing last-in, first-out lists. Some examples follow:

1	8	16	Modification Type	Effective Address	Reference
	LDA	Z,DIC	(IT)		
Z	TALLYC	B,-10,I	(IT)	Y	1
	ARG	Z		X	2
	ARG	X		Z	3
	ARG	Y		.	.
				.	.
B	NULL				

Assuming an initial tally of -10, the Tally Runout indicator is set on the 10th reference; there, the 12-bit tally field in the indirect word overflows and becomes all zeros.

EXAMPLES:

```
1       8       16                  32
ADRES1  LDA     ADRES2,DIC
        TTF     ADRES1

ADRES2  TALLYC  ADRES3,-4,*N        word addressing and indirect
        ARG     AD4,*
        ARG     AD3
        ARG     AD2,*N
        ARG     AD1,*N
ADRES3  BSS     1
AD1     ARG     A
AD2     ARG     B
AD4     ARG     C
```

A is the first effective address, B is the second, AD3 is the third, and C is the fourth.

Add Delta (T) = AD Variation. The Add Delta (AD) variation is provided for programming situations where tabular data to be processed is stored at equally spaced locations, such as data items, each occupying two or more consecutive memory addresses. It functions in a manner similar to the ID variation, but the incrementing (delta) of the address field is selectable by the user.

Each time such a reference is made to the indirect word, the address field of the indirect word is increased by delta and the tally portion of the indirect word is decremented by 1. The addition of delta and decrementing are done after the effective address is provided for the instruction operation.

The following examples show the effect of successive references using AD modification:

```
1       78      16          Modification    Effective
                            Type            Address     Reference

        LDAQ    Z,AD        (IT)            B           1

Z       ETALLY  B,20,2                      B+2         2
B       EBSS    40                          B+4         3
                                             .           .
                                             .           .
                                            B+2n        n+1
The Tally Runout indicator                   .           .
is set on the 20th reference.                .           .
```

```
1       78      16                              32
ADRES1  LDAQ    ADRES2,AD
        TTF     ADRES1
          .
          .
ADRES2  ETALLYD ADRES3,10,2             word addressing with DELTA
ADRES3  EBSS    20
```

The first effective address is ADRES3; the second is ADRES3+2. The tally decreases from 10 to 0.

Subtract Delta (T) = SD Variation. The Subtract Delta (SD) variation is useful in processing tabular data in a manner similar to the AD variation except that the table can easily be scanned from back to front using a programmer-specified increment. The effective address from the indirect word is decreased by delta and the tally is increased by 1 each time an SD reference is made to the indirect word. This is done before supplying the operand address to the current instruction, making the SD variation analogous to the DI variation.

Address Modification Octal Codes

Address modification and 2-digit octal codes for each type of modification are listed in Table 5-1.

Table 5-1. Address Modification Octal Codes

HIGH ORDER OCTAL DIGIT	LOW ORDER OCTAL DIGIT 0	1	2	3	4	5	6	7
0	N	AU	QU	DU	IC	AL	QL	DL
1	0	1	2	3	4	5	6	7
2	N*	AU*	QU*		IC*	AL*	QL*	
3	0*	1*	2*	3*	4*	5*	6*	7*
4	F				SD	SCR		
5	CI	I	SC	AD	DI	DIC	ID	IDC
6	*N	*AU	*QU	*DU	*IC	*AL	QL	*DL
7	*0	*1	*2	*3	*4	*5	*6	*7

Address Modification Flowchart

The process of address modification is illustrated in flowchart form in Figure 5-2. Address register modification is not included in this example.

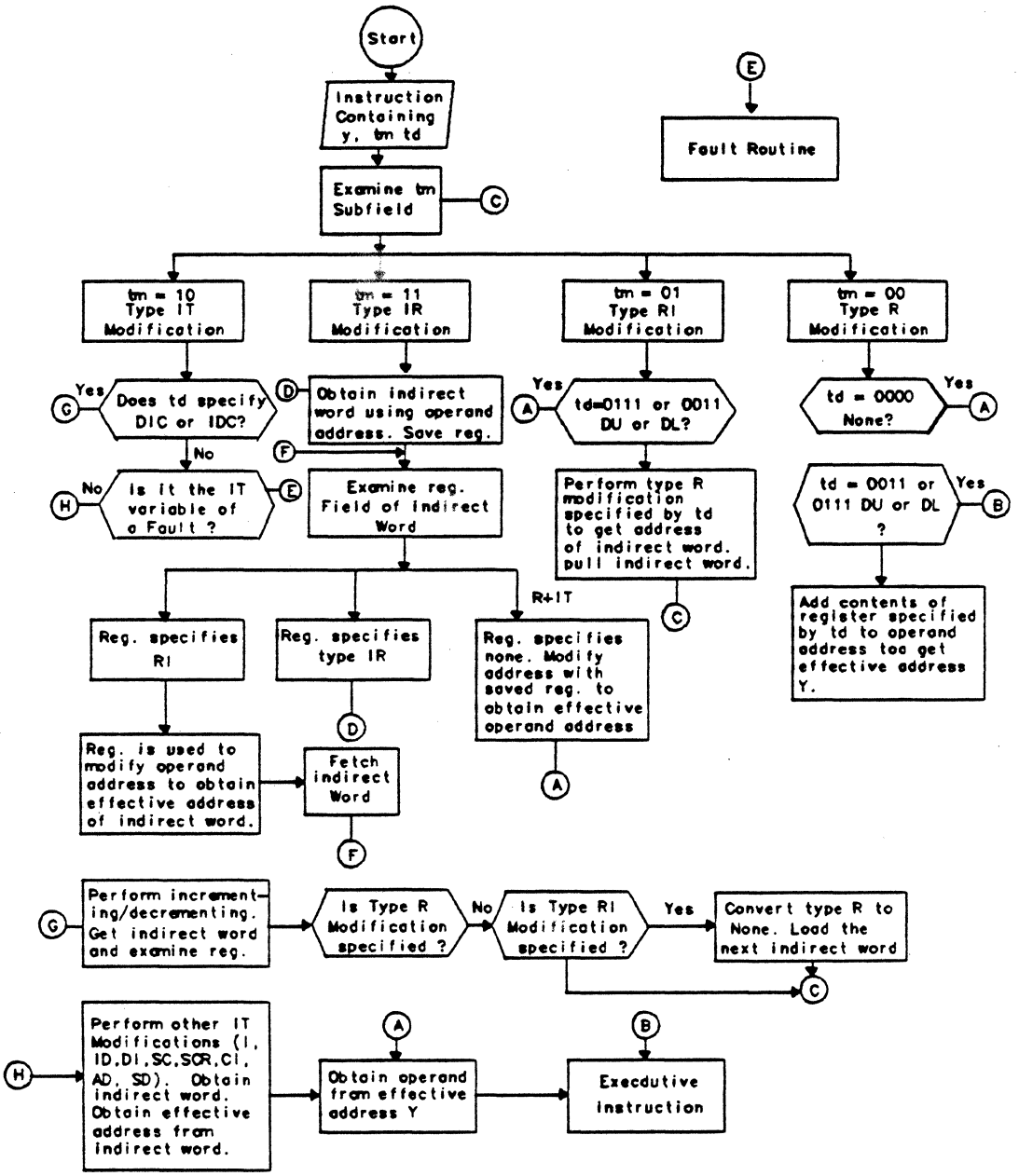


Figure 5-2. Address Modification Flowchart

Floatable Code

Program statements may be written in floatable code. Such statements may then be executed from any location in memory without relocation at load time. Floatable code is created by use of instruction counter (IC) modification in all references to locations within a program. Thus, to transfer to location SYM, the following statement can be written:

```
TRA   SYM-*,IC
```

or

```
TRA   SYM,$
```

The assembler accepts the currency symbol ($) as a valid IC register designator. The following tag fields in a machine instruction are permitted:

Mnemonic	Octal Code
$	04
$*	24

The assembler computes the difference between the value of the address location argument of the variable field and the current location as the content of the address field of the instruction word. The IC is then supplied for modification. *$ is illegal and will be assembled as *IC.

NOTE: The FLOAT pseudo-operation or $ modification does not apply when used with SYMREF symbols or within the range of a BLOCK pseudo-operation.

Address Modification With Address Registers

Address registers (ARn) provide a second-level indexing capability. The address register format allows addressing on a character or bit basis and is used by the character and bit manipulation instructions of the processor. When an address register is used to modify an address in which character and/or bit addressing is not used, the character and bit positions of the address register are ignored.

SINGLE-WORD ADDRESS MODIFICATION

When an address register is to be used in address preparation, its application is specified in the instruction word. All single-word instructions to which address modification is applicable have the same instruction word format as shown in Figure 5-3.

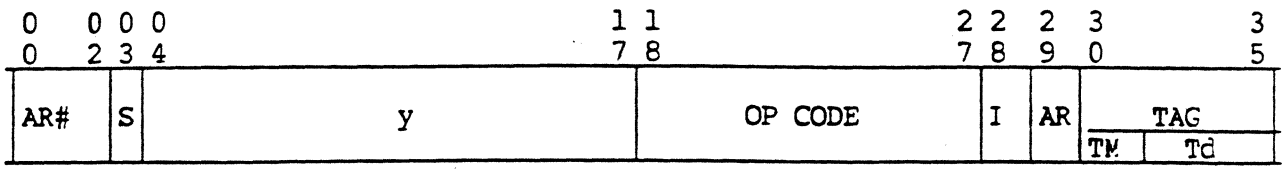


Figure 5-3. Single-Word Instruction Format

AR# - Address register number, if bit 29 = 1

S - Sign bit, if bit 29 = 1

y - Address field bits 0-17 or bits 3-17, depending on the state of bit 29. Must be an absolute value if AR mode is used.

OP CODE - 10-bit operation code field

I - Program interrupt inhibit bit

AR - Address register bit. If bit 29 = 1, use address register specified in bits 0, 1, and 2 of y field for address modification and use operand descriptor register specification in bits 0,1, and 2 of y field as the segment descriptor. Bit 3 (sign) is then extended to bits 0, 1, and 2. If bit 29 = 0, no address register modification is performed and the ISR is used as the segment descriptor.

TAG - Tag field: Used to control address modification

Tm - (Bits 30-31): Type of address modification

Td - (Bits 32-35): Index register or modification variation designator

NOTE: With some instructions, certain address modification is not permitted, and if such modification is specified, an Illegal Procedure fault (IPR) occurs. (Refer to the individual instruction specifications in Section 8.)

The address preparation for a single-word instruction with bit 29 = 1 proceeds as follows:

1. The three most-significant bits of y (0, 1, 2) are decoded to determine which of the eight address registers is to be used.

2. Bit 3 of the y field is extended to fill bit positions 2, 1, and 0, thus forming a two's complement signed number.

3. The two's complement y field is then added to the contents of the specified address register. The character and bit positions of the address register are ignored and the contents of the address register remain unchanged.

4. Address modification continues as specified by the tag field of the instruction word.

Diagramatically, address preparation for a single-word instruction is described below in Figure 5-4.

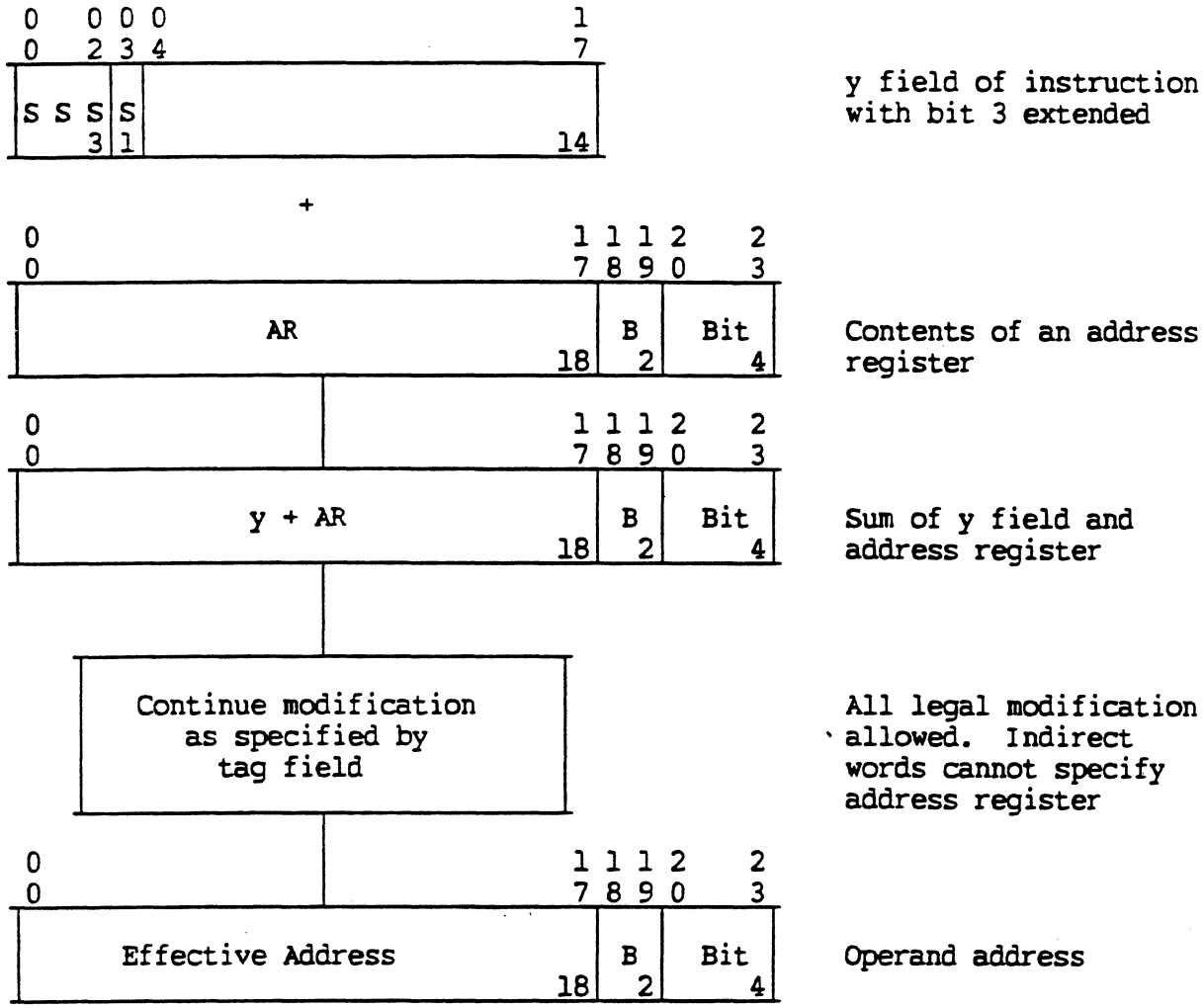
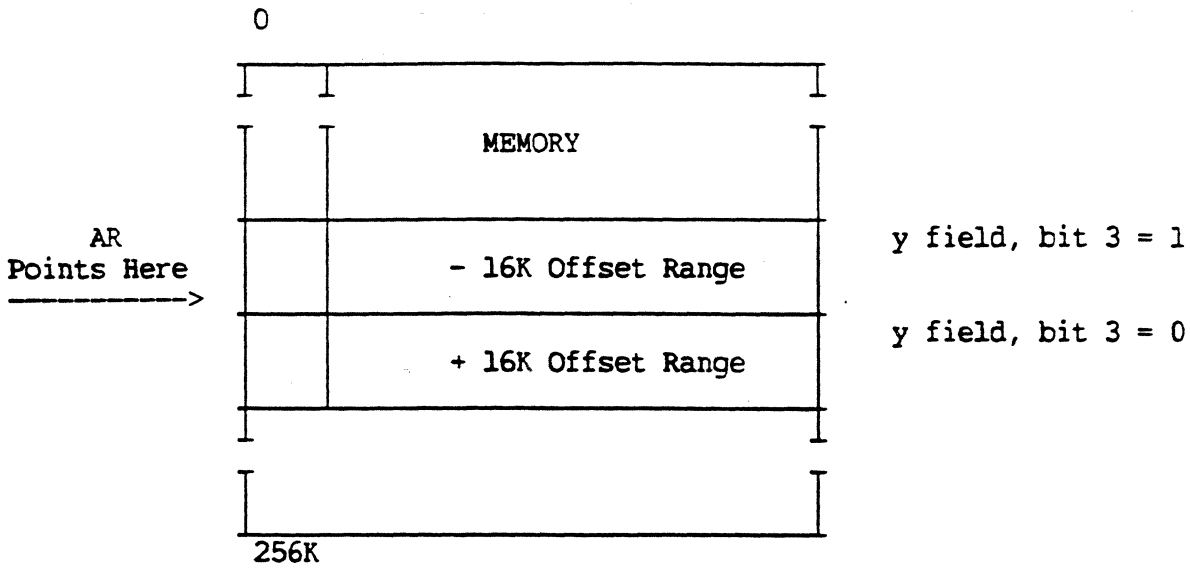


Figure 5-4. Address Preparation For Single-Word Instruction

When bit 29 = 0, the first step of the address modification procedure using the address register is omitted and the only address modification performed is that specified by the tag field.

When an address register is specified, extending bit 3 of the y field to form a two's complement signed number effectively designates bit 3 as a sign bit. This leaves 14 bits, 4 through 17, with which to designate an address offset. Thus an address offset with values between -2**14 and 2**14-1 can be specified. An address register, then, contains a complete 18-bit memory address which may be offset ± 16K by the partial address contained in the y field of the instruction, as shown below.



Coding Examples:

1. LDQ 4,N,2

   Effective Address = 4 + $C(AR2)_{0-17}$

2. LDQ -4,N,2

   Effective Address = -4 + bits 0-17 of C(AR2)

MULTIWORD ADDRESS MODIFICATION

The general format of a multiword instruction is shown in Figure 5-5.

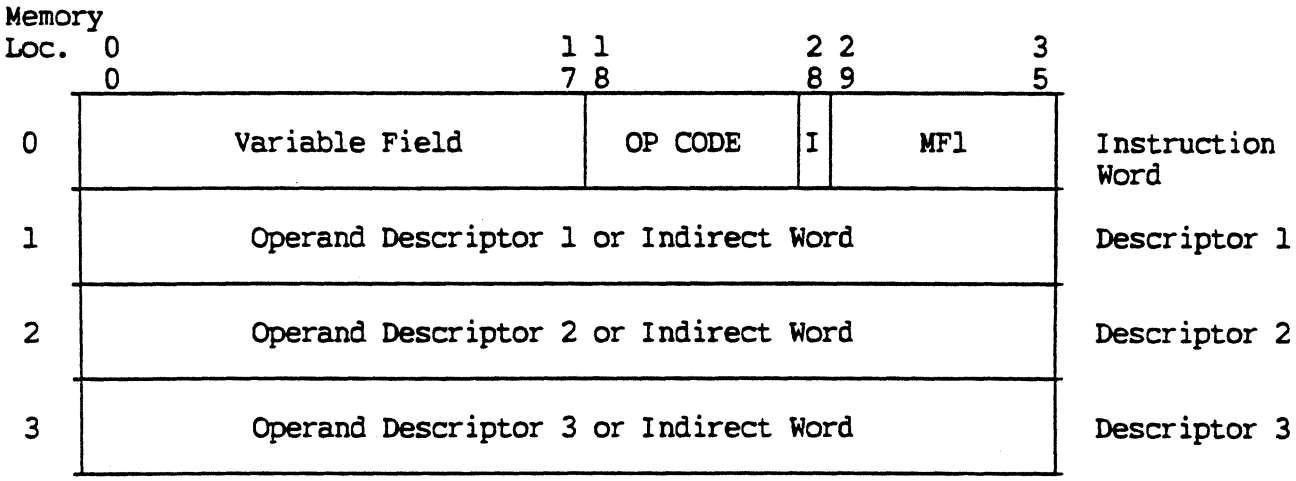


Figure 5-5. Multiword Instruction Format

where:

Variable Field - Contains additional information concerning the operation to be performed, depending on the particular instruction. When descriptors 2 and 3 are present, most instructions provide a corresponding MF2 (bits 11-17) and MF3 (bits 2-8) within the variable field to describe the address modification to be performed on these operands when present. Exceptions to this are the CMPCT, MVT, SCD, SCDR, SCM, SCMR, TCT, and TCTR instructions.

OP CODE - The 10-bit operation code field; octal representation consists of three octal digits corresponding to bit positions 18-26 and a 1 for bit position 27.

I - The program interrupt inhibit bit

MF1 - Modification field 1 (MF1) describes address modification that is to be performed for descriptor 1.

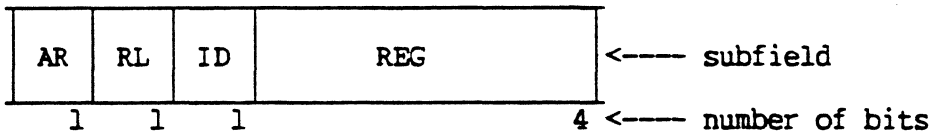
MULTIWORD MODIFICATION FIELD

Each modification field (MF) contained in a multiword instruction is a 7-bit field specifying address modification to be performed on the operand descriptors. The modification field is interpreted as follows:

2 3 4 5 through 8 <--- bits (MF3)

11 12 13 14 through 17 <--- bits (MF2)

29 30 31 32 through 35 <--- bits (MF1)



AR - Address Register Specifier

0- No address register used.

1- Bits 0-2 of the operand descriptor address field specify the address register to be used in computing the effective address of the operand. Bits 0 - 2 also specify the operand descriptor register that defines the segment containing the operand.

RL - Register or Length

0- Operand length is specified in the N field (bits 32-35) of the operand descriptor.

1- Length of operand is contained in the register that is specified by code in the N field (bits 32-35) of the operand descriptor, in the machine format of REG (the coding format is different).

ID - Indirect Operand Descriptor

0- The operand descriptor follows the instruction word in its sequential memory location.

1- The operand descriptor location contains an indirect word that points to the operand descriptor. Only one level of indirection is allowed.

REG - Address modification register selection for R-type modification of the operand descriptor address field. The REG codes are approximately the same as the single-word modifications. In addition, for indirect string length specification (RL = 1), the N field codes are similar to the REG field. A comparison of these codes is shown in Table 5-2.

Table 5-2. Register Codes

Octal Code	REG In MF (1)	REG In Indirect Word When ID = 1 (2)	Bits 32-35 Of N When RL = 1	td Field Of Tag
0000	None	None	IPR Fault	None
0001	AU	AU	AU	AU
0010	QU	QU	QU	QU
0011	DU	IPR Fault	IPR Fault	DU
0100	IC	IC	IPR Fault	IC
0101	A (3)	A (3)	A (3)	AL
0110	Q (3)	Q (3)	Q (3)	QL
0111	IPR Fault	IPR Fault	IPR Fault	DL
1000	X0	X0	X0	X0
1001	X1	X1	X1	X1
1010	X2	X2	X2	X2
1011	X3	X3	X3	X3
1100	X4	X4	X4	X4
1101	X5	X5	X5	X5
1110	X6	X6	X6	X6
1111	X7	X7	X7	X7

(1) Register content is interpreted as a character or bit index. For an alphanumeric descriptor, this index is the number of 9-bit, 6-bit, or 4-bit characters, depending upon the data type specified in the descriptor. For a numeric descriptor, it is the number of 9-bit or 4-bit characters, also dependent upon the data type specified. For a bit descriptor, it is the number of bits.

(2) Register contents are interpreted as a word index.

(3) The A- and Q-registers provide for indexing by a number greater than 2**18-1. When the A or Q register is specified, the number of right-justified bits for indexing depends on the type of unit reference specified in the operand referring to the A- or Q-register, as follows:

18 bits for full-word (36-bit) operations

21 bits for 9-bit and 6-bit character operations

22 bits for 4-bit character operations

24 bits for bit operations

All addressing is modulo addressing. For example, when software desires to index backwards by N words, it indexes forward by 2**18-N words. This same method is also used in character and bit indexing.

Unit	No. of Units/Word	No. to Effectively Yield -N
Word	1	2**18 - N
9-bit	4	4 * 2**18 - N (2**20 - N)
4-bit	8	8 * 2**18 - N (2**21 - N)
6-bit	6	6 * 2**18 - N
1 bit	36	36 * 2**18 - N

For 1-bit and 6-bit, 4-bit, and 9-bit characters, A and Q can be respectively loaded with 36,DU; 6,DU; 8,DU; or 4,DU; and N can then be subtracted.

The index register designations may be specified by a symbol defined by the user to have a value in the octal range of 0, 1, ...,7 (or 10, 11,...,17 when the RL usage is in a descriptor that does not immediately follow the multiword instruction - an indirect descriptor).

Example:

```
1       8       16
XA      BOOL    17

        MLR     (0,1),(0,1)
        ADSC9   A,0,XA
        ADSC9   B,0,XA
```

is used to specify a move of the number of characters specified by the current value of index register 7.

Similarly,

```
1       8       16
        MLR     (0,1,1),(0,1)
        ARG     LA
        ADSC9   B,0,XA
          .
          .
LA      ADSC9   A,0,XA
```

provides for the sending address of the move to be specified indirectly in the word labeled LA.

As a precautionary measure, all index register symbols should be defined with octal values in the range 10, 11,...,17, since the assembler uses only the low-order 3 bits in all contexts except the indirect descriptor where the symbol cannot be identified from context as an index register designation.

The content of the IC is always interpreted as a word address when used in address modification. During the entire execution of a multiword instruction, the IC points to the instruction word. Thus, if IC address modification is involved with a descriptor word, the instruction word address is used.

Specifying DU or DL type address modification in the REG field of an indirect operand descriptor is illegal and causes an IPR fault.

DU address modification is legal for MF2 of the SCD, SCDR, SCM, and SCMR instructions; for all other instructions, an IPR fault occurs.

Operand Descriptors

The operand descriptors describe the data to be used in the operation and provide the basic address for obtaining the data from memory. A unique operand descriptor format is required for each of the three data types: bit string, alphanumeric, and numeric. The operand descriptor machine formats are as shown in Figures 5-6, 5-7, and 5-8.

BIT STRING OPERAND DESCRIPTOR

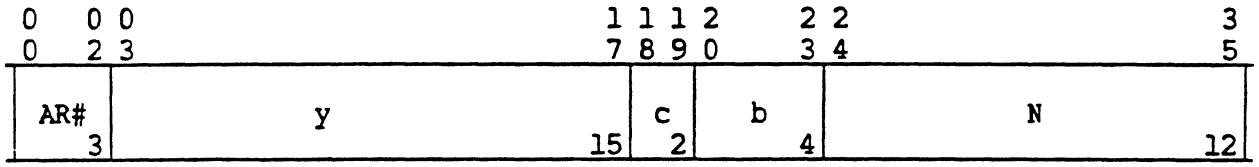


Figure 5-6. Bit String Operand Descriptor Format

Coding format for the bit string descriptor, BDSC, is:

BDSC - Bit descriptor

```
1       8       16
_____________________________

        BDSC    LOCSYM,N,c,b,AM
```

ALPHANUMERIC OPERAND DESCRIPTORS

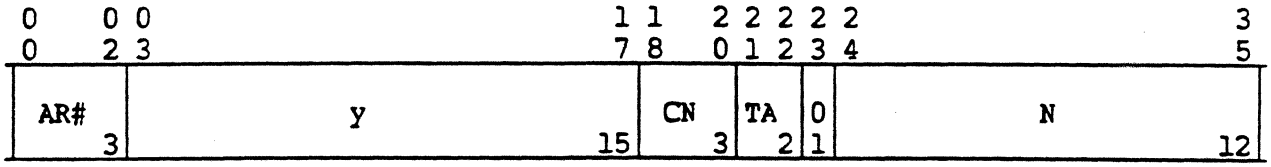


Figure 5-7. Alphanumeric Operand Descriptor Format

Coding formats for the alphanumeric descriptors are:

ADSC9 - ASCII alphanumeric descriptor

```
1       8       16
_____________________________

        ADSC9   LOCSYM,CN,N,AM
```

ADSC9 sets the TA field for 9-bit ASCII characters.

ADSC6 - BCI alphanumeric descriptor

```
1       8       16
_____________________________

        ADSC6   LOCSYM,CN,N,AM
```

ADSC6 sets the TA field for 6-bit BCI characters.

ADSC4 - Packed decimal alphanumeric descriptor

```
1       8       16
_____________________________

        ADSC4    LOCSYM,CN,N,AM
```

ADSC4 sets the TA field for 4-bit packed decimal characters.

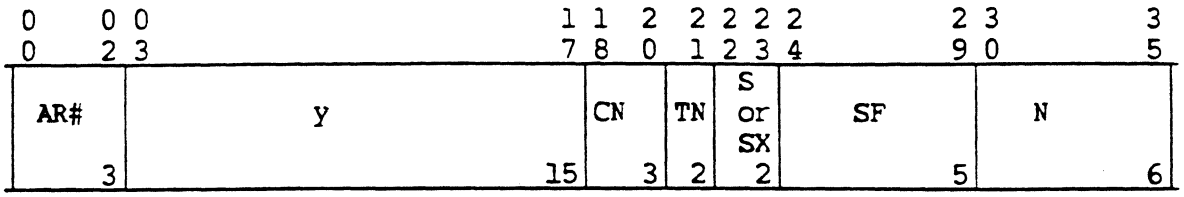


Figure 5-8. Numeric Operand Descriptor Format

Coding formats for the numeric descriptors are:

NDSC9 - ASCII numeric descriptor

```
1       8       16
_______________________________________

        NDSC9   LOCSYM,CN,N,S,SF,AM
```

NDSC9 sets the TN field for 9-bit ASCII characters.

NDSC4 - Packed decimal numeric descriptor

```
1       8       16
_______________________________________

        NDSC4   LOCSYM,CN,N,S,SF,AM
```

NDSC4 sets the TN field for 4-bit packed decimal characters.

The legend for the machine and coding formats of the descriptors is as follows:

y = starting data word address
18 bits (0-17) if address register not specified in MF; 15 bits (3-17) if address register specified in MF, with bit 3 extended;
15 bits (3-17) if address register specified in MF, with bit 3 extended (i.e., if bit 3 is zero, bits 0-2 are also considered to be zero; if bit 3 is 1, bits 0-2 are also considered to be 1s).

c = starting character position within a word of 9-bit characters.

Code	Char.
00	0
01	1
10	2
11	3

b = starting bit position within a 9-bit character.

Code	Bit	Code	Bit	
0000	0	0101	5	All other combinations of
0001	1	0110	6	these 4 bits are illegal
0010	2	0111	7	codes and will cause an IPR
0011	3	1000	8	fault.
0100	4			

N = either the number of characters or bits in the data string if RL = 0 in MF; or a 4-bit code (bits 32-35) that specifies a register (see Table 5-2) that contains the number of characters or bits if RL = 1 in MF

CN = starting character number within the data word specified by the starting data word address. Legal codes for the CN depends on the data type as shown below. Coding entry is by the character shown under CN Character.

Data Type	CN Character	Legal Codes	Illegal Codes
9-bit	0	000	001
	1	010	011
	2	100	101
	3	110	111
6-bit	0	000	110
	1	001	111
	2	010	
	3	011	
	4	100	
	5	101	
4-bit	0	000	
	1	001	
	2	010	
	3	011	
	4	100	
	5	101	
	6	110	
	7	111	

TA = a code that defines which type of alphanumeric character is used in the data

Code	Data Type
00	9-bit
01	6-bit
10	4-bit
11	Illegal - causes IPR fault

TN = a code that defines which type of numeric character is specified.

Code	Data Type
0	9-bit
1	4-bit

S = sign and decimal type (coding entry is by character)

Character	Code	Description
0	00	Floating-point, leading sign
1	01	Scaled fixed-point, leading sign
2	10	Scaled fixed-point, trailing sign
3	11	Scaled fixed-point, unsigned

SX = sign and scaling (for X operation codes)

If TN = 0 (unpacked data)
00 leading sign, overpunched, scaled
01 leading sign, separate, scaled
10 trailing sign, separate, scaled
11 trailing sign, overpunched, scaled

If TN = 1 (packed data)
00 leading sign, separate, floating-point
01 leading sign, separate, scaled
10 trailing sign, separate, scaled
11 no sign, scaled

SF = scaling factor

A two's complement binary number that gives the scale point position for scaled decimal numbers. The decimal point is assumed to be immediately to the right of the least-significant digit. The scaling factor is treated as a power of ten exponent where a positive number moves the scaled decimal point to the right and a negative number moves it to the left. Since the SF field is 6 bits, the largest number expressible is M x 10**31 and the smallest number is M x 10**-32, where M is the value of the data described by the numeric operand descriptor.

This field is ignored if S = 00.

Example: If data = 12345, S is not 00, and SF = -3, the value is 12.345.

AM = address register modification, used when AR = 1 in MF field

INDIRECT WORD

The basic instruction word containing the operation code is followed by either zero, two, or three descriptor words, with the number of descriptor words determined by the particular instruction. The descriptor words contain either the operand descriptor or an indirect word that points to the operand descriptor. When an indirect word points to the descriptor, the format of the indirect word is shown in Figure 5-9.

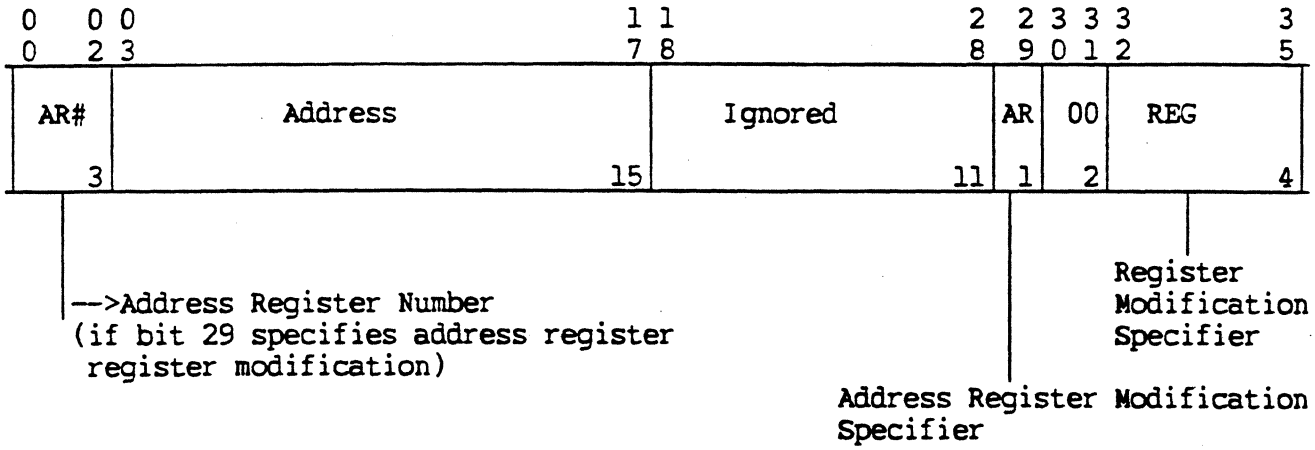


Figure 5-9. Indirect Word Format

The AR and REG fields are identical in function to the corresponding modification fields in the instruction word, except that the register content specified by the REG field of an indirect word is interpreted as word index only.

Indirect words can be generated with the ARG pseudo-operation as follows:

```
1       8       16
        ARG     LOCSYM,RM,AM
```

where:

LOCSYM - address

RM - register modification

AM - address register modification

for example:

```
1       8       16
        ARG     DFPRSS,,4
```

OPERAND DESCRIPTOR ADDRESS PREPARATION

A flowchart of the operations involved in operand descriptor address preparation is shown in Figure 5-10. The chart depicts the address preparation for operand descriptor 1 of a multiword instruction as described by modification field 1 (MF1). A similar type address preparation would be carried out for each operand descriptor as specified by its MF code. A detailed description of the flowchart follows:

1. The multiword instruction is obtained from memory.

2. The indirect (ID) bit of MF1 is queried to determine if the descriptor for operand 1 is present or is an indirect word.

3. This step is reached only if an indirect word was in the operand descriptor location. Address modification for the indirect word is now performed. If the AR bit of the indirect word is 1, address register modification step 4 is performed.

4. The y field of the indirect word is added to the contents of the specified address register.

5. A check is now made to determine if the REG field of the indirect word specifies that a register type modification be performed.

6. The indirect address as modified by the address register is now modified by the contents of the specified register, producing the effective address of the operand descriptor.

7. The operand descriptor is obtained from the location determined by the generated effective address in item 6.

8. Modification of the operand descriptor address begins. This step is reached directly from 2 if no indirection is involved. The AR bit of MF1 is checked to determine if address register modification is specified.

9. Address register modification is performed on the operand descriptor as described under "Address Modification with Address Registers" above. The character and bit positions of the specified address register are used in one of two ways depending upon the type of operand descriptor (i.e., whether the type is a bit string descriptor or a numeric or alphanumeric descriptor).

10. The REG field of MF1 is checked for a legal code. If DU is specified in the REG field of MF2 in one of the four multiword instructions (SCD, SCDR, SCM, or SCMR) for which DU is legal, the CN field is ignored and the character or characters are arranged within the 18 bits of the word address portion of the operand descriptor.

11. The count contained in the register specified by the REG field code is appropriately converted and added to the operand address.

12. The operand is retrieved from the calculated effective address location.

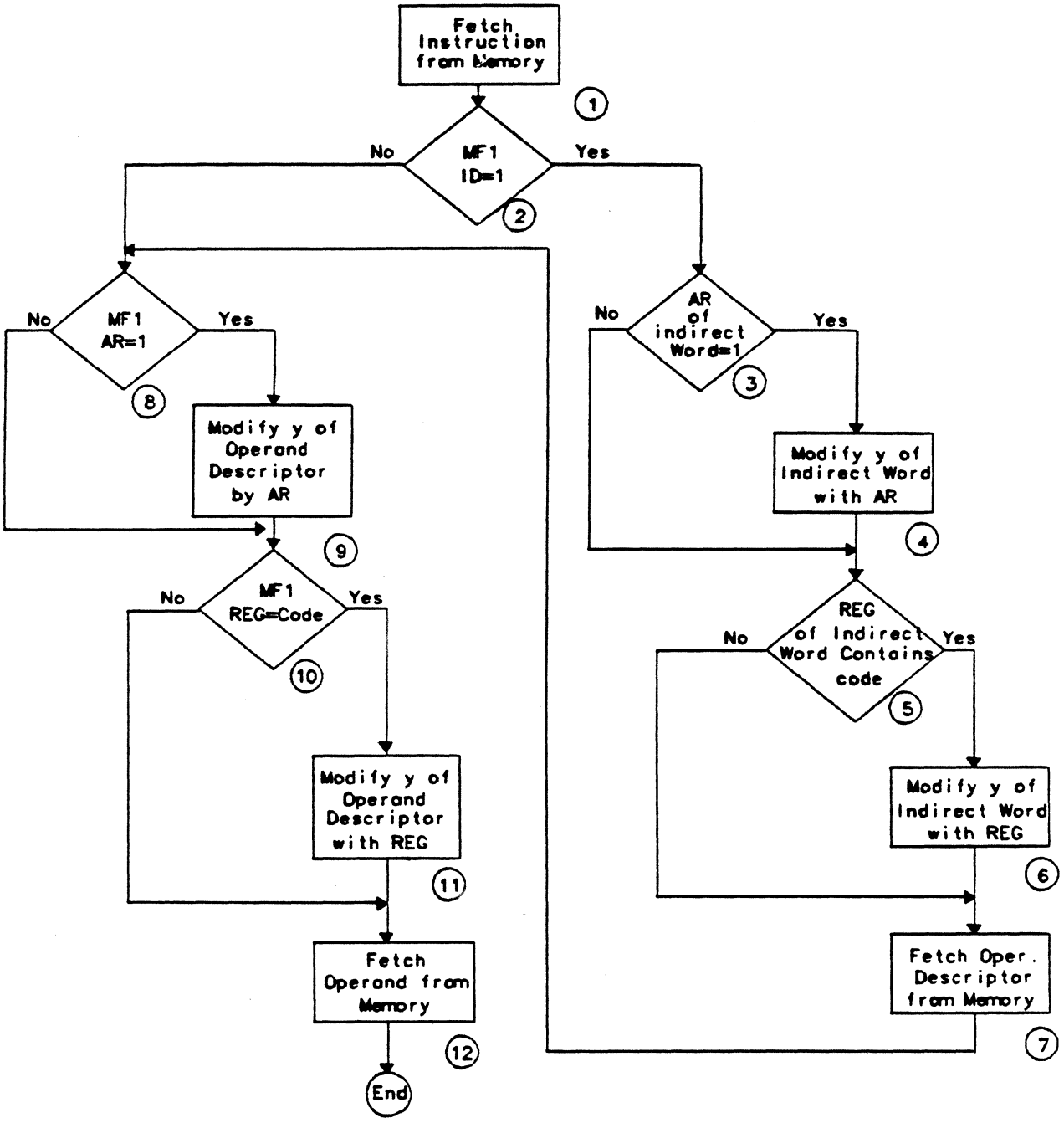
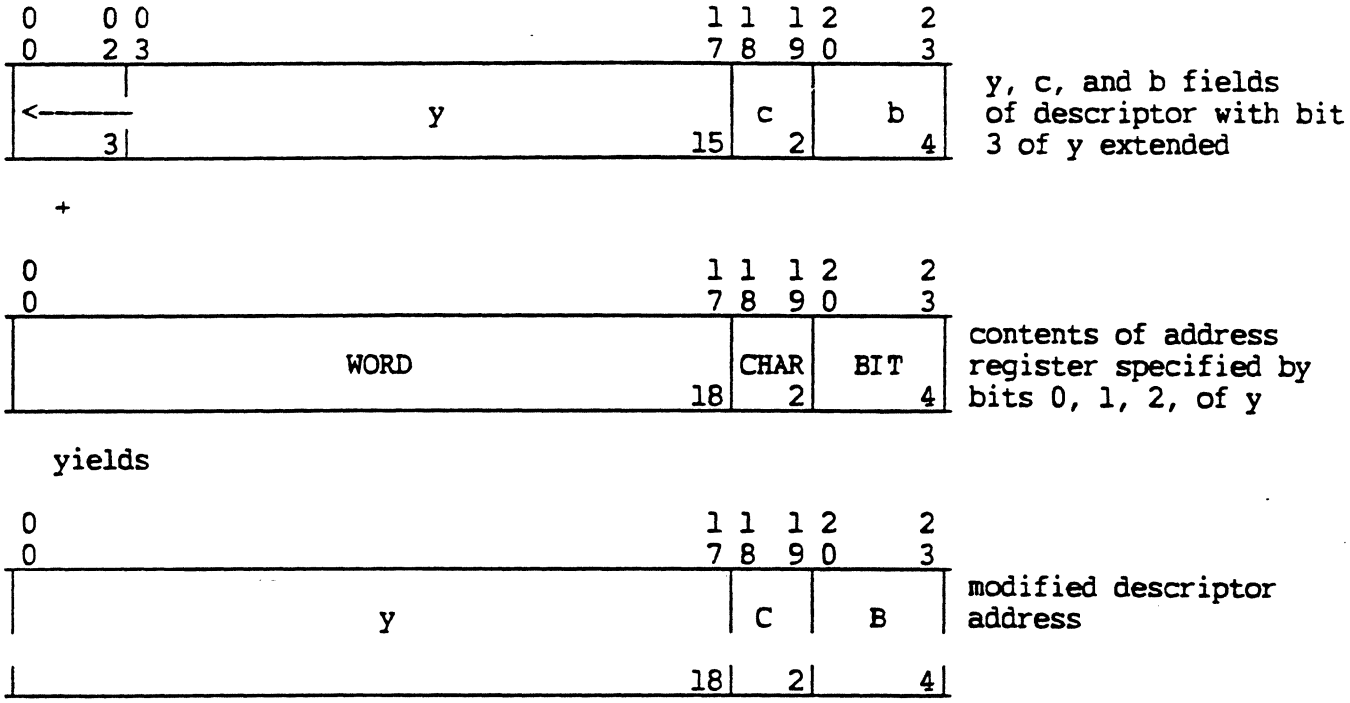


Figure 5-10. Flowchart For Operand Descriptor Address Preparation

Operand descriptor address preparation is illustrated in the flowchart of Figure 5-10. Procedures for the preparation of bit string addresses and alphanumeric/numeric addresses follow.

Bit String Address Preparation



where:

Y = WORD + y

C = CHAR + c

B = BIT + b

1. If (BIT + b) exceeds 8, a carry is generated to character position C and B = (BIT + b) -9:

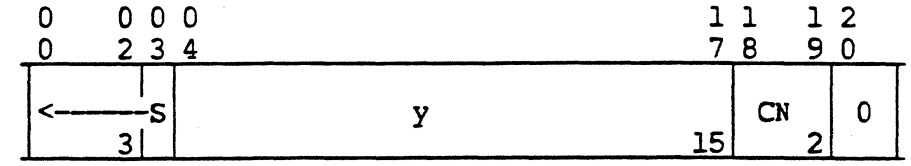
```
   BIT =  7
     b =  5
BIT + b = 12, carry 1 to C and B = 12 -9 = 3
```

2. If (CHAR + c + carry from B) exceeds 3, a carry is generated to the word address and C = (CHAR + c + carry from B) -4:

```
  CHAR =  2
     c =  3
 carry +  1
     =  6, carry 1 to word address and
           C = 6 -4 = 2
```

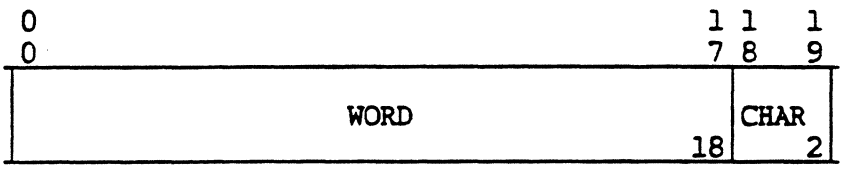
Alphanumeric/Numeric Address Preparation

First the data type designator (TA for alphanumeric, TN for numeric) is checked to determine the character size. If the data is in 9-bit characters, then the descriptor address and CN fields can be added directly to the address register contents as follows:



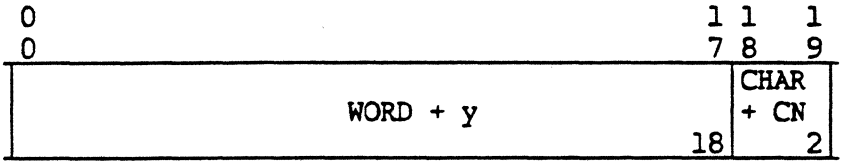
y and CN fields of the numeric or alphanumeric descriptor, bit 3 extended

+



contents of WORD & CHAR positions of address register designated by bits 0, 1, 2 of y

yields



modified character address

Bits 20-23 of the address register are ignored. CHAR is added to bits 18 and 19 of CN. Bit 20 of the descriptor is zero and is not used. If CHAR + CN is greater than 3, a carry is generated to WORD + y and CHAR + CN = (CHAR + CN) -4.

If the data is in 4- or 6-bit characters, the 9-bit character representation contained in the CHAR and BIT portions of the specified address register is interpreted to determine the corresponding 4- or 6-bit character position within the memory word. Translation to a 4-bit character location can be accomplished as follows:

C = 2 (CHAR) + [(BIT + 4)/9 truncated]

If CHAR = 3 and BIT = 7,

then C = 2(3) + 1 = 7

If CHAR = 3 and BIT = 4,

then C = 2(3) + 0 = 6

Translation to a 6-bit character location can be accomplished as follows:

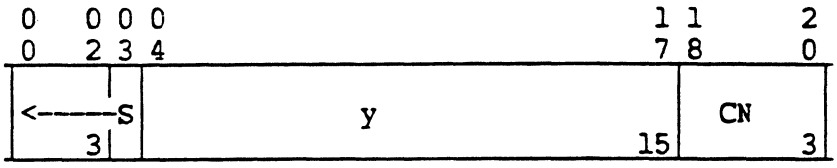
$$C = \frac{9\,(CHAR) + BIT}{6} \qquad \text{(truncated)}$$

If CHAR = 3 and BIT = 7,

$$\text{then } C = \frac{9\,(3) + 7}{6} = 5$$

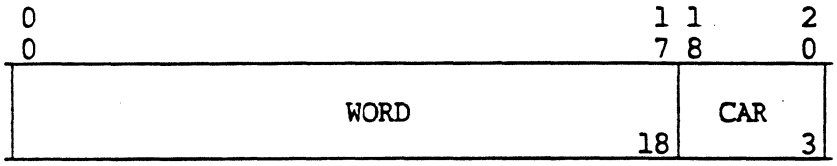
The remainder of 4 which represents the bit position within character position 5 is ignored. This means forcing the address register to point to the next lower character boundary.

The address modification can now take place.



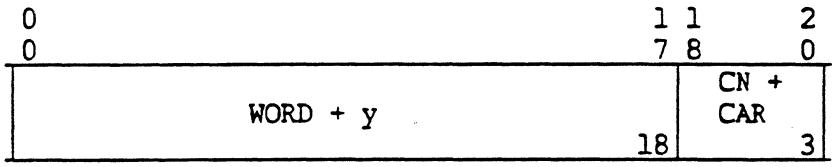
y and CN fields of the numeric or alphanumeric descriptor, bit 3 extended

+



contents of WORD position of address register indicated by bits 0,1,2 of y CAR is the char. location translated from CHAR and BIT of address register

yields

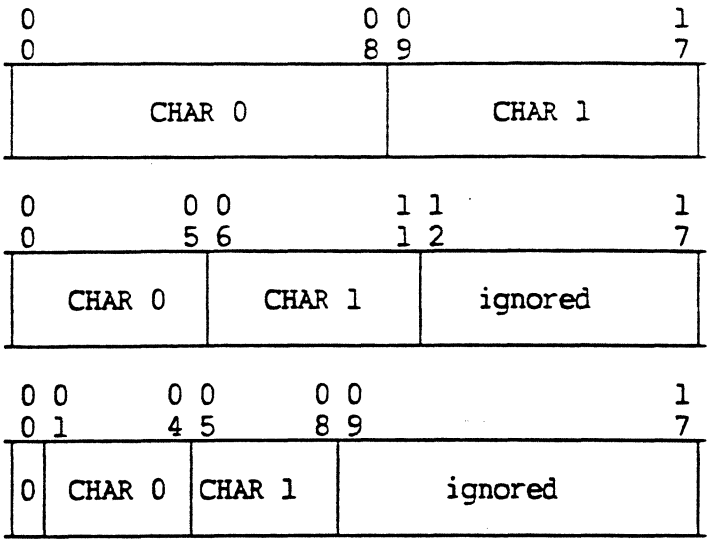


For 4-bit character mode, if CN + CAR is greater than 7, a carry is generated to WORD + y and CN + CAR = (CN + CAR) -8.

For 6-bit character mode, a carry is generated to WORD + y when CN + CAR is greater than 5 and CN + CAR = (CN + CAR) -6.

In the next step of operand descriptor address preparation, as indicated in item 10 in the flowchart of Figure 5-10, the REG field is checked for a legal code. If DU is specified in the REG field of MF2 in one of the four multiword instructions (SCD, SCDR, SCM, or SCMR) for which DU is legal, the CN field is ignored and the character or characters are arranged within the 18 bits of the word address portion of the operand descriptor as follows:

Operand descriptor word address field (y) Character type (TA)



Where only one character is involved (SCM, SCMR), only character 0 is used.

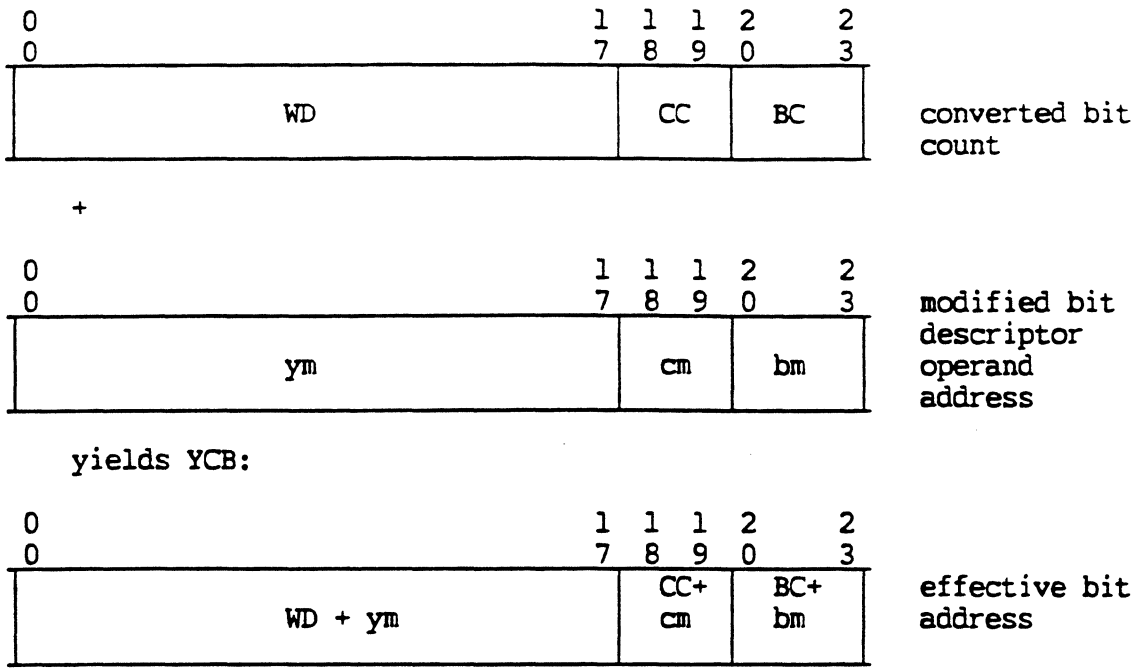
In step 11, in the flowchart of Figure 5-10, the count contained in the register specified by the REG field code is appropriately converted and added to the operand address. The count conversion required depends upon the type of data.

Bit Operations. The bit count contained in the register is effectively divided by 36 to give a word count (WD) with a bit remainder (BR). Dividing the bit remainder by 9 gives a character count with a bit remainder. Thus the original bit count (BC) is converted to a word count, 9-bit character count (CC) and bit remainder, and is in proper form to add to the bit operand address. An example of the effective conversion is shown below:

bit count from register/36 = WD and BR

BR/9 = CC and BC

Expressed as a 24-bit address modifier



Carries may occur from (BC + bm) to (CC + cm) and from (CC + cm) to (WD + ym).

There are two conditions to note in forming WD:

1. If WD is a small number (expressible in less than 18 bits), it is right-justified in the 18-bit word area with zero-fill in the most-significant bit positions. Thus bit counts are always positive; they are not two's complement and there are no bit extensions.

2. If the bit count comes from the A- or Q-registers, division by 36 may produce a WD greater than 2**18-1. In such a case, the result is interpreted modulo 2**18. For example, if the bit count is (2**24)-1:

   $\frac{(2**24)-1}{36}$ = 466,033 with BR = 27

   Thus, WD = 466,033 - 262,144 = 203,889

   And, BR/9 = 27/9 = 3 with 0 remainder

   So that, WD = 203,889

   CC = 3

   BC = 0

No errors occur; the operation is legal and the results are predictable.

Character Operations. The character count contained in the register is divided by 4, 6, or 8 (depending upon the data type), which gives a word count with a character remainder. The word and character counts are then appropriately arranged in 21 bits (18-word address and 3 for character position) and added to the modified descriptor operand address. The appropriate carries occur from the character positions to the word when the summed character counts exceed the number of characters in a 36-bit word. When the A- or Q-registers are specified, large counts can cause the result of the division to be greater than 2**18-1, which is interpreted modulo 2**18, the same as for bit addressing.

As the final step, (12 in flowchart in Figure 5-10) the calculated effective address location is used to retrieve the operand.

EXAMPLES:

```
1       8       16                  32
-------------------------------------------------------

 * OPERAND DESCRIPTOR EXAMPLES

        MLR     ,,020,1  move blanks to output record
        ADSC6   ,,0
        ADSC6   PRTOUT,0,55+80-31

        MLR     move columns 31-80
        ADSC6   RDWRK+5,0,80-31+1 to print columns 55-104
        ADSC6   PRTOUT+9,0,80-31+1

        LDX7    31-1,DU  ditto
        LDX6    55-1,DU
        LAR5    =V18/RDWRK
        LAR4    =V18/PRTOUT

        MLR     (1,,,7),(1,,,6)
        ADSC6   ,,80-31+1,5
        ADSC6   ,,80-31+1,4

        LAR5    =V18/RDWRK  ditto
        LAR4    =V18/PRTOUT
        LDX3    80-31+1,DU

        MLR     (1,1),(1,1)
        ADSC6   5,0,X3,5
        ADSC6   9,0,X3,4
```

ADDRESS GENERATION IN THE ES MODE

This subsection discusses the generation of effective addresses only insofar as it differs from the NS mode.

Instruction Address Field And Register Formats

The instruction field and register used in the generation of an effective address are interpreted as follows.

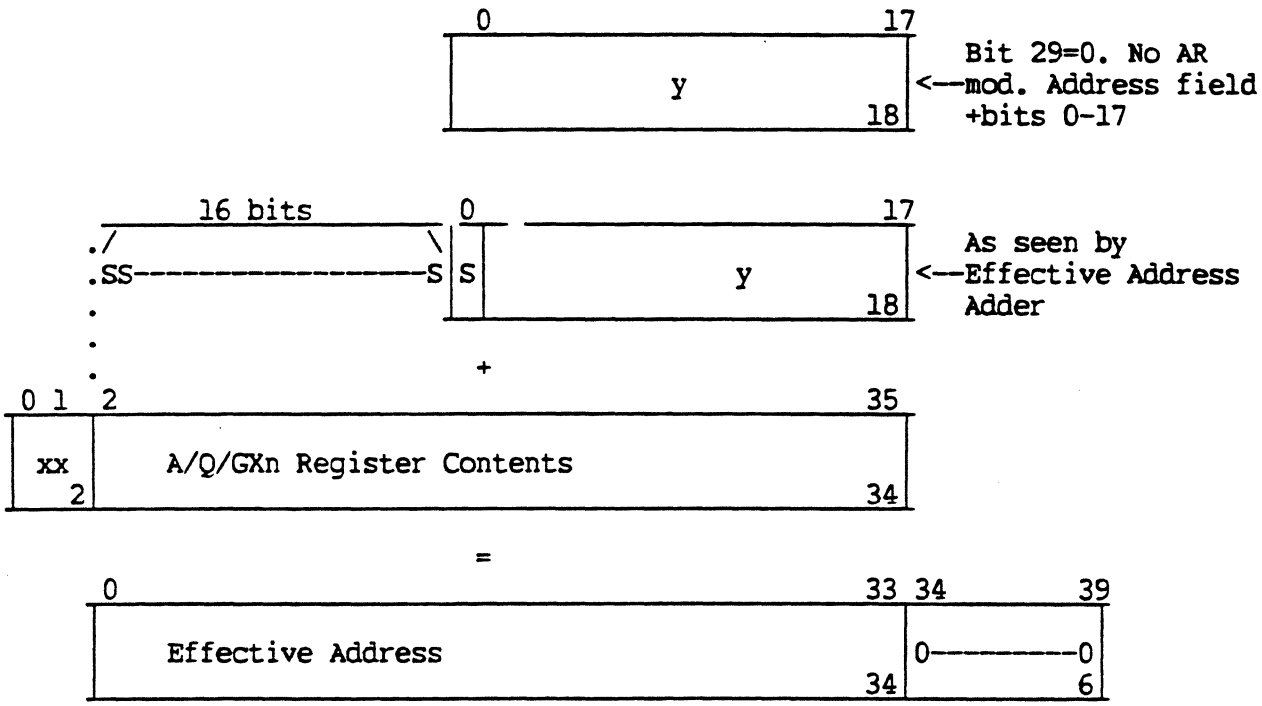
INSTRUCTION ADDRESS FIELD

Address preparation for all instructions starts with the address field of an instruction word (or the address field of an indirect word or data descriptor). All instruction words have the same format as shown in Figure 5-3.

Definitions for the individual fields of this format are found under "Single-Word Address Modification" in this section. The diagrams that follow start with only the address portion of an instruction field (bits 0 - 17).

Address Modification With No AR Indicated

When bit 29 = 0, no AR modification is specified. The sign (S) of (y) is extended 16 bits to the left, starting at bit 0 (rather than bit 3) as indicated below.



The y field of an instruction/indirect word/data descriptor is interpreted as given in the two's complement form. Bit 0 is assumed as a sign. To generate the effective address, bit 0 is extended 16 bits to the left. Bit 17 expresses the word location. The effective address (Y) field is +/- 128KW-1. When the A, Q, or a GXn register is used in the R modification of a basic instruction (single-word) or a vector instruction, bits 2 through 35 are treated as word address and bits 0 and 1 are ignored. An AL/QL specification in the tag field modification specifies 36-bit A/Q registers. An AU/QU specification results in an IPR fault. Address modification specified by the tag field is performed resulting in the effective address.

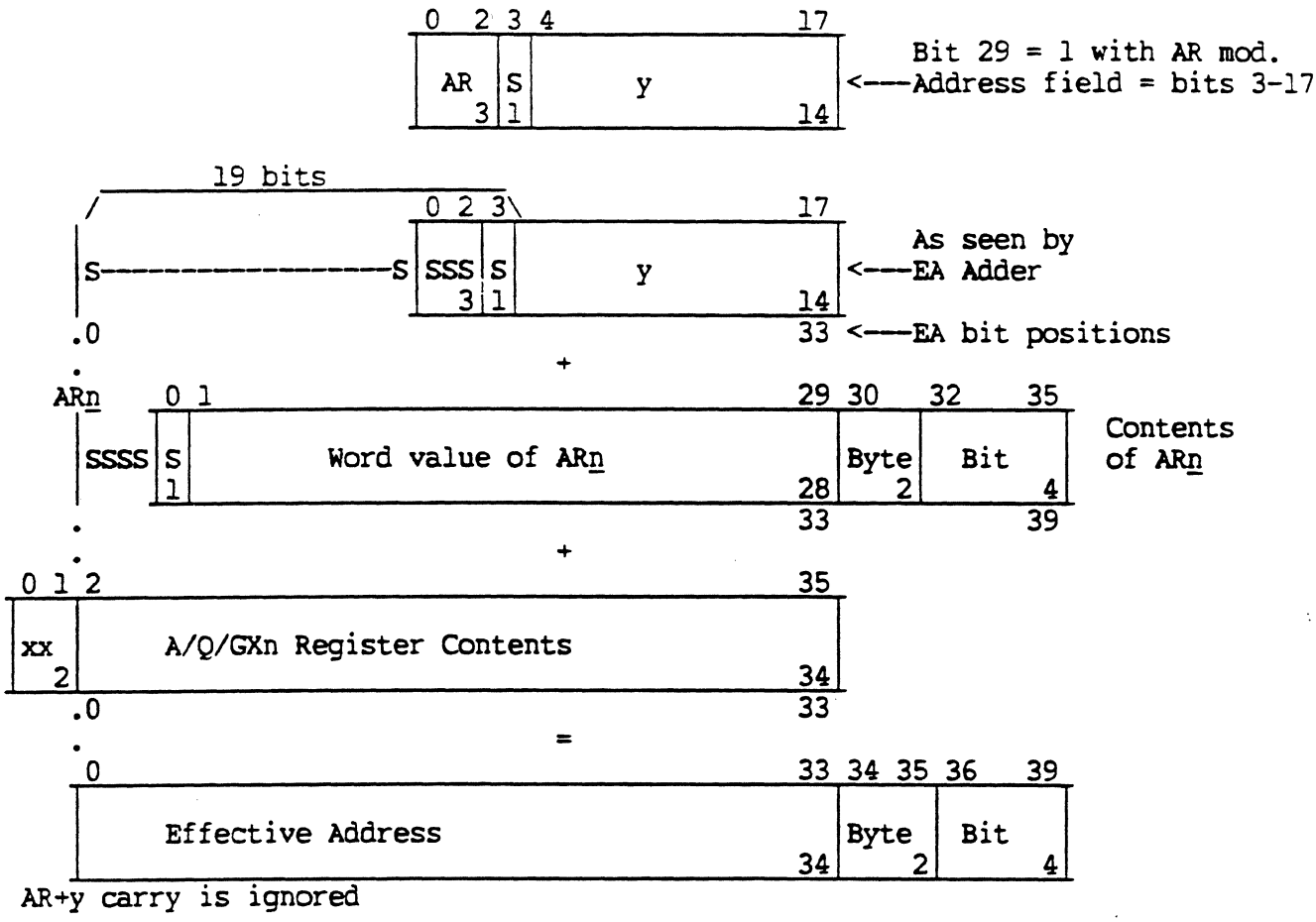
EXAMPLES:

	1	8	16	Effective Address
(1)		EAX4	1	
		LDA	B,4	Y = B+1
	B			
(2)		LDA	=4,DL	
		LDQ	C,AL	Y = C+4
	C			
(3)		EAQ	3	
		STA	B,QL	Y = B+3
	B			

With no AR modification specified, address modification is processed in the same way as address modification in NS mode, with the exception of the AU/QU modification.

Address Modification With AR Indicated

Address register modification is performed when instruction word bit 29 = 1 or when the AR bit of a multiword instruction's MF field is 1.



Bits 3 through 17 of an instruction/indirect word/data descriptor are interpreted as given in a two's complement form. Bit 3 is assumed as a sign. Thus, the range of Y is +/- 16KW-1. To generate an effective address, bit 3 is extended 19 bits to the left. Bit 17 expresses the word location.

The address register (ARn) is extended to 36 bits as indicated in the previous format. ARn is interpreted as given in a two's complement form with bit 0 as a sign bit. In effective address generation, bit 0 is extended 4 bits to the left. Bits 0 through 29 are interpreted as a word address, bits 30 and 31 as a byte address within the word, and bits 32 through 35 as a bit address within the byte. If BIT > 8, BIT = 8 is assumed.

Every specification of an index register (Xn) is interpreted as specifying a 36-bit GXn. An AL/QL specification in the register modification (R modification, REG modification, N when RL = 1) specifies the 36-bit A/Q registers. Any AU/QU specification results in an IPR fault. When GXn is used in the R modification of a basic instruction (single-word instruction), bits 2 through 35 are treated as a word address.

When GX/A/Q is used in the REG modification of a multiword instruction, bits 0 through 35 are treated as the number of characters specified by the bit number in the data descriptor.

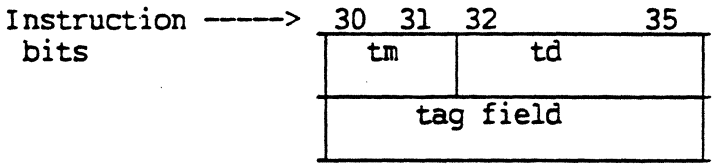
Because effective address generation in ES mode involves sign extension, an instruction such as LDA LOCSYM causes a Bound fault if LOCSYM is greater than or equal to 128K words, regardless of the instruction segment bound.

EXAMPLES:

	1	8	16	Effective Address
(1)	EAX2	2	(X2=2)	
	AWDX	1,2,3	AR3 = 3|0|0	
	STZ	B,2,3	Y = B+5	
(2)	EAX3	1	(X3=1)	
	AWDX	2,3,1	AR1=3|0|0	
	LDA	B,,1	Y=B+3	
(3)	AWDX	4,,3	AR3=4|0|0	
	EAX4	B	X4= address of B	
	STA	1,4,3	Y=B+5	
(4)	EAX4	B		
	AWDX	0,4,2	AR2= address of B	
	STA	2,,2	Y=B+2	

Tag Field Modification

In a basic instruction (single-word instruction), a tag field modification is performed after the AR modification. The tag field format follows:



The interpretation of a tag field and the accompanying modification method are the same as in the NS mode except that the address modification by the register A/Q/GXn/IC is altered as illustrated below. This applies to generation of the following:

an operand address in R modification (tm = 00)

an indirect word address in RI modification (tm = 01)

an operand address in IR modification (tm = 10)

The following should be noted with A/Q/GXn modification:

1. EA (effective address) may be represented as Y.
2. The GXn specification code is identical to the Xn specification code.
3. The A/Q specification code is identical to the AL/QL specification code.
4. An AU/QU specification results in an IPR fault.

EXAMPLES:

R-Type

```
                                                        Effective
     1          8          16                          Address
(1)             EAX2       1
                LDA        B,2                         Y=B+1

(2)             LDQ        =3,DL
                LDA        B,QL                        Y=B+3
```

RI-Type

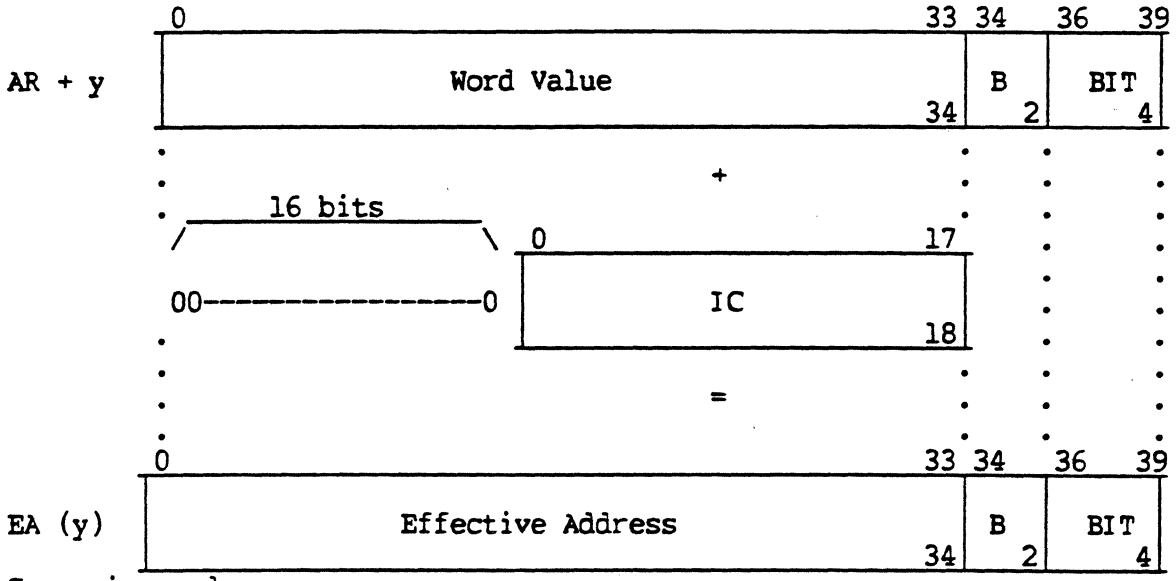
```
          Z     ARG        B
                ARG        A,2*
                ORG        A+5
          A     ARG        B,5*
                ORG        B+1
                 .          .
                 .          .
(1)             EAX2       1
                LDA        Z,2*                        Y=B+1

(2)             EAX1       0
                STQ        Z,1*                        Y=B

(3)             EAX2       3
                STA        Z,2*                        Y=A+5
```

IR-Type

```
                                                        Effective
     1          8          16                          Address
(1)             LDQ        3,DL
                LDA        Z,*QL                       Y=B+4
                 .
                 .
          Z     ORG        B+1

(2)             EAX4       3
                EAX5       6
                STA        C,*4                        Y=Z+9
                 .

          C     ARG        B,*5
          B     ORG        Z+3
```

```
                                                          Effective
      1       8       16                                  Address
(3)           EAX1    3
              LDQ     X,*1                                Y=B+8
                .
                .
      X       ORG     B+5
```

When IC modification is specified, effective address development is as follows:



Carry ignored

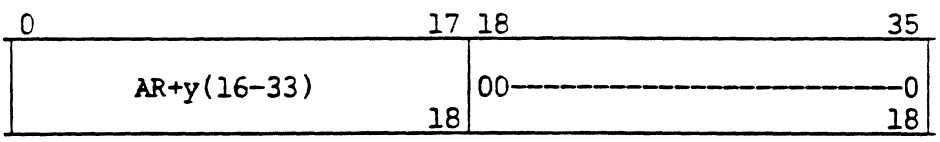
The contents of the instruction counter extended on the left with 16 bits zero-filled is added to the contents of AR + y.

EXAMPLES:

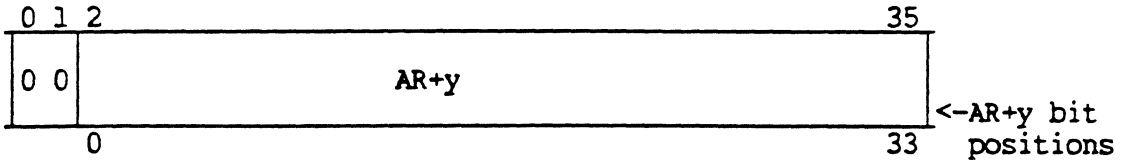
```
                                                          Effective
      1       8       16                                  Address
      IC added to AR

(1)           AWDX    0,QL,3
              AWDX    1,QL,4
              AWDX    2,QL,2
              SZN     TEST
              TZE     TEST                                Y=IC+AR3
              TMI     0,$,4                               Y=IC+AR4
              TRA     0,$,2                               Y=IC+AR2
                .
                .
(2)           AWDX    1,AL,2
              LDA     2,$,2                               Y=IC+AR2
```

When DU/DL modification is specified, effective address modification interprets the operand data as follows:

For DU



For DL



EXAMPLES:

```
                                                        Effective
1       8       16                                      Address
```

Compare GX1 to AR3

```
(1)         EAX1    A                           GX1 = address of A
            CMPX    1,DL,3
```

Load AU with contents of AR2

```
(2)         EAX3    B
            AWDX    0,3,2                       AR2=address of B
            LDA     0,DU,2
```

Operand Descriptor Modification

When REG modification is specified in the MF field of a multiword instruction, it is processed as follows.

When A/Q/GXn is specified

The 36 bits of A/Q/GXn are used as the character number which is the character address.

An AU/QU specification results in an IPR fault.

EXAMPLES:

```
1       8       16                                  Effective
                                                    Address
```

(1) This moves the string "SOURCE" to the first six characters of TO. The contents of X3 act as an offset into the source text.

```
        LDX3    =11,DL
          .
          .
        MLR     (,,,3),,040
        ADSC9   FROM,1,6
        ADSC9   TO,0,6
          .
          .
FROM    ASCII   9,THIS IS THE SOURCE TEXT
TO      BSS     2
```

(2) The string "LE " is moved to XB, starting at the third character of XB. The Q register can be used in the same way.

```
        LDA     =4,DL
          .
          .
        MLR     (,,,A),(,,,,),040
        ADSC9   XA,0,3
        ADSC9   XB,2,3
          .
          .
XA      ASCII   5,SAMPLE TEXT TO MOVE
XB      BSS     3
```

When IC is specified in the REG modification, it is treated as an 18-bit word address.

EXAMPLES:

```
1       8       16                                  Effective
                                                    Address
```

The string "HIS IS" is moved to Y, beginning with the first character.

```
        EAX3    Y
        AWDX    0,3,2                       AR2=address of Y
          .
          .
        MLR     (,,,IC),(1,,,),040
        ADSC9   3,1,6
        ADSC9   0,0,6,2
X       ASCII   4,THIS IS THE TEXT
Y       BSS     2
```

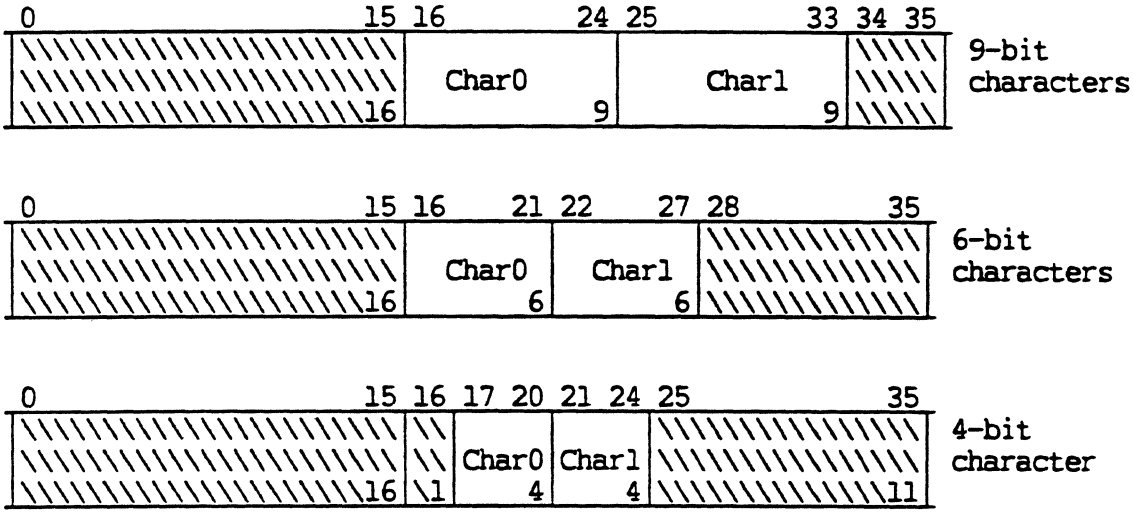
When DU/DL is specified

DL - An IPR fault occurs.

DU - Permitted only in the SCD, SCDR, SCM, and SCMR instructions.

The effective address (EA(y)) generated by the operand descriptor is treated as follows.

Bits 16 through 33 of the effective address (EA(y)) are interpreted as character data according to its data format (TA or TN field of the descriptor).



For the SCM or SCMR instructions, only CHAR0 indicated in the diagrams is used. The shaded portions are ignored during effective address generation.

ADDRESS DEVELOPMENT

Virtual Memory Addressing

Virtual memory provides the processor with a virtual memory capability, consisting of a directly addressable virtual space of 2**43 bytes and the mechanisms for translating this virtual memory address to a real memory address. Memory paging is an integral part of the translation process for this conversion. An absolute addressing mode that allows bypassing the translation process is also provided. When the processor is operating in the absolute addressing mode, the virtual memory address and the real memory address are the same.

To provide for virtual memory management, assignment, and control, the 2**43 byte virtual memory space is divided into smaller units called working spaces, and segments.

- o Working Spaces (WS)

  The 2**43 bytes of virtual memory space are divided into 512 2**34-byte working spaces (WS). WS numbers used to generate a particular virtual memory address are obtained from one of the eight WS registers or a segment descriptor register (DRn). The WS number is represented in a segment descriptor register either by the content of a specified WSR or by a 9-bit WSN field.

- o Segments

  A segment is part of a working space and may be as small as one byte or as large as 2**32 bytes for an extended segment. (GCOS disallows the use of contiguous working spaces for a single segment.) Thus, unlike the fixed size of a WS, a segment size is variable. Segments are described by a 72-bit descriptor.

  When a virtual address is generated, the descriptor (more commonly referred to as the segment descriptor) is contained in a register such as the instruction segment register (ISR). For operands, the descriptor may be contained in other segment descriptor registers. The area of virtual memory constituting a segment is "framed" by the segment descriptor by defining a base value relative to the base of the WS and a bound value relative to the base of the segment.

Virtual memory affects memory address development for both instructions and operands in Privileged Master, Master and Slave modes of operation.

OPERAND ADDRESS PROCEDURE

In the first phase of address generation, the effective address (EA) of the operand is generated as previously described for effective address generation. The EA is that address obtained after all register modification and indirect processing has taken place. It is an 18-bit word, 20-bit byte, or 24-bit bit address in the NS mode, and a 30-bit word, 32-bit byte, or 36-bit bit address in the ES mode.

After the EA has been formed, the processor hardware forms the virtual memory address of the operand using the base, bound, and WS values from 1 of 9 segment descriptors. If bit 29 of the instruction for which the operand address is being prepared is zero, then the operand resides in the instruction segment and the base, bound, and WS from the instruction segment register (ISR) are used to form the virtual address of the operand; if bit 29 of the instruction is 1, then descriptor register n (DRn) specified by bits 0, 1, and 2 of the address field of the instruction is used. Note that specifying DRn constitutes specifying ARn and vice versa.

When indirect EA development is involved, the following rules apply:

a. When DRn and ARn are involved (instruction bit 29 = 1), ARn is applied only to the first address in a chain of indirect addresses. However, the base, bound, and WS from DRn are applied to each memory reference in the indirect chain.

b. When no DRn/ARn is specified (instruction bit 29 = 0), the base, bound, and WS of the ISR are applied to each memory reference in an indirect chain.

c. A word in an indirect chain cannot specify a DRn.

d. An XEC or XED[1] instruction does not constitute an indirect chain; therefore, the instruction executed may specify a different DRn than the XEC/XED instruction, or no DRn. If the instruction executed by the XEC/XED does not specify a DRn, the base, bound, and WS from the ISR are used to form the virtual address of the operand.

INSTRUCTION ADDRESS PROCEDURE

Virtual addresses for instructions are always formed using the value in the instruction counter (IC) and the base, bound, and WS from the ISR.

Virtual Address Generation For NS Mode

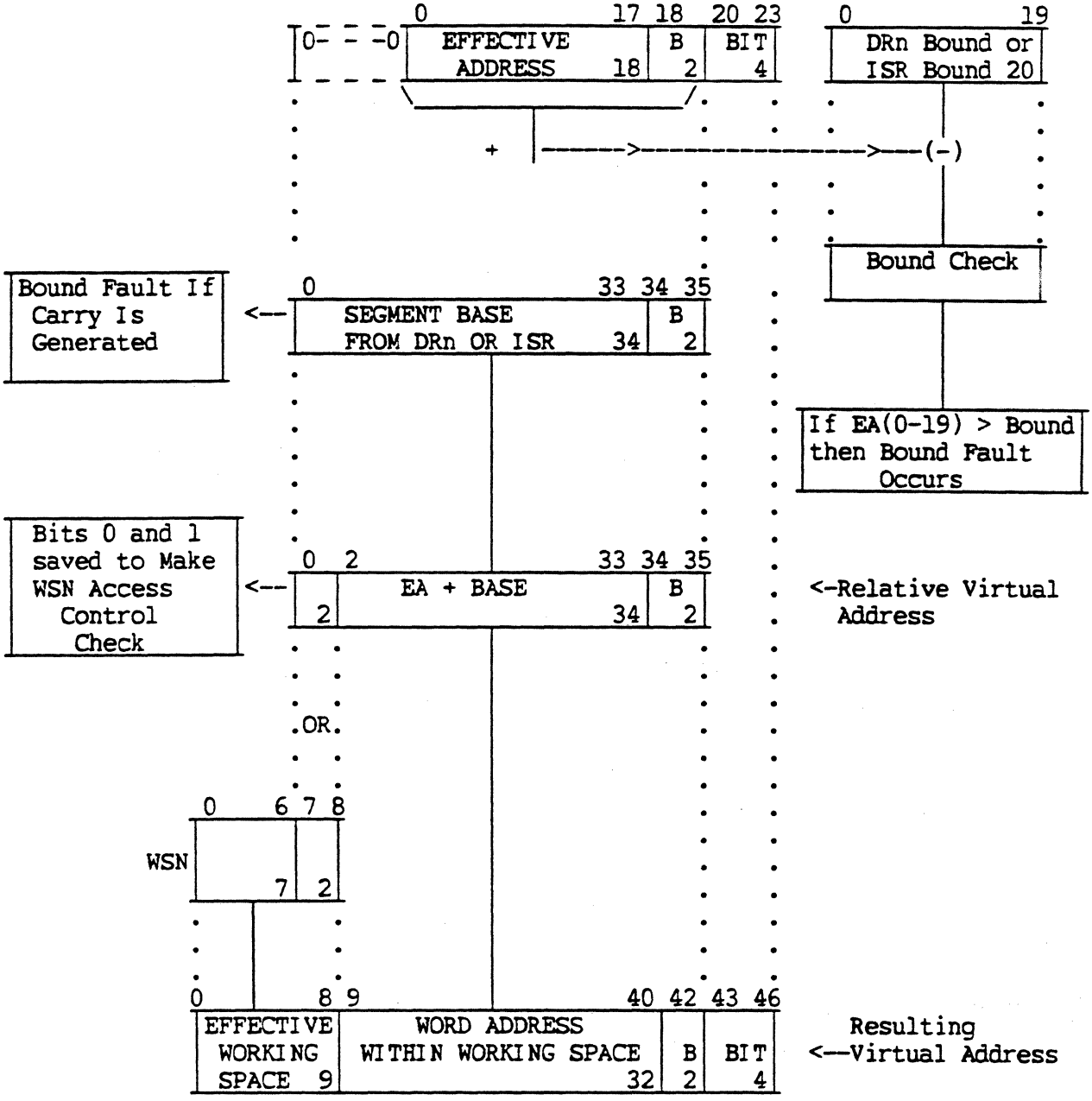
For all memory accesses, a virtual address must be generated. The mechanics of generating the virtual memory address depend on whether the involved segment descriptor is a standard descriptor or a super descriptor. Thus, the procedure described below for generating the operand virtual address with a standard descriptor also applies to virtual address generation for accessing the instruction, argument, parameter, and linkage segments (the registers holding the descriptors that define these segments may only contain standard descriptors).

---

1. XED executes in NS mode only.

STANDARD DESCRIPTOR NS MODE

The method of forming an operand virtual address with a standard descriptor is shown in Figure 5-11. If instruction bit 29=0, the ISR is used; if bit 29=1, then DRn is used.



where: B - page byte
WSN - working space number

Figure 5-11. Virtual Address Generation Using Standard Descriptor (NS Mode)

The bound check is applied to the effective address at the byte level. The bound check is shown for byte or bit instructions; the checks for single-word or multiword instructions require inclusion of the base in upper- and lower-bound algorithms.

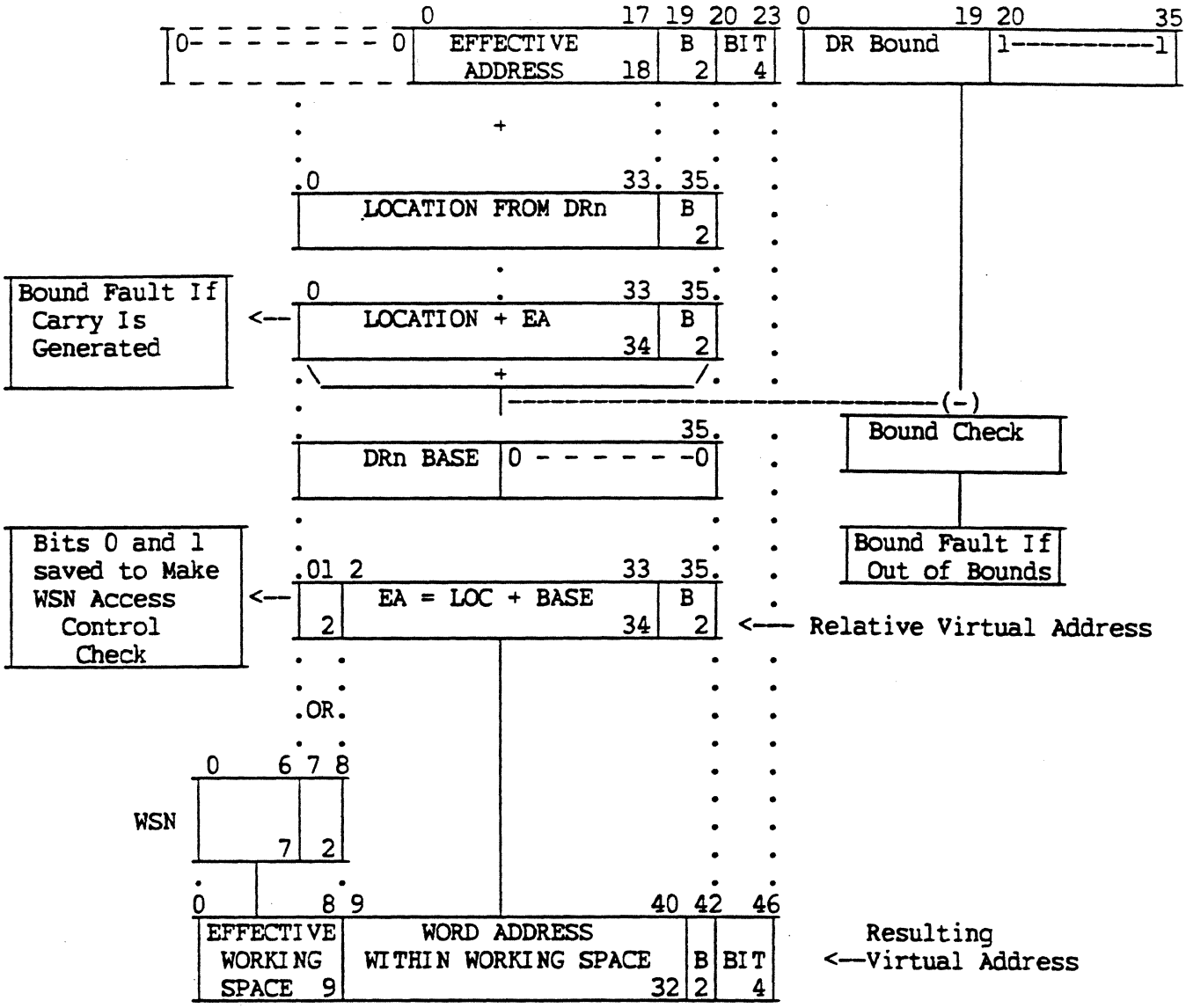
If a carry is generated when the EA is added to the base, an out-of-bound situation exists, resulting in a Bound fault.

The effective WSN is formed by ORing the low-order two bits of the working space number with bits 0 and 1 of the sum of EA + BASE.

The bit address from the EA becomes the bit address of the virtual address.

SUPER DESCRIPTOR NS MODE

The method of forming an operand virtual address with a super descriptor is shown in Figure 5-12.

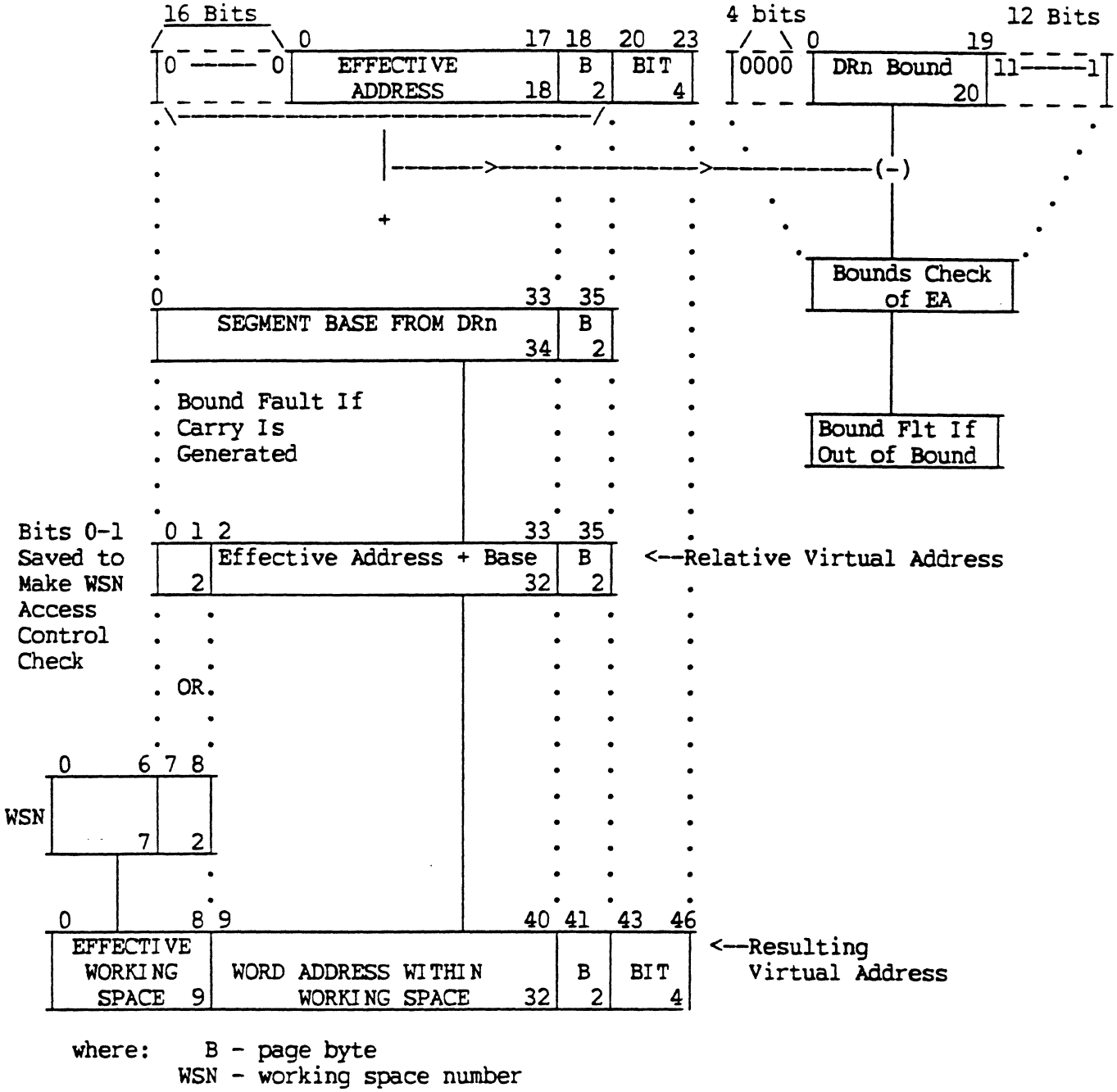


where: B - page byte
WSN - working space number

Figure 5-12. Virtual Address Generation Using Super Descriptor (NS Mode)

EXTENDED SEGMENT DESCRIPTOR NS MODE

The method of forming an operand virtual address with an extended segment descriptor is shown in Figure 5-13. It is the same as that using a standard segment descriptor except in the bound check.



where: B - page byte
WSN - working space number

Figure 5-13. Virtual Address Generation Using Extended Segment Descriptor (NS Mode)

Virtual Address Generation For ES Mode

In the ES mode, a 36-bit effective address is added to a segment descriptor to generate a virtual address. The method used for generation of virtual addresses differs depending upon whether the related segment descriptor is a standard segment descriptor or an extended segment descriptor. Super descriptors must not be used for address generation in ES mode as any attempt to do so results in an IPR fault.

STANDARD DESCRIPTOR ES MODE

The method of forming an operand virtual address with a standard descriptor in ES mode is shown in Figure 5-14. If instruction bit 29=0, the ISR is used; if bit 29=1, then DRn is used.

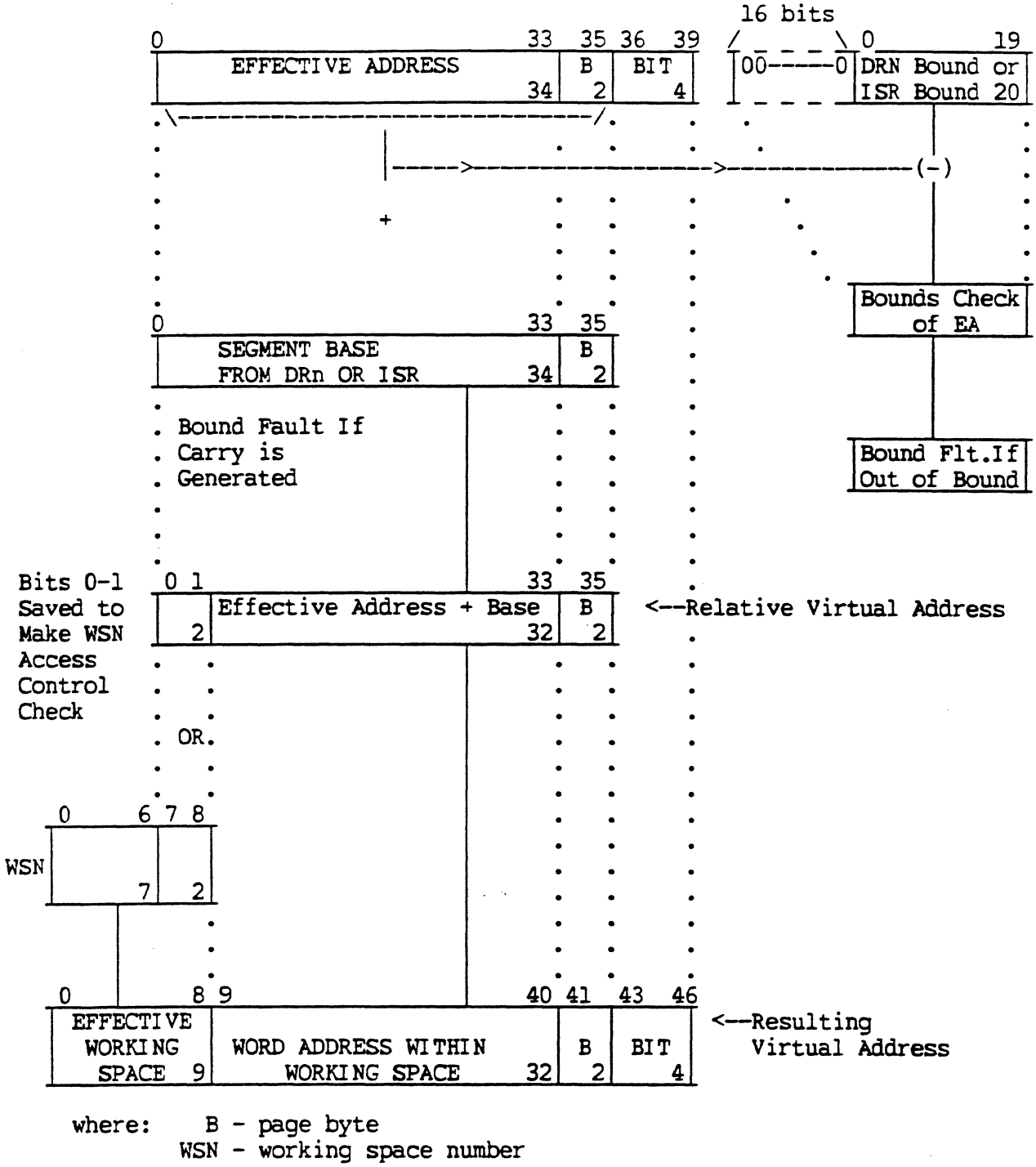
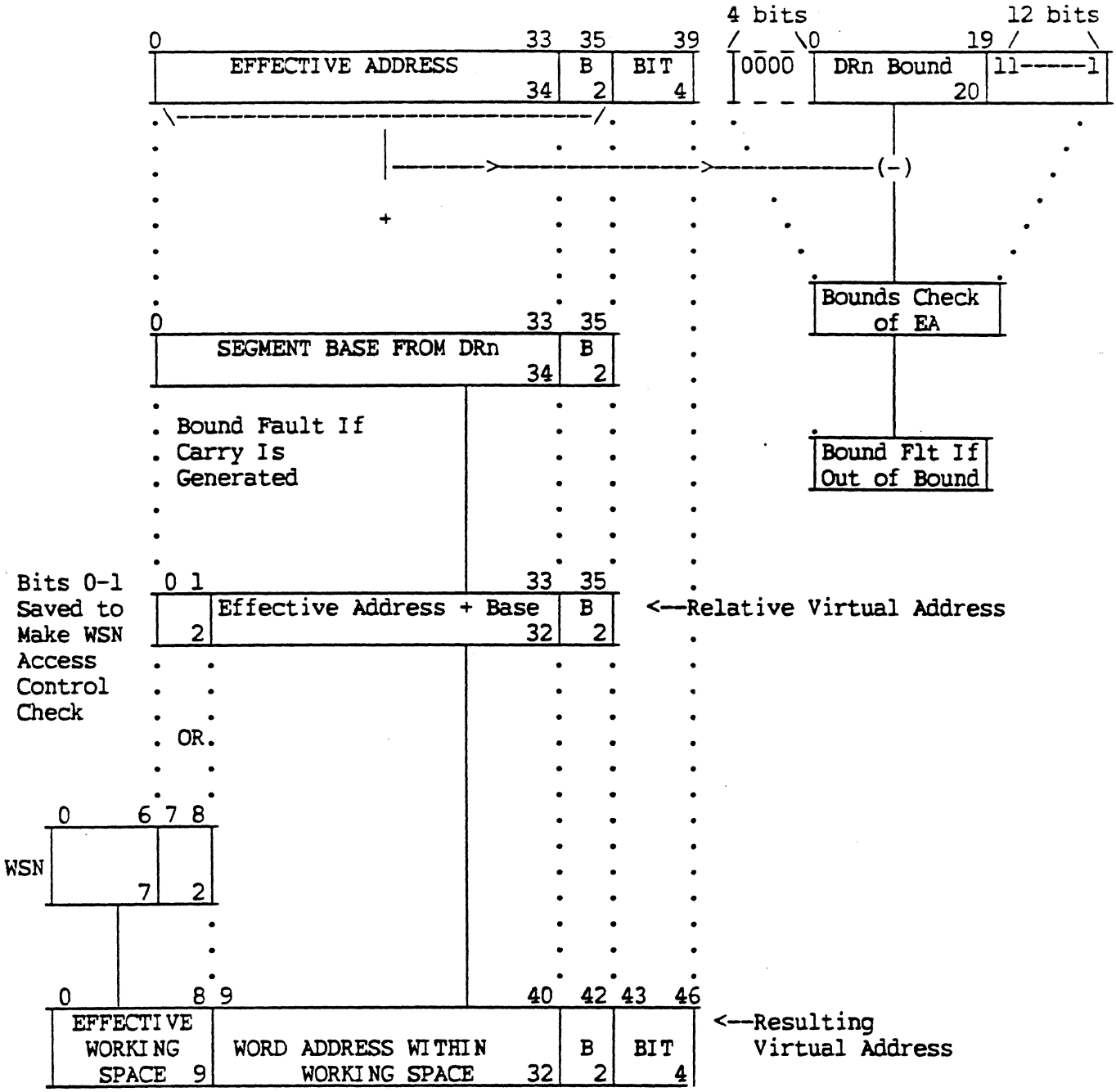


Figure 5-14. Virtual Address Generation Using Standard Descriptor (ES Mode)

EXTENDED SEGMENT DESCRIPTOR ES MODE

The method of forming an operand virtual address with an extended segment descriptor (T = 12) is shown in Figure 5-15. It is the same as that using a standard segment descriptor except in the bound check.



where: B - page byte
WSN - working space number

Figure 5-15. Virtual Address Generation Using Extended Segment Descriptor (ES Mode)

Absolute Addressing Mode

Virtual memory provides an absolute addressing mode. When the processor uses the absolute addressing mode, a virtual address is generated. However, the virtual address is not mapped to a real address; it is used as the real address with a maximum size limitation of 2**28 words (256 megabytes).

The processor utilizes the absolute addressing mode when the referenced working space register or descriptor (with working space number) contains WSN = 0. In these cases, the upper two bits of the segment base are not OR'ed with the working space number. The absolute address mode is fully set by the direct value of the WSN.

To use the absolute addresing mode, the CPU must be in Privileged Master Mode. If these conditions are not satisfied, a Command fault occurs when an attempt is made to reference working space zero. The housekeeping bit is assumed ON when working space zero is referenced.

When the processor is in the absolute addressing mode, address preparation proceeds as in normal virtual address development. (Refer to Figure 5-16.)

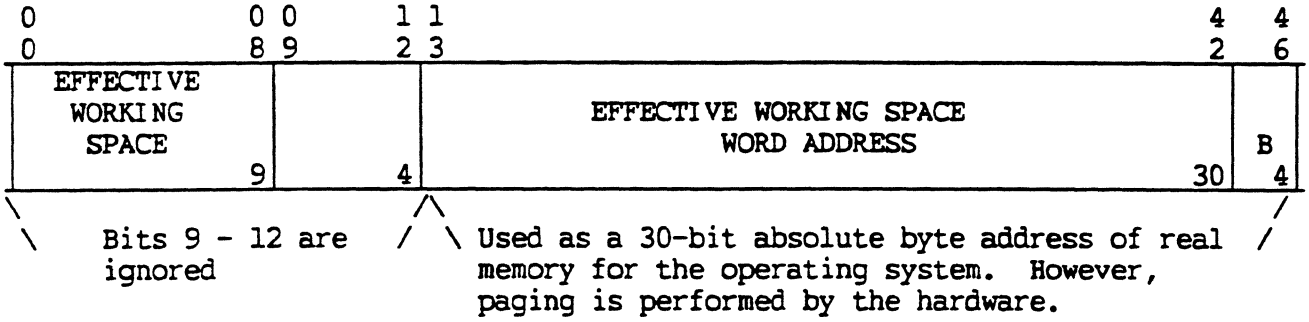


Figure 5-16. Effective Absolute Address

Paging

After generation of a virtual address, an address translation process for mapping a virtual memory address to a real memory address is performed by paging, in order to create a real memory address for accessing the real memory.

Paging does not differ between the NS or ES mode.

ADDRESS TRANSLATION PROCESS

Memory paging is an integral part of the address translation process for mapping a virtual memory address to a real memory address. Each of the 512 working spaces is supported by one page table or one section table (SCT). The working space page table directory (WSPTD) is a 512-word table, indexed by a 9-bit WSN. A WSPTD entry contains the real memory address of a page table or section table. The section table consists of up to 4K words called page table base words (PBW). Each PBW defines the real memory address of a page table. When paging is performed using section tables, PBWs cause the page table to be divided into 1K blocks and allow them to be distributed throughout memory.

PAGE TABLE DIRECTORY WORD FORMAT

The format of the page table directory word is given in Figure 5-17.

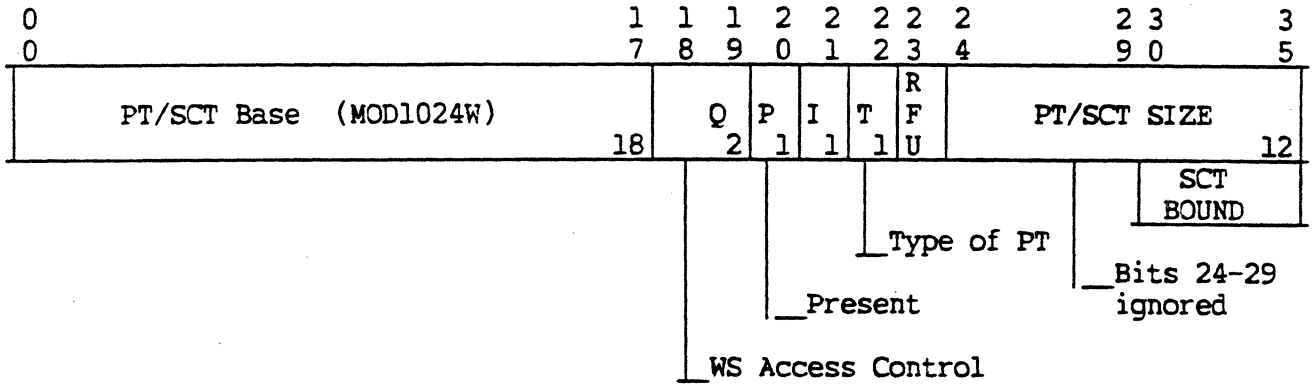


Figure 5-17. Page Table Directory Word (PTDW) Format

Bits	Description
0-17	The modulo 1024 base address (real memory address) of a page table (PT) or a section table (SCT).
18,19	Provide a hardware method to force the isolation of the WS. When one or more WS is allocated to a process, software will record in these bit positions of the associated PTDW, the relative WSN within the set of up to four possible numbers. These bits are used to check the WSN at translation from a virtual memory address to a real memory address. An SCL2 fault occurs if the check fails.
20	= 0, the PT/SCT is not present. (A missing working space fault occurs.) = 1, the PT/SCT is present.
21	Ignored
22	0 = indicates a dense PT. 1 = indicates an SCT.
23	Reserved for future use.
24-35	The size of the PT/SCT. o For a dense page table, bits 24 to 35 indicate the modulo 64 size of the PT. o For a section table, bits 30 to 35 indicate the modulo 64 size of the SCT. Bits 24-29 are ignored. o If bits 30 to 35 are zero, the size of 64 words is assumed.

PAGE TABLE BASE WORD FORMAT

The format of the page table base word is given in Figure 5-18.

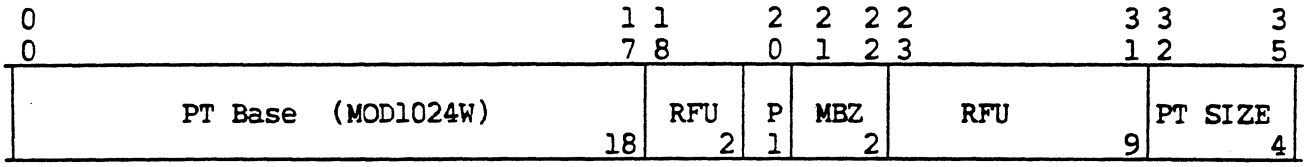


Figure 5-18. Page Table Base Word (PBW) Format

Bits	Description
0-17	Indicate the modulo 1024 base address (real memory address) of a dense page table.
18,19	Reserved for future use.
20	= 0, the PT is not present. (A missing working space fault occurs.) = 1, the PT is present.
21,22	Must be zero.
23 to 31	Reserved for future use.
32 to 35	Define the modulo 64 size of a dense page table. If 0, the size of 64 words is assumed.

PAGE TABLE WORD FORMAT

The format of the page table word is given in Figure 5-19.

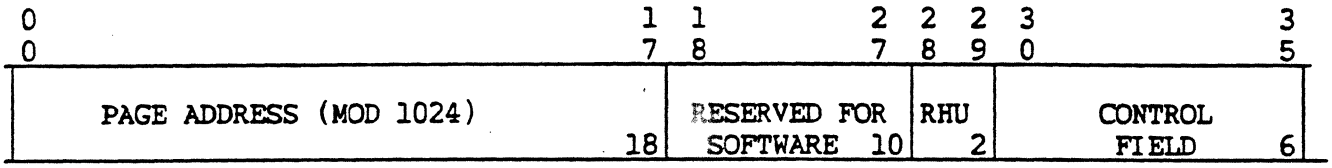


Figure 5-19. Page Table Word (PTW) Format

Bits	Description
0-17	The page modulo 1024 base address (real memory address).
18-27	Reserved for software use and may not be altered by the hardware.
28,29	Reserved for hardware use and may be changed by the hardware.

Control Field:

30	- Processor page present/missing bit = 0, page is not in memory (missing) = 1, page is in memory (present)	} Interpreted only by processor
31	- Write control bit = 0, page can not be written = 1, page can be written	} Bit 31 is interpreted by processor and IOP

Control Field:

32	- Housekeeping bit = 0, nonhousekeeping page = 1, housekeeping page	} Interpreted only by processor
33	- IOP page present/missing bit = 0, page is not in memory (missing) = 1, page is in memory (present)	} Not interpreted by processor
34	- Page modified bit = 0, page was not modified = 1, page was modified	} Interpreted only by processor
35	- Page access bit = 0, page was not accessed = 1, page was accessed	} Interpreted only by processor

When the processor accesses the page table word (PTW), the hardware checks bit 30. If bit 30 = 0, a Missing Page fault occurs and no other faults that might be caused by the page table word are checked. Refer to the discussion of "Page Table Word Control Field Faults" in Section 6.

Note that the processor and the IOP have separate bits to indicate a missing page. Thus, during I/O, a page may be present to the IOP but missing to the processor or vice-versa. When a page is accessed by the processor, and the PTW is accessed in main memory by hardware, bit 35 of the PTW is set to 1 by the hardware.

When a write occurs to a page, and the modified bit in the page table word in associative memory is 0, this bit is set to 1 and bits 34 and 35 of the page table word in main memory are set to 1 by the hardware.

Note that if a write occurs to a page, and the modified bit in the page table word in associative memory is 1, no changes are made to the page bits. Software may have reset the page access bit, bit 35, to zero. This bit remains zero under this condition.

MAPPING THE VIRTUAL ADDRESS TO A REAL ADDRESS

If a prior memory reference to the same page has already mapped that page to real memory, and if that mapping is still present in the associative memory of the processor, then the mapping is accomplished by concatenating the Word field of the virtual address to the modulo 1024 real address of the page, to produce the real address for the memory reference. Otherwise, the mapping proceeds by locating and obtaining the Page Table Directory Word (PTDW).

If the PTDW indicates that the page table is not present (PTDW.P=0), then the mapping is not completed, and a Missing Working Space fault is generated. If the page table is present (PTDW.P=1) but PTDW.Q≠1, bits 0-1 of the relative virtual address are compared and if they are not equal, then the mapping is not completed, and a Class 2 Security Fault is generated.

DENSE PAGE TABLE

When a dense page table is used, the CPU interprets the virtual address as shown in Figure 5-20.

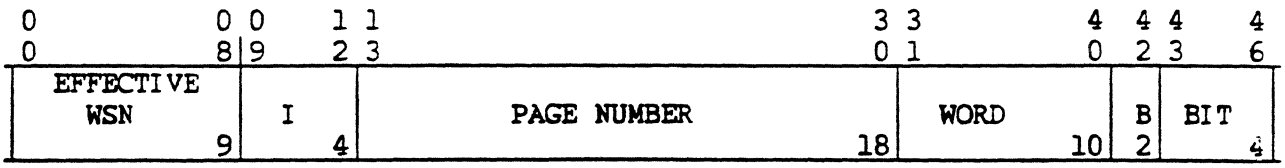


Figure 5-20. Virtual Address

Bits	Description
0-8	Working space to be accessed.
9-12	Ignored
13-30	Page number is used as an offset or index into the PT for this WSN, for locating the PTW. The page number is relative to the PT base address (real memory address) which comes from the PTDW.
31-40	Determines which word within the 1024-word page is being addressed.
41-46	Byte and bit positions within the word, if applicable.

LOCATING THE PAGE TABLE DIRECTORY WORD

The Page Directory Base Register (PDBR) contains the modulo 512 word address of the Working Space Page Table Directory (WSPTD). Figure 5-21 shows how the hardware uses the effective WS number from the virtual address as an offset into the WSPTD to obtain the Page Table Directory Word (PTDW) for address translation using a dense page table.

Figures 5-21, 5-22, 5-23, and 5-24 illustrate virtual to real mapping using a dense page table. In Figure 5-21 below, the dense page table base address in the PTDW is modulo 1024 words. PTW bits 0 to 17 are the modulo 1024W page start address.

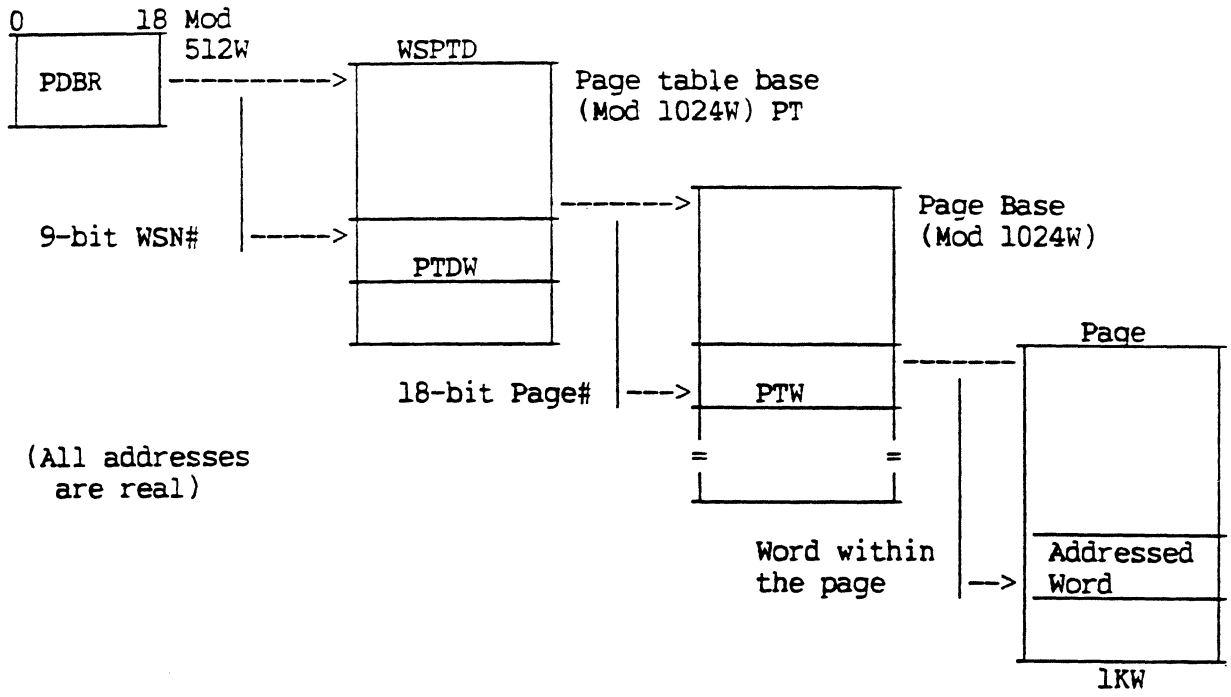


Figure 5-21. Address Mapping Using A Dense Page Table

In Figure 5-22, the PDBR indicates the base (mod 512 words)of the 512-word WSPTD. The 9-bit effective WS number is combined with the 19 bits from the PDBR to generate the real memory address to access the WSPTD. The PTDW includes the real memory address (mod 1024 words) of the page table. The PT entry location is determined by the 18-bit page number of the virtual address. The PTW includes the real memory address (mod 1024 words) of the page. The 10-bit word address field of the virtual address is combined with the 18-bit real memory address of the page to generate a 28-bit real memory word address. This generation is illustrated in Figures 5-22, 5-23, and 5-24.

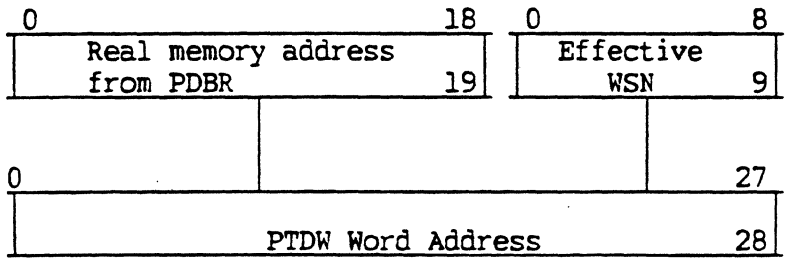


Figure 5-22. PTDW Address

Virtual to real mapping through a Dense PT is shown in Figure 5-23.

The PTDW contains the base address (0 modulo 1024W) of the PT. The address of the PTW is equal to the base address plus the 18-bit page number. The mapping of the virtual address to the real address is completed when the PTW is obtained. The mapping is then saved by the hardware in the associative memory. The PTW contains the real address (0 modulo 1024) of the page. The 10-bit word field of the virtual address is concatenated with the page real address to form the real word address.

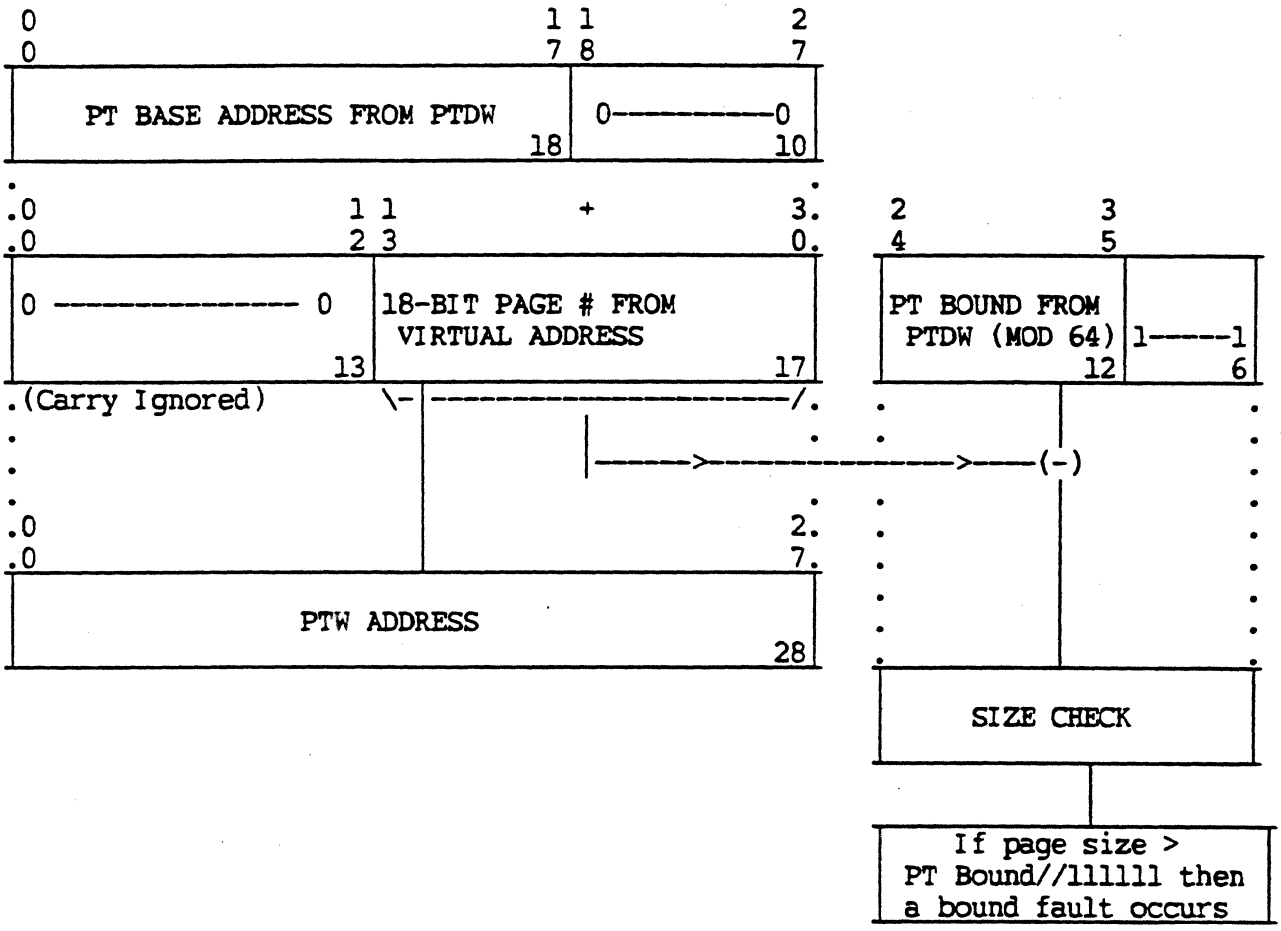


Figure 5-23. PTW Address

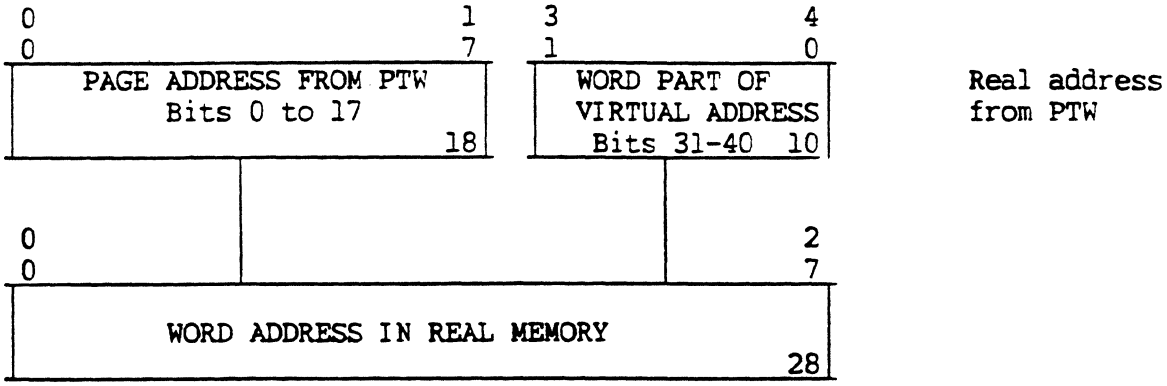


Figure 5-24. Word Address

SECTION TABLE

The section table allows the page table for a working space to be fragmented into sections. The PTDW specifies the base of the section table, which contains up to 4K of page table base words (PBW), each of which defines a page table for a section. When a section table (SCT) is specified by the PTDW, the virtual address is interpreted as shown in Figure 5-25:

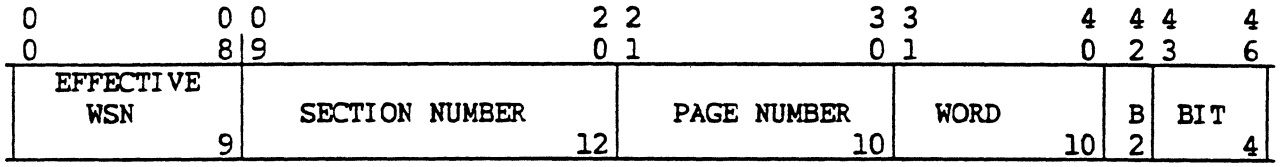


Figure 5-25. Virtual Address

Bits	Description
0-8	Working space to be accessed
9-20	Section number. An offset of the SCT base for accessing the PBW in the SCT. The SC number is a value relative to the SCT base indicated by the PTDW.
21-30	Page number is used as an offset or index into the PT for this WSN, for locating the PTW. The page number is relative to the PT base address (real memory address) indicated by the PBW.
31-40	Determines which word within the 1024-word page is being addressed
41-46	Byte and bit positions within the word, if applicable

Figure 5-26 illustrates virtual to real mapping when using a section table.

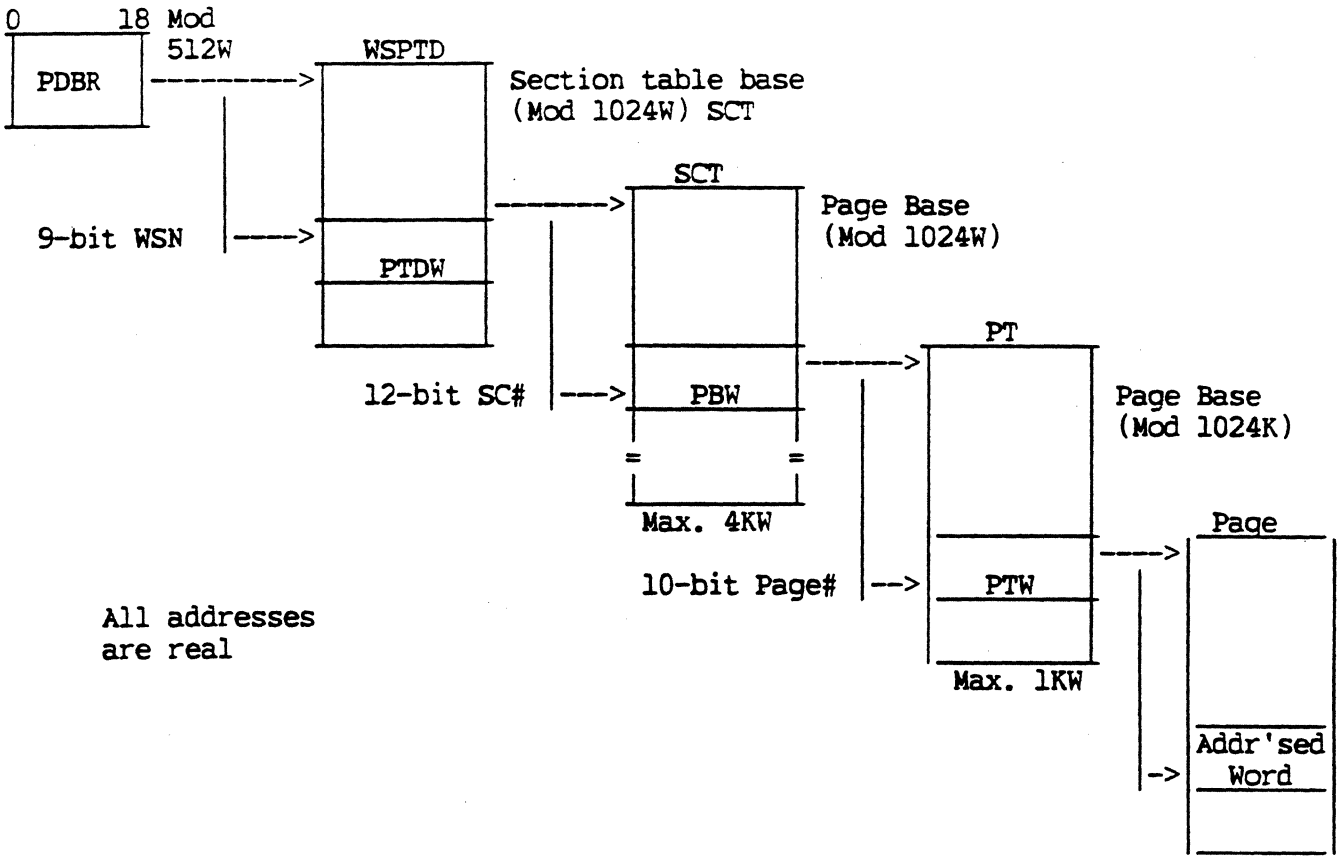


Figure 5-26. Address Mapping Using A Section Table

Development of a word address from a section table is illustrated in Figures 5-27, 5-28, and 5-29.

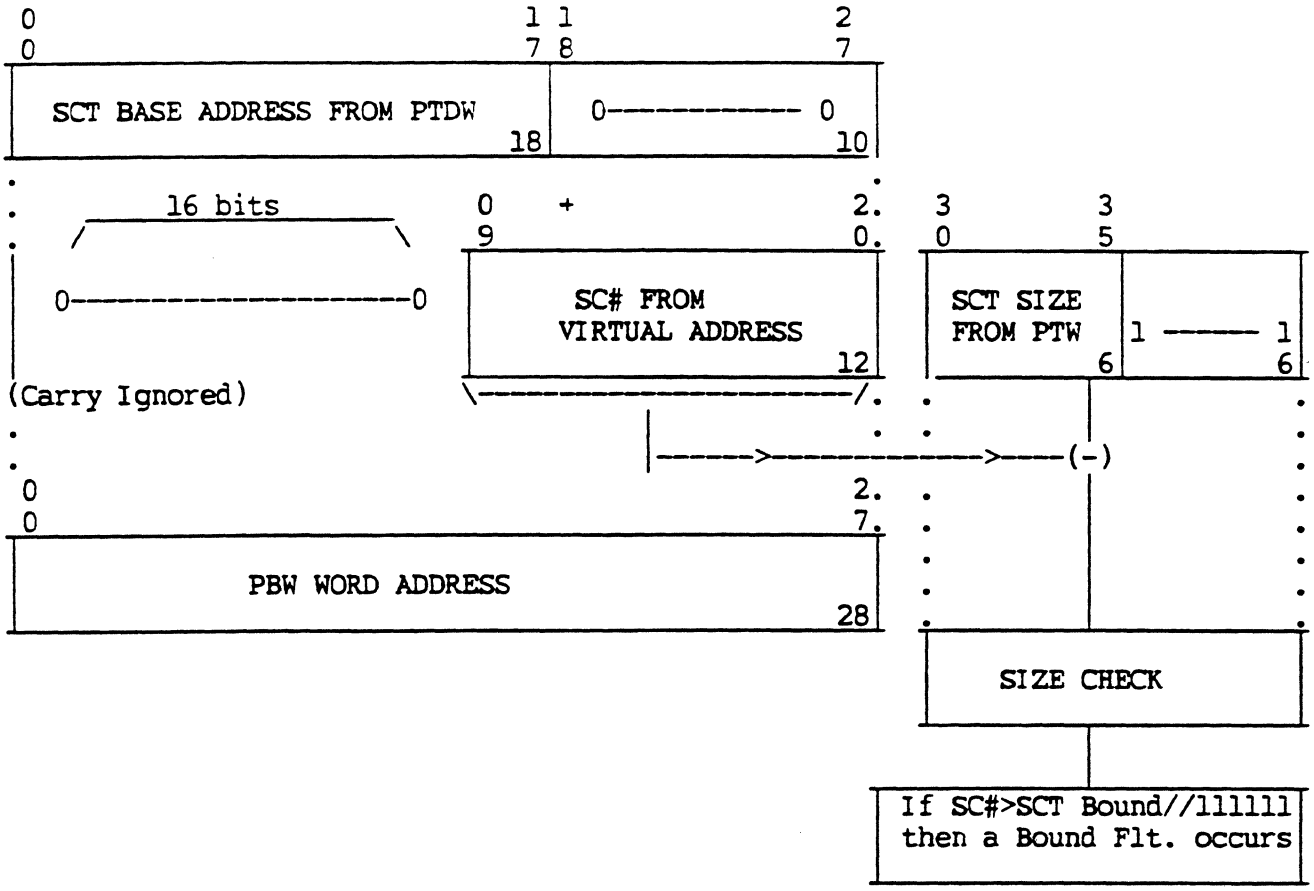


Figure 5-27. PBW Address

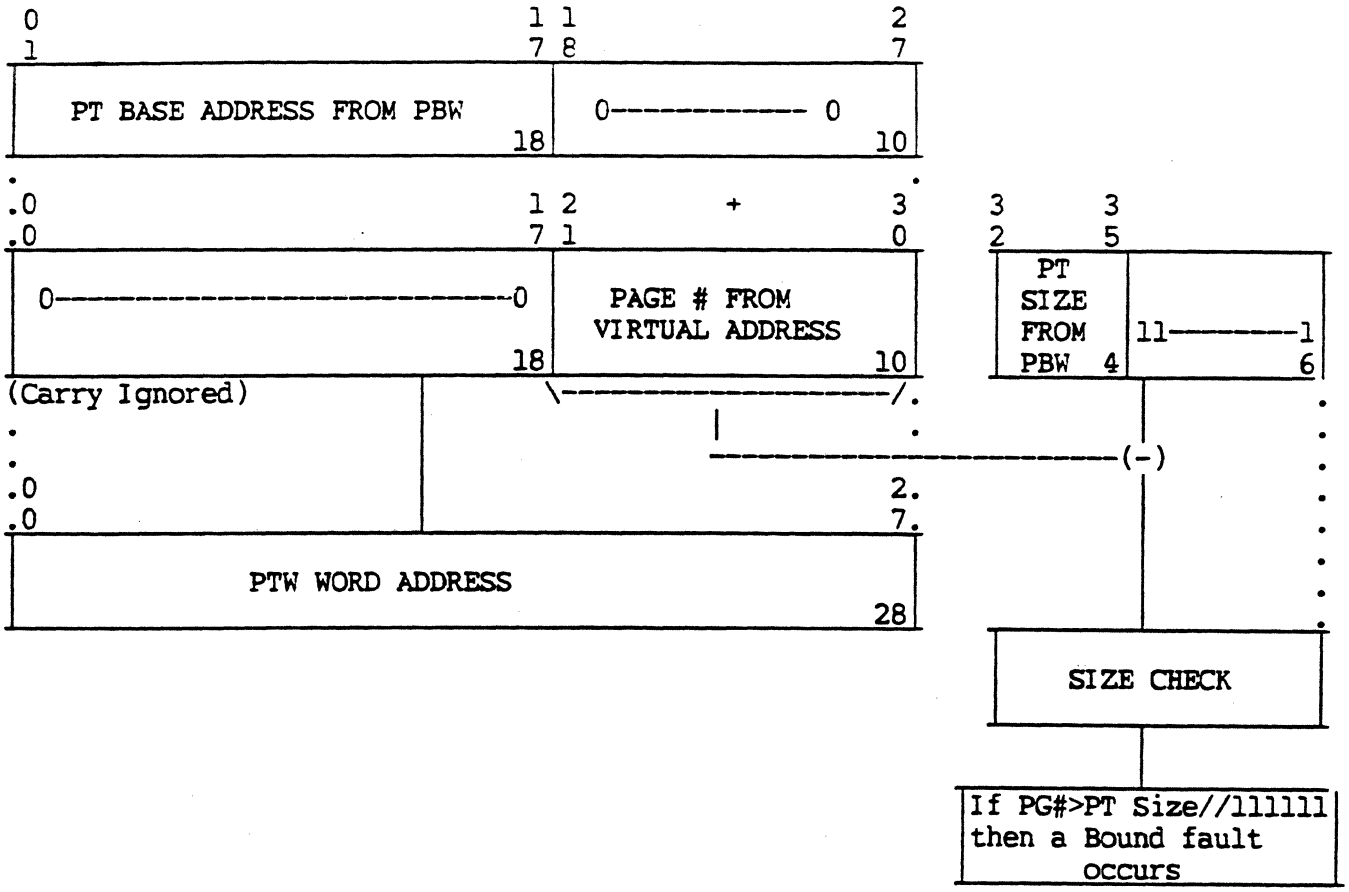


Figure 5-28. PTW Address

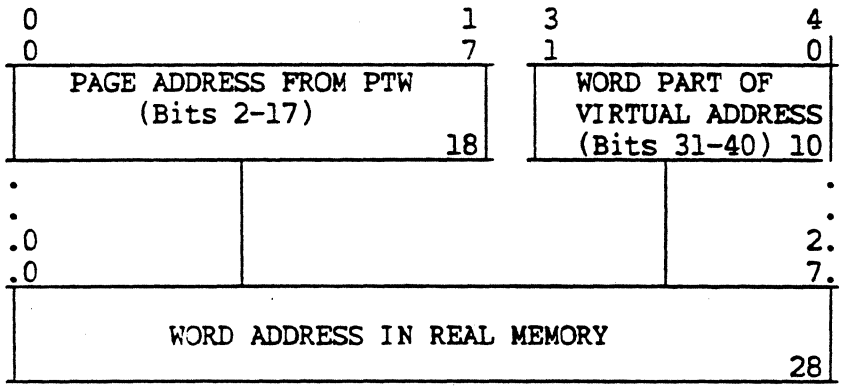


Figure 5-29. Word Address

After a virtual address has been mapped to a real address as described earlier, page table word information is stored in the associative memory (AM) in such a way that a subsequent reference to this page can be mapped in one step. The format of the data stored by an SCPR 16 from the associative memory is shown in Figure 5-30.

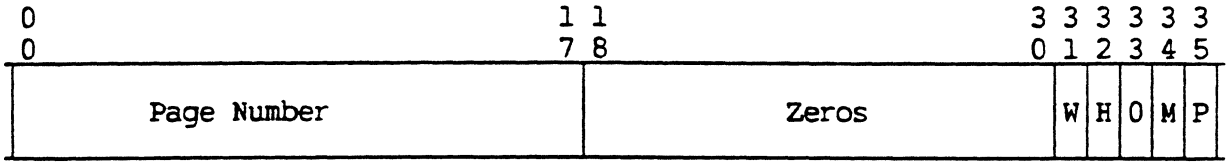


Figure 5-30. Page Table Word Associative Memory (PTWAM) Format

Bits	Description
0-17	The first 17 bits hold the page number
18-30	Zeros
31-35	Page control bits:

W - write

H - housekeeping

M - modified

P - parity on PTWAM storage

When an operand virtual address is mapped from an associative memory entry and the operation modifies the page, the hardware checks the modified (M) control bit. If the M bit in the AM entry is OFF, the processor turns the M bit of the AM entry ON, refetches the page table word for this AM entry from main memory, and turns the M control bit in the page table word ON. The access bit in the page table word is also set ON at this time, since it may have been turned OFF by the software. If the M bit of the AM entry is ON at the beginning of the mapping, no change is required.

The associative memory is arranged in 64 rows by 2 columns. Each intersection of a row and a column contains a 35-bit entry like the one shown above.

Page table directory words from associative memory are stored by SCPR 16 with the following format.

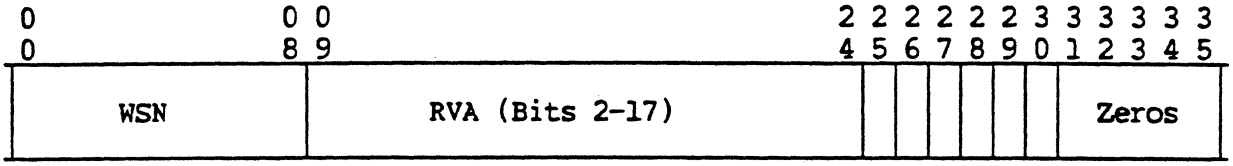


Figure 5-31. Associative Memory Directory Word

Bits	Description
0-8	Working space number
9-24	Real virtual address (RVA) bits 2-17
25	When set = 1 indicates parity error
26	When set = 1 indicates full; 0 indicates empty
27	Round robin counter 0 = level 0 1 = level 1
28	Status of level A 0 = ON 1 = OFF
29	Status of level B 0 = ON 1 = OFF
30	When set = 1 indicates enable associative memory

The PTWAM directory word is obtained from the directory with its contents placed into the A register by the Store Central Processor Register instruction SCPR with tag = 17. The word is loaded from the A register and put into the PTWAM directory by the Load Central Processor Register (LCPR) instruction. Both of these instructions must be used in Privileged Master mode.

The PTWAM has two levels, A and B, and 64 columns from a total of 128 entries. The LCPR ,17 instruction causes the following A-register bits to be loaded into the directory word pointed to by the effective address:

0 --> Full/empty bit

$C(A)_{27}$ --> Round robin counter (RRO)

$C(A)_{28}$ --> Level A set OFF

$D(A)_{29}$ --> Level B set OFF

The PTWAM has only one full/empty(F/E) bit. When F/E = 1, both Level A and Level B are full. When F/E = 0, the round robin counter (RRO) specifies whether or not Level A is full. A typical operation sequence following execution of LCPR 17 specifies the full/empty states as follows:

Entry	F/E	RRO	Level A	Level B
	0	0	Empty	Empty
1	0	1	Full	Empty
2	1	0	Full	Full
3	1	1	Full	Full
4	1	0	Full	Full

When a new address not contained in the associative memory has been mapped and the associative memory is full, the new entry replaces the older entry in the row (using the RRO algorithm).

The associative memory may be disabled (any further comparisons or matches are ignored) by:

a. Executing a CAMP instruction with effective address bits 16-17 = 1.

b. Encountering an address compare of two or more columns in one of the 64 rows.

If one of the levels is OFF, the entry is still made in that level corresponding to the state of the RRO counter. On a subsequent PTW search, the OFF state of the level is recognized and a match is not permitted.

The associative memory is cleared whenever the following occurs:

a. The processor is manually initialized.

b. The processor is enabled, and the CAMP instruction is executed with effective addrss bits 16-17 equal to 00, 10, or 11. If EA bits 16-17 = 01, the associative memory is disabled but not cleared.

c. The processor is disabled, and the CAMP instruction is executed with effective address bits 16-17 = 10.

d. The processor is disabled, and the Load Page Table Directory Base Register (LPDBR) instruction is executed.

CACHE MEMORY

A description of the visible portion of cache memory control follows. Cache directory data is returned to the A register on the instruction SCPR 15 from the entry selected by the effective address.

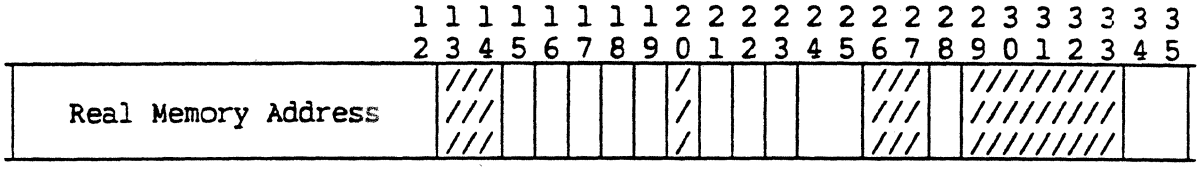


Figure 5-32. Cache Directory Word

Bits	Description
0-12	Most significant 13 bits of the real memory address
13-14	Not used
15	Parity on bits 0-9 of the real memory address
16	Cache block full/empty bit (normal mode) NOTE: When certain cache blocks are used by PATROL, these blocks are set to empty prior to normal use by the CPU.
17	Selected level parity error
18	Cache enable bit (1 = enable)
19	Cache block full/empty bit (PATROL mode)
20	Unused
21	Cache enabled for instruction fetch (1 = enabled)
22	Parity on bits 10-12 of the real memory address
23	Cache to register flag (1 = ON)

Bits	Description
24-25	Level 0,1 ON when = 1
26-27	Unused
28	Least recently used (LRU) register
29-33	Unused
34-34	Lockup fault register

Address Truncation

The instruction set contains instructions that operate on words, double-words, 9-bit bytes, 6-bit characters, 4-bit characters, and bits. Instructions and indirect and tally words that specify 6- or 9-bit characters are considered word instructions. In accessing the operand, the full byte level virtual address is determined. The address is then truncated in accordance with the address type of the instruction, and the access is also in accordance with the type of instruction.

An exception to this procedure applies to the 8-word instructions, such as LREG and SREG. The effective address is truncated to a modulo 8 word address prior to adding the base. Following the addition of the base, the virtual address is then truncated to a double-word address.

The user is responsible for ascertaining correctness of operation of an instruction as influenced by such address truncation.

Bounds Checking

Virtual memory allows specifying the base and bound of a segment to the 9-bit byte level, enabling a finer level of security control. Because the processor interfaces with word-oriented main memories, certain restrictions are also imposed to minimize the impact on performance and hardware complexity. The size of a segment described by a super descriptor is modulo 2**26 bytes; therefore, the bounds checking is always the same: BOUND (lower extended with 26 one bits) $\geq$ LOCATION + EFFECTIVE ADDRESS. The following information applies only to standard descriptors and extended descriptors.

WORD AND DOUBLE-WORD OPERATIONS

Word, double-word, or a succession of word accesses as in the LREG and SREG instructions are made to real memory word or double-word boundaries. Segments that begin or end on byte or word positions and that do not correspond to word or double-word boundaries may be accessed by word or double-word instructions. The processor adds the 2-bit byte position held in an address register (if selected) to the byte position of the base before truncating the final virtual address to point to a word or double-word. If this truncation results in the virtual address dropping below the base value, a lower bound check will declare an out-of-bounds condition in this case and a Bound fault occurs. Thus, the first word or double-word of a segment may be accessed with word-oriented instructions only when the word or double-word is entirely within the segment.

Half-word accesses, such as the LXLn instruction, are treated as word accesses in both the lower-and upper-bounds check. If a segment begins in the middle of a word, the LXLn and SXLn instructions cannot be used to access the lower half-word. If the segment ends in the middle of a word, the LDXn, STXn, LXLn, ADXn, etc., instructions cannot be used to access the upper half-word.

The STCA, STCQ, STBA, and STBQ instructions store 6-bit or 9-bit characters into character/byte locations within a word. These are considered as word accesses and require the entire word to be within the segment.

Indirect and tally words that specify character/byte locations are considered as addressing words that must be fully contained in the segment. The virtual address is truncated to the next lowest word boundary (i.e., the character position in the base is not added to the character position held in the indirect and tally word).

NOTE: This information is included to provide a warning for users of the operating system and user software. If segments are "shrunk" (see the LDDn and CLIMB instructions), and the byte portion of the virtual base is changed, a word or double-word access to the new segment may be truncated to a different location within the segment.

All instruction segments must begin at a 0 modulo 8 location and end at a 7 modulo 8 location. Any transfer or CLIMB instruction that attempts to load the instruction segment register must specify a segment base whose 5 least-significant bits are 0s, and a segment bound whose five least-significant bits are 1s. This condition allows the processor to access blocks of eight words for LPL, SPL, LREG, SREG, LAREG, and SAREG instructions with the assurance that if the first word is on an assigned page and is within the segment boundary, the other words will also be so located.

All descriptors loaded into the SSR, PSR, LSR, ASR, or DSDR registers must begin and end on double-word boundaries (the three least-significant bits of the base are 0s and the three least-significant bits of the bound are 1s).

BYTE OPERATIONS

For all 9-bit and 4-bit character operations using multiword instructions, the upper-bound check is made at the 9-bit byte level. A lower-bound check is not required since the effective address is always greater than or equal to zero.

For all 6-bit character operations using multiword instructions, the boundary checking is on a double-word basis, meaning that a double-word containing any 6-bit character of the operand must be fully in bounds. If access is attempted to a segment with a base or bound not on a double-word boundary, a Bound fault is generated.

BIT STRINGS AND TABLE OF TRANSLATE INSTRUCTION

Multiword bit string instructions and the index table of the translate instructions (MVT, TCT, and TCTR) have double-word bound checking applied. Thus, a double-word that includes any part of these operands must be fully in bounds. If access is attempted to a segment that has a base or bound not on a double-word boundary, a Bound fault is generated.

BOUND CHECK EQUATIONS

The address truncation procedure described previously forces bounds checking to vary depending upon the type of instruction specified. The resulting three upper-bound and lower-bound checks are listed in Table 5-3. A Bound fault is generated if the bound checks are violated.

Table 5-3. Bound Check Equations

Instruction	Bound Check	
Double-Word (includes bit string and 6-bit character instructions)	Upper	(BASE + EA)0-32||111≤ BASE + BOUND
	Lower	(BASE + EA)0-32||000≥ BASE
Single-Word	Upper	(BASE + EA)0-33||11 ≤ BASE + BOUND
	Lower	(BASE + EA)0-33||00 ≥ BASE
Byte (includes 9-bit byte, 4-bit byte)	Upper	EA 0-19 ≤ BOUND
	Lower	Always satisfied

The base, bound, and effective address (EA) addresses represented in the bound check equations are for 9-bit bytes. For 4-bit byte and bit instructions, the effective address represents the 9-bit byte in which these small quantities are contained. The single- and double-word bound check equations include the effect of address truncation; the truncated address is then extended to the largest byte contained therein for the upper-bound check and to the lowest byte for the lower-bound check. The byte checks refer to the byte accessed; in multibyte instructions such as MLR, the access checks are applied to each byte.

Physical accesses, which may be larger than those corresponding to a given instruction (and which therefore may include bytes not contained in the segment), are not bound checked beyond the byte range corresponding to the instruction.

SECTION 6

FAULTS AND INTERRUPTS

Faults and interrupts both result in an interruption of normal sequential processing, but there is a difference in how they originate. Generally, faults are caused by events or conditions that are internal to the processor; but interrupts are caused by events or conditions that are external to the processor. Faults and interrupts enable the processor to respond promptly when conditions occur that require system attention.

DESCRIPTION OF FAULTS AND INTERRUPTS

When the processor responds to a fault, interrupt, or special systems entry (PMME), the ICLIMB version of the CLIMB instruction is executed. Because this is an inter-domain transfer of control, an entry descriptor is required; the entry descriptor is obtained from a fixed memory location. The interrupt, fault, special systems entry, and Backup fault entry descriptor locations (in real memory) are as follows:

Location (octal)	Entry Descriptor
30-31	Interrupt
32-33	Fault
34-35	Special systems entry
40-41	Backup fault

FAULT PROCEDURES

When a fault occurs, the processor generates the appropriate fault code and executes the ICLIMB version of the CLIMB instruction. During the safe store part of the ICLIMB, the generated fault code is stored along with a flag to indicate that the safe store frame is the result of the occurrence of a fault (bit 11 of word 5 is set to 0).

If the fault occurred during a multiword instruction, the pointer and length registers will be saved in the safe store frame.

The second word of the "wired-in" ICLIMB instruction is assumed as described for interrupts. (Refer to "Interrupt Procedure" later in this section.)

If an entry descriptor is not found in the fixed fault vector location or if another fault should occur (e.g., a parity error) while the processor is attempting to CLIMB to the fault handler, the processor attempts to obtain an entry descriptor from the Backup fault vector location. If this second location does not contain an entry descriptor, the processor enters the HALT state. If the second fault occurs prior to the transfer of control to the new domain at the end of the ICLIMB, then the safe store frame will overlay the original frame (with the same information execpt for fault code). If the second fault occurs during the transfer of domains, such as a page fault when obtaining the next instruction, then a second frame is filled specifying the new domain and the fault code of the type fault that caused the backup condition.

The processor is placed in the Privileged Master mode for the execution of the "wired-in" ICLIMB instruction. Upon exiting the ICLIMB, the processor remains in the Privileged Master mode if flag bit 26 of the new instruction segment register (ISR) is 1. If flag bit 26 of the new ISR is 0, the processor cycles to Master mode.

FAULT PRIORITY

Faults are organized into five groups to establish priority for the recognition of a specific fault when two or more faults occur at the same time in different groups. (Refer to Table 6-1.)

Only one fault within a priority group can be active at any one time. If two or more faults occur concurrently within a priority group, only the fault that occurs first through normal program sequence is recognized.

FAULT RECOGNITION

Processor-detected faults can be categorized in several ways. Table 6-1 lists the faults in order of the octal fault code, shows the priority assigned by the processor, and lists the priority group number.

Faults in Groups I and II cause the operations in the processor to terminate unconditionally.

Faults in Group V are recognized under the same conditions that program interrupts are recognized. Faults in Group V have priority over program interrupts and also can be inhibited from recognition by engaging the inhibit bit in the instruction word.

Table 6-1. Processor Faults By Fault Code

Fault Code	Octal Code	Fault Name	Priority	Group
00001	02	Bound (BND)	9	IV
00010	04	Master mode entry (MME)	10	IV
00011	06	Fault tag (FTAG)	13	IV
00100	10	Timer runout (TRO)	23	V
00101	12	Command (CMD)	8	IV
00110	14	Derail (DRL)	11	IV
00111	16	Lockup (LUF)	4	II
01000	20	Connect (CON)	22	V
01001	22	Parity (PAR)	7	IV
01010	24	Illegal procedure (IPR)	12	IV
01011	26	Operation not completed (ONC)	3	II
01101	32	Overflow (OVF)	6	III
01110	34	Divide check (DIV)	5	III
01111	36	Execute (EXF)	2	I
10000	40	Security class 1 (SCL1)	14	IV
10001	42	Dynamic linking (DYN)	15	IV
10010	44	Missing segment (MSG)	16	IV
10011	46	Missing working space (MWS)	17	IV
10100	50	Missing page (MPG)	18	IV
10101	52	Security class 2 (SCL2)	19	IV
10110	54	Address trap (ADT)	21	IV
(See NOTE)	--	Safe store stack (SSSF)	20	IV

NOTE: The safe store stack overflow fault has no fault code because it may occur with any other fault. If a safe store stack fault occurs, the fault code is contained in bits 12-16 of safe store stack frame word 5. (Refer to Figures 8-7 and 8-8 for a description of the safe store stack).

FAULT CATEGORIES

There are four general categories of faults:

1. Instruction-generated faults
2. Program-generated faults
3. Virtual memory-generated faults
4. Hardware-generated faults

Instruction-Generated Faults

An instruction generated fault can be traced to the execution of a particular instruction. It may be an operating system service request or an illegally coded instruction. The instruction-generated faults are the following.

1. Master Mode Entry (MME)

   A Master Mode Entry instruction was executed.

2. Derail (DRL)

   A Derail instruction was executed.

3. Fault Tag

   A fault tag address modifier (F) was recognized. Fault tag is a variation of the Indirect then Tally modification. Indirect cycles terminate upon recognition of F, and the operation is not completed. The tag field (bits 30-35) of the instruction or indirect word is set to 40 (octal) to cause the Fault Tag fault.

4. Connect (CON)

   The processor received a signal from a system controller indicating that some processor in the system executed a CIOC instruction directed to this processor.

5. Illegal Procedure (IPR)

   The attempted execution of an illegal instruction sequence or modification generates an IPR fault. The attempted execution of a legal Master mode instruction in the Slave mode causes a Command (CMD) fault.

   The attempted execution of any of the unassigned instruction operation codes generates an Illegal Procedure fault.

   An IPR fault occurs for any register specification that contains a tag defined as illegal.

An IPR fault occurs when an attempt is made to repeat any multiword instruction with the use of the RPT, RPD, or RPL instructions[1] or to XEC or XED[2] any multiword instruction. (An XEC instruction may point to a multiword instruction; however, the descriptors for the multiword instruction must be stored in memory immediately following the XEC instruction.)

An IPR fault occurs for:

a. any attempt to address through a descriptor of type T = 7, 10, or 12-15 by any instruction

b. any attempt to address through a descriptor of type T = 5, 8, 9, or 11 by any instruction other than CLIMB

c. any attempt to address through a descriptor of type T = 1 or 3 by any instruction other than CLIMB, LDDn, or STDn

d. any attempt to address through a descriptor of type T = 1, 3, 5, 8, 9, or 11 for vectors by the LDD or CLIMB instruction

An IPR fault occurs when a CLIMB instruction is passing parameters (E = 1, DR0 = 0, 2, 4, or 6) and attempts to use a vector that has S and D fields = 00, 1760 (octal) or 00, 1761 (octal) or V = 10 binary.

An IPR fault occurs when a LDDn instruction attempts to use a vector that has S and D fields = 00, 1760 (octal), or V = 10 binary.

An IPR fault occurs when a LDPn instruction attempts to use an operand that has S and D fields = 00, 1760 (octal).

An IPR fault occurs when the S and D fields of a CLIMB instruction have S = 00 and D = 1761, or 1763 through 1767 (octal).

An IPR fault occurs if the LDDn or CLIMB instruction specifies a shrink operation (normal or data stack) of a descriptor with T = 5 or 7-15.

An IPR fault occurs during a CLIMB instruction when a valid entry descriptor does not refer to a standard descriptor (T = 0).

An IPR fault occurs if the OCLIMB version of the CLIMB instruction is specified and the Safe Store Bypass Flag is zero.

An IPR fault occurs during a CLIMB instruction that either was initiated by a fault or interrupt or encounters the special systems entry and the descriptor accessed from the fixed location is not T = 5, 8, 9, or 11.

---

1. RPT, RPD, RPL execute in NS mode only.

2. XED executes in NS mode only.

An IPR fault occurs during the CLIMB instruction when the descriptor referenced by the S and D fields is not T = 0, 1, 2, 3, 8, 9, or 11. Also, if this descriptor has T = 1 or 3, it must refer to a descriptor with T = 5, 8, 9, or 11 or the fault will occur.

An IPR fault occurs during a Load Safe Store Register (LDSS) instruction if the descriptor to be loaded into the safe store register register (SSR):

a. does not have T = 1 or 3

b. has T = 1, but does not have flag bits 20, 21, 27, and 28 = 1 and flag bits 25 and 26 = 0

c. has T = 3 but does not have flag bits 20 and 21 = 1

d. has a base that is not modulo-2 words (bits 33-35 are not equal to 000)

An IPR fault occurs during the Load Data Stack Descriptor Register (LDDSD) instruction if the descriptor to be loaded into the data stack descriptor register (DSDR):

a. does not have T = 0

b. has a base that is not modulo-2 words (bits 33-35 are not equal to 000)

c. has a bound that is not 7 modulo-8 bytes (bits 17-19 are not equal to 111)

d. has flag bit 22 (store) = 1

An IPR fault occurs during the Load Extended Address n (LDEAn) instruction if the descriptor to be loaded does not have T = 4 or 6 (super descriptor).

An IPR fault occurs during the Load Argument Stack Register (LDAS) and Load Parameter Segment Register (LDPS) instruction if the descriptor to be loaded:

a. does not have T = 1

b. has a base that is not modulo-2 words (bits 33-35 are not equal to 000)

c. has flag bit 27 equal to 1 and a bound that is not 7 modulo-8 bytes (bits 17-19 are not equal to 111)

An IPR fault occurs when an unconditional transfer (TRA, TSXn), or a satisfied conditional transfer (TNZ, TPL, etc.) attempts to load a descriptor into the instruction segment register (ISR) that either does not have type T = 0 or does not have a modulo-8 word base and bound. If this fault is detected, the ISR is not changed.

An IPR fault occurs in the CLIMB instruction when a standard descriptor (T = 0) that is to become a new ISR descriptor does not have a modulo-8 word base and bound. This fault occurs before the domain register are changed.

Program-Generated Faults

The program-generated faults occur through some action under the control of either the process itself or the operating system. There are four major categories of program generated faults, each of which has several subcategories:

1. Arithmetic Faults

   a. Overflow (OVF). An Arithmetic overflow, exponent overflow, or exponent underflow has been generated. The generation of this fault is inhibited when the overflow mask is in the masked state. Subsequent clearing of the overflow mask to the unmasked state does not generate this fault from previously set indicators. The Overflow fault mask state does not affect the setting, testing, or storing of indicators.

      For the automatic fault on truncation, the procesor executes the Overflow fault. Note that the overflow mask bit (indicator register) does not affect automatic fault on truncation.

   b. Divide Check (DIV). A Divide Check fault is generated when the actual division cannot be carried out for one of the reasons specified below:

      1) DIV instruction - if the dividend equals -2**35 and the divisor equals zero or minus 1

      2) DVF instruction - if the absolute value of the dividend is greater than or equal to the absolute value of the divisor or if the divisor equals zero

      3) FDV, FDI, DFDV DFDI instr's. - if the mantissa of the divisor equals zero

      4) DV2D, DV3D instructions - if the divisor equals zero or if the quotient is to be stored in scaled format and the calculated length required for the quotient is greater than 63.

2. Elapsed Time Interval Faults

   a. Timer Runout (TRO). This fault is generated when the count in the timer register reaches zero and cycles to minus 1. If the processor is in Privileged Master mode, the recognition of this fault will be delayed until the processor returns to the Master or Slave mode. This delay does not inhibit the counting in the timer register. (Refer to the Disconnect (DIS) instruction in Section 8 for the exception to this action.)

b. Lockup (LUF). The processor remains inhibited for greater than the lockup time. Examples of this condition are the coding TRA * or the continuous use of the inhibit bit.

Master mode lockup time is set at 128 milliseconds and Slave mode lockup time is specified by the lockup fault register as seen in the settings below. These times can be loaded in Privileged Master mode using the Load Central Processor Register (LCPR) instruction with the register specified in the tag field.

Settings of the Lockup fault register are as follows:

Bits 34-35	Milliseconds
00	8.0
01	16.0
10	32.0
11	64.0

c. Operation Not Completed (ONC) This fault is generated due to one of the following conditions:

1) No system controller is attached to the processor for the address specified.

2) Operation is not completed. An ONC fault can be generated by disabling the SCU ports via program control while the program is being executed.

NOTE: A ONC fault can also be generated by hardware malfunction.

3. Command Faults

a. Attempted execution of instructions requiring Privileged Master mode when the processor is not in Privileged Master mode.

b. Attempted use of working space register zero in Slave mode, or attempt access to working space zero when the processor is not in the Privileged Master mode.

c. Used a vector in Master mode or Slave mode with an LDDn or LDPn instruction that specifies S = 00 and D = 1761, 1763, or 1764 (octal) (type change, DSDR or SSR).

d. A connect instruction addressed to a halted or disabled port. An entry is made in the port's connect queue even though the port is halted or disabled.

NOTES:
1. A fault or interrupt places the processor in the Privileged Master mode for the execution of the "wired-in" ICLIMB instruction.

2. If a CLIMB instruction specifies the special system entry version (PMME), this fault is not checked for the access of the new ISR.

4. Bound (BND) This fault is generated when:

   a. No physical memory exists for the effective address.

   b. An address is outside the segment boundary.

   c. An attempt is made to use absolute addressing or dense paging with a relative virtual address $\geq$ 2**28 words.

   d. An attempt is made to access the contents of an empty segment (flag bit 27 = 0) of a type T = 0, 1, or 4 segment.

      NOTES: 1. When "pushing" descriptors on the argument segment during the execution of the SDRn or CLIMB instruction, the fault does not occur if flag bit 27=0 but does occur if ASR bound plus 8 bytes > 8192 bytes (2K words).

             2. If this fault occurs for any version of the CLIMB instruction, it is generated when the new descriptor for the instruction segment register (ISR) is obtained.

   e. An attempt is made to access the contents of a type T = 0, 1, 2, or 3 segment and:

      1) The upper or lower bound is exceeded.

      2) The addition of the base and the effective address fields produces a carry.

   f. An attempt is made to access the contents of a type T = 4 or 6 segment and:

      1) The bound field is exceeded.

      2) The addition of either the location and effective address fields or the location, effective address, and base fields produces a carry.

   g. The E field equals 1 during the execution of the CLIMB instruction, descriptor register 0 contains a T = 1 descriptor (parameters are framed by descriptor register 0), and P+1 > DR0 bound, or DR0 flag bit 27 = 0 (bound not valid).

   h. Boundary violations occur in the shrink operation as indicated in the description of the LDDn instruction in Section 8, or when preparing descriptors during a CLIMB instruction.

   i. An attempt is made to execute a multiword instruction that specifies 6-bit or bit string data in a segment whose base or bound is not modulo-2 words.

Virtual Memory-Generated Faults

Virtual memory-generated faults are:

1. Security Fault, Class 1 (SCL1) occurs as follows:

   a. Upon an attempt to obtain instructions via a sequential instruction fetch, an unconditional transfer, a satisfied conditional transfer, or a CLIMB instruction in one of the illegal processor modes specified in Table 6-2.

Table 6-2. Processor Modes

Bit Status	Privileged Master Mode	Master Mode		Slave Mode	Illegal Combinations(1)			
Master Mode bit in indicator register (IR)	ON	ON		OFF	ON	OFF	OFF	OFF
Privileged bit in instruction segment register	ON	OFF		OFF	ON	ON	ON	OFF
Housekeeping bit 32 in page table word (PTW) for the instruction(2)	ON	ON	OFF	OFF	OFF	ON	OFF	ON

(1) Results in a Security Fault, Class 1

(2) The housekeeping bit is assumed to be ON when working space zero is referenced and the processor addresses real memory directly. (There is no page table from which to retrieve the housekeeping bit.)

   b. Upon attempt to modify a housekeeping page of a type T = 0, 2, 4, or 6 segment in Master mode

      Housekeeping pages of type T = 1 or 3 segments may be modified in Master mode under the following conditions:

      1) CLIMB instruction - Safe store and push parameters on the argument stack

      2) SDR<u>n</u> instruction - Push to the argument stack

      3) STD<u>n</u> instruction - If instruction bit 29 = 1 and DRm is T = 1 or 3

   c. Upon an attempt to access or modify a housekeeping page of a type T = 0, 2, 2, 4, 6 segment in Slave mode.

NOTE: When a CLIMB instruction is executed in Slave mode and it invokes the special systems entry (PMME), the Security fault, class 1 occurs if E = 1, DR0 = 0, 2, 4, or 6, and a housekeeping page is accessed.

This condition cannot occur for the SDRn instruction but occurs for the LDPn, LDDn, CLIMB, and STDn instructions as follows:

1) LDPn - operand access

2) LDDn - vector access(es) and data stack clear

3) CLIMB - vector access(es) and the access for the second word of the instruction If the system entry (PMME) is invoked, the fault detection is not overwritten.

4) STDn - instruction bit 29 = 1; DRm type T = 0, 2, 4, or 6

d. Upon an attempt to access or alter a nonhousekeeping page of a type T = 1, 3, 8, 9, or 11 segment

This condition only occurs for the LDDn, LDPn, CLIMB, SDRn, and STDn instructions. Any other reference to a type T = 1 or 3 segment causes an IPR fault. The conditions under which the Security Fault, class 1, can occur are:

LDDn or LDPn - accesses of descriptor from parameter segment (S = 00, D < 1760), argument segment (S = 10), or linkage segment (S = -1 or 11)

LDDn - instruction bit 29 = 1, DRm is type T = 1 or 3

CLIMB - accesses to obtain the new LSR and ISR descriptors

- accesses for safe store or restore

- accesses to the parameter, argument, or linkage segments for descriptors to be passed

- accesses to the argument segment to store parameters

STDn - instruction bit 29 = 1 and DRm is type T = 1 or 3

STRn - write to argument segment

2. Dynamic Linking Fault (DYN)

A Dynamic Linking fault occurs if the S, D field of a programmed CLIMB (CALL, LTRAS, LTRAD) points to a dynamic linking descriptor (T = 5), or to an indirect descriptor (T = 1 or 3) which points to a dynamic linking descriptor. Any attempt by any other instruction to address through a dynamic linking descriptor causes an IPR fault.

3. Missing Segment Fault (MSG)

   A Missing Segment fault is generated when an attempt is made to access memory using a segment descriptor whose flag bit 28 equals zero. This condition can occur only with descriptor types T = 0, 1, or 4.

4. Missing Working Space Fault (MWS)

   A Missing Working Space fault is generated during virtual to real memory mapping when the word obtained from the working space page table directory has bit 20 (page table or section table missing/present) equal to zero.

5. Missing Page Fault (MPG)

   A Missing Page fault is generated during virtual to real memory mapping when the page table word has bit 30 (page missing/present) equal to zero When a Missing Page fault occurs, the processor stores an appropriate value in FRTRY to indicate whether or not the fault is recoverable if software supplies the missing page and returns to the program.

   0 = Missing Page fault is not recoverable

   1 = Missing Page fault is recoverable

   Word 5, bit 0 of the safe store frame is defined as the retry flag (FRTRY). FRTRY has a defined value only when a Missing Page fault occurs. The value of FRTRY is undefined for all other faults.

   When a Missing Page fault occurs, the processor stores an appropriate value in FRTRY to indicate whether or not the fault is recoverable if software supplies the missing page and returns to the program.

   0 = Missing Page fault is recoverable

   1 = Missing Page fault is not recoverable

   Recoverable means that if the faulting instruction did not modify the instruction being executed or any of its string descriptors, and if software pages in the missing page updates the PTW and OCLIMBs, then execution is resumed exactly as if the fault had not occurred, except for the time delay.

   The only reasons for which the processor sets FRTRY = 1 (not recoverable) in the safe store frame are:

   1) Occurrence of a Missing Page fault while executing an RPT, RPD, or RPL instruction[1].

   2) Occurrence of a Missing Page fault while executing an instruction pointed to by an XEC or XED[2] instruction

---

1. RPT, RPD, RPL execute in NS mode only.

2. XED executes in NS mode only.

3) Occurrence of a Missing Page fault during an indirect and tally operation

Before the EIS numeric, MVE, DTB, or BTD instructions execute, all pages containing parts of the operands and pages in which the results are to be stored must be in memory concurrently. Thus, in processing a Missing Page fault on one of these instructions, the paging software should not remove one of the pages referenced by the instruction; otherwise, upon return to the instruction, another Missing Page fault will occur.

6. Security Fault, Class 2 (SCL2)

A security Fault, class 2, is generated for the following field violations on descriptors and page table words:

a. In a segment descriptor, if an attempt is made to violate flag bits 20, 21, 22, or 25 (read, write, store, or execute) as follows:

1) An attempt is made to read any type of data (except instructions for execution and for the ISR in the CLIMB instruction) from a segment whose descriptor has flag bit 20 = 0 (read not allowed)

2) An attempt is made to alter (write) a segment whose flag bit 21 = 0, except when pushing descriptors on the argument stack during the CLIMB or SDRn instructions

3) An attempt is made to store data into type T = 1 or 3 segments using the STDn instruction and the descriptor being stored does not have store permission (bit 18 of an entry descriptor with type T = 8, 9, or 11; bit 22 for all other descriptor types)

4) An attempt is made to execute a transfer instruction to a segment in which the execute control flag (bit 25) does not equal 1. This fault is also detected in the CLIMB instruction when the new ISR is obtained before any registers have changed

b. In a page table word, if an attempt is made to violate flag bit 31 (write control)

A Security fault, class 2, is generated when bits 18 and 19 (working space access control) of the page table directory word do not match bits 0 and 1 of the 36-bit relative virtual address (attempt to violate working space).

This fault is also generated during the execution of the OCLIMB version of the CLIMB instruction if the data being loaded from the safe store frame is incorrect as follows:

a. The descriptor to be loaded into the ISR does not have the following format:

1) Type field T = 0

2) Flag field bits 25, 27, and 28 = 1

3) Base field = 0 modulo-32 bytes

4) Bound field = 31 modulo-32 bytes

b. The descriptors to be loaded into the PSR and ASR do not have the following format:

1) Type field T - 1

2) Base = 0 modulo-8 bytes

3) Bound = 7 modulo-8 bytes when flag bit 27 = 1

c. The descriptor to be loaded into the LSR does not have the following format:

1) Type field T = 1

2) Flags field bits 20, 23, 27, and 28 = 1, and bits 21, 24, 25, and 26 = 0.

3) Base field = 0 module-8 bytes

4) Bound field = 7 modulo-8 bytes

A Security Fault, class 2, is generated on intersegment transfers when flag bit 25 = 0 in the descriptor for the target segment.

7. Safe Store Stack Fault (SSSF)

The Safe Store Stack fault occurs to report to the operating system that the safe store stack has only one or two 64-word or 80-word frames remaining. Two different conditions cause a Safe Store fault.

a. If the safe store stack overflow occurs as a result of a CLIMB instruction, two frames are stored:

1) The first frame is the normal calling domain frame without the overflow flag set.

2) The second frame is set up to return control to the first instruction of the called domain.

The overflow flag is set. Control passes to the fault processor via the entry descriptor at real memory address 32-33 (octal).

The hardware detects a safe store overflow condition by assuming a worst case condition -- two full frames must remain available after a normal, successful CLIMB, or overflow will be reported. Thus, if in the NS mode the SSR bound --

< 191 words + 3 bytes (allows three more 64-word frames)

safe store overflow occurs.

If the processor is in ES mode, the formula for the SSR bound is --

< 239 words + 3 bytes (allows three more 80-word frames)

b. While generating the safe store frame, the hardware updates the SSR base and bound to determine whether a Safe Store Stack fault should be indicated in the safe store frame together with the original fault or interrupt. If the fault or interrupt exhausts the safe store stack, the frame is stored with the safe store overflow flag set to 1 in word 5 bit 10. The original fault code or interrupt cell number is stored in word 5, bits 12-16. Control is passed through the entry vector at real memory address 32-33 (octal) to the fault processor. (The Safe Store Stack fault is not executed; a separate safe store stack frame is not stored.) The SSR points to the current stack frame (i.e., the one just laid down). The bound includes the current frame plus any available stack space.

NOTE: GCOS monitors the SSSF bit in each fault or interrupt frame in the safe store stack and initiates appropriate action whenever this bit is set to 1.

c. Refer to Figures 8-7 and 8-8 for a description of the safe store stack.

8. Backup Fault

A Backup fault occurs if a fault or interrrupt occurs during the initiation of a "wired-in" ICLIMB instruction, of if any fault occurs during the execution of this ICLIMB.

A Backup fault also occurs if there is an SSR Bound fault. A succession of Safe Store Stack faults without any increase in the safe store frame bound, causes an SSR Bound fault.

A safe store frame is not laid down for the Backup fault. However, the Backup fault flag is set in the CPU mode register. If another fault, of any type, occurs with the Backup fault flag set, the CPU will halt. When a Backup fault occurs, software is advised to initiate a memory dump. Software is also responsible for resetting the Backup fault flag.

Hardware-Generated Faults

The hardware generated faults generally occur because a failure occurred in the hardware. Hardware generated faults are:

1. Operation Not Completed (ONC). This fault is generated because one of the following conditions occurred:

   a. The processor did not generate a memory operation within 1 to 2 milliseconds and is not executing the Delay Until Interrupt Signal (DIS) instruction.

   b. The system controller terminated a double-precision cycle.

   c. When returning to an interrupted multiword instruction, incorrect data is loaded into the Pointer and Length Registers.

2. Parity (PAR). This fault is generated when a parity error is detected in any of the following:

   a. Single- or double-word fetch. If the odd instruction contains a parity error, the instruction counter retains the location of the even instruction.

   b. Indirect word fetch. If a parity error exists in an indirect then tally word in which the word is normally altered and replaced, the contents of the memory location are unaffected.

   c. Operand fetch. When a single-precision operand, C(Y), is requested, the contents of the memory pair at Y and Y+1, where Y is even, or Y-1 and Y, where Y is odd, are read from memory. The system controller does not report a parity error if it occurs in C(Y+1) or C(Y-1), but restores the C(Y+1) or C(Y-1) with its parity bit unchanged.

   d. On any instruction for which the C(Y) are taken from a memory location (this includes the "to storage" instructions such as ASA and ANSA), the processor operation is completed with the faulty operand before entering the fault routing.

   e. On data from the system controller

   f. On data from the processor data bus

   g. On zone-address-command (ZAC) lines in the system controller and memory units

   The generation of this fault is inhibited when the parity mask indicator is in the masked state. Subsequent clearing of the parity mask to the unmasked state does not generate this fault from a previously set parity error indicator. The parity mask does not affect the setting, testing, or storing of the parity error indicator.

3. Execute Fault (EXF). An Execute fault is generated by the maintenance interface and the command E/F (Execute Fault) that forces the fault.

MODE FAULTS

Privileged Master Mode Faults

When the processor is in Priviliged Master (nonabsolute addressing) mode, all instructions must be fetched from housekeeping pages of type T = 0 segments. An attempt to obtain an instruction from a nonhousekeeping page causes a Security Fault, class 1. An exception applies for those instructions executed by an XEC or XED[1]. Such instructions may be accessed from either housekeeping or nonhousekeeping pages.

References to type T = 0, 2, 4, and 6 segments to access or alter data other than instructions may be to either housekeeping or nonhousekeeping pages. References to type T = 1 and 3 segments for descriptors must be to housekeeping pages or a Security fault, class 1, is generated.

Master Mode Faults

When the processor is in Master mode, instructions may be fetched from housekeeping or nonhousekeeping pages of type T = 0 segments; operands may be fetched from housekeeping or nonhousekeeping pages of type T = 0, 2, 4, or 6 segments. However, operands may not be stored on housekeeping pages (only Privileged Master mode instructions may modify these housekeeping pages); any attempt to modify a housekeeping page in Master mode causes a Security fault, class 1.

The only instructions that may modify type T = 1 or 3 segments without generating an IPR fault are the CLIMB (safe store and pushing parameters on the argument stack), the SDR*n*, and the STD*n* instructions. For these operations, housekeeping pages must be referenced or a Security fault, class 1, is generated.

Slave Mode Faults

When the processor is in Slave mode, instructions must be fetched from nonhousekeeping pages of type T = 0 segments. Attempt to obtain an instruction from a housekeeping page results in a Security fault, class 1. Operands must be fetched from or stored into nonhousekeeping pages of type T = 0, 2, 4, or 6 segments. Since descriptors in type T = 1 or 3 segments are not treated as operands, they may be stored or fetched from housekeeping pages in Slave mode. Thus, the SDR*n* and STD*n* instructions may store the contents of a DR*n* in a type T = 1 or 3 segment, but the page must be a housekeeping page; otherwise, a Security fault, class 1 is generated. Also, the LDD*n*, LDP*n*, and CLIMB instructions may obtain descriptors from a type T = 1 or 3 segment, but the page must be a housekeeping page; otherwise, a Security fault, class 1, is generated.

---

1. XED executes in NS mode only.

Any Mode Faults

Instructions that may refer to type T = 1 or 3 segments (LDPn, LDDn, SDRn, STDn, and CLIMB) must refer to a housekeeping page when obtaining or storing the identified descriptor or safe store data; otherwise, a Security fault, class 1, is generated.

Privileged instructions (such as LDSS, LDAS, and STSS) that load descriptors from type T = 0, 2, 4, or 6 segments into registers, or store descriptors from registers into segments, do not require the housekeeping bit.

Nonprivileged instructions (such as STAS, STPS, and STDn) that store descriptors from registers into T = 0, 2, 4, or 6 segments do not require the housekeeping bit. (However, the STDn instruction may refer to either main memory or descriptor memory.)

Nonprivileged instructions (such as STAS, STPS, and STDn) that store descriptors from registers into T = 0, 2, 4, or 6 segments do not require the housekeeping bit. (However, the STDn instruction may refer to either main memory or descriptor memory.)

MISCELLANEOUS FAULTS

Segment Descriptor Flag Faults

The flags field in a segment descriptor provides the operating system software a procedure for assigning use attributes to the address space framed by the segment descriptor. Once assigned by software, these attributes defined by the flags field are hardware-enforced. The following is a discussion of the use of the flags field and the manner in which faults are generated upon an attempt to "violate" one of the flags. The definition of the flags field is described in Section 3 "Memory Organization".

1. Read/Write Permission Flags (bits 20-21). The read/write flags apply to memory accesses for operands, descriptors, and indirect words from T = 0, 1, 2, 3, 4, and 6 segments (obtaining instructions from a segment is controlled by the execute flag). Thus, in preparing the operand address for a read-from-memory instruction (e.g., LDA), the hardware checks the read flag to determine whether or not a read from memory is allowed, the hardware terminates the operation with a Security fault, class 2, and the page accessed bit in the PTW is not set. In a similar manner, when preparing the operand address for store-to-memory instructions (e.g., STA), the hardware checks the write flag to determine whether or not a store operation is allowed in the segment; if not, a Security fault, class 2, is generated, the page accessed and modified bits in the PTW are not set, and the operand is not stored.

Write permission is not needed for the SDRn instruction, for pushing descriptors on the argument segment in the CLIMB instruction, or for the STDn instruction when bit 29 = 1 and the descriptor in DRm has T = 1 or 3.

When a read-alter-rewrite (RAR) operation (e.g., AOS instruction) is performed, the write flag is checked on the read cycle. Thus, if write permission is not allowed, a Security fault, class 2, occurs before the read portion is executed, preventing any change in the indicators.

All indirect operand address preparation requires that the segment have read permission to obtain the indirect word. For an Indirect then Tally operation, the segment must have both read and write permission; read permission to obtain the indirect word and write permission to store. If these permissions are not granted, a Security fault, class 2, is generated.

The segment descriptor contained in the instruction segment register (ISR) must have execute permission (see following description of execute flag).

Read permission is not required to access a current instruction segment. Thus, in preparing an operand address using the ISR (bit 29 of instruction = 0 or, for multiword instruction, the AR bit of the MF field = 0), a read-from-memory is always permitted independent of the read flag (write flag must still be checked as described above for a store operation). The execute flag overrides the read flag only when the descriptor is in the ISR.

When an XEC or XED[1] instruction refers to its operand with bit 29 ON (using some DRn), the operand descriptor in the DRn must provide read permission (execute permission is not required).

2. Store By STDn Permission Flag (bit 22; or bit 18 of T = 8, 9, and 11 descriptors). This flag is checked by the hardware only during the execution of an STDn instruction that is to store a DRn in a T = 1 or 3 segment. An attempt to save a DRn in a T = 1 or 3 segment with the DRn store flag bit = 0 causes a Security Fault, class 2.

3. Bit 23. This flag is undefined. The DPS 8000 does not support a bypass cache flag. Instead, the two instructions Store A Conditional On Q (STACQ) and Store A Conditional (STAC) should be used by software when modifying PTWs. These instructions cause a read-lock/write-unlock sequence from/to memory. Cache is bypassed; if a cache hit occurs and the conditional test is satisfied, then the cache block is updated. (Refer the individual descriptions of STACQ and STAC in Section 8.)

4. Execute Flag (bit 25). The execute flag determines whether instructions from the segment may be executed. A segment that has execute permission does not require read permission in order to execute instructions; to execute instructions encompasses reading them from memory (instruction fetch).

---

1. XED executes in NS mode only.

The execute flag is checked by the hardware before a new instruction segment descriptor is loaded into the ISR during execution of the CLIMB instruction or one of the transfer instructions that has bit 29 = 1. Thus, if an attempt is made to load the ISR with a descriptor of type T = 0 that has flag bit 25 = 0 (no execute), a Security fault, class 2, occurs.

5. Privileged Flag (bit 26). The privileged flag applies only to instruction segments. To load the ISR with a descriptor of type T = 0 that has flag bit 26 = 1 (privileged), the Master mode indicator bit must be ON (except during an OCLIMB, ICLIMB, PCLIMB, or GCLIMB instruction that either invokes the special systems entry or is the result of a fault or interrupt); otherwise, a Security fault, class 1, occurs. With the processor executing in Privileged Master mode, operands and instructions executed by an XEC or XED[1] may originate from nonprivileged segments. When the processor is in Master mode or Slave mode, the instructions executed by an XEC or XED may originate from a privileged segment; that is, the hardware does not check the privileged bit of the segment from which the XEC or XED instruction obtains the instructions to be executed.

6. Bound Valid Flag (bit 27). The bound valid flag specifies that the bound field of the descriptor is valid (the descriptor describes a nonempty segment). Any attempt to access an empty segment of type T = 0, 1, or 4 (flag bit 27 = 0) results in a BND fault. The hardware does not allow the ISR to be loaded with the descriptor in which the bound is not valid. The bound valid flag has a somewhat different use with respect to the ASR in that descriptors may be pushed on the argument stack when the stack descriptor indicates not valid and ASR flag bit 27 is set to 1 by the hardware (see the CLIMB and SDRn instructions in Section 8).

7. Available Segment Flag (bit 28). The available segment flag indicates whether or not the segment is present in real memory (bit 28 = 1). Any attempt to generate a memory address using a type T = 0, 1, or 4 segment descriptor that has bit 28 = 0 (segment not availaable) causes a Missing Segment fault. The hardware does not allow the ISR to be loaded with a "missing" segment descriptor. For type T = 2, 3, or 6 descriptors, the segment present bit is assumed to be 1 and the segment must be available.

Page Table Word Control Field Faults

Certain control field bits of the page table word (PTW) are monitored by the hardware and may cause particular faults to occur. Each bit of the PTW control field and associated faulting is discussed below (the PTW) format is described in Section 5.

1. Processor Page Present/Missing Control Field (bit 30). Each time the processor hardware fetches a PTW in mapping a virtual address to a real address, control field bit 30 is checked. If bit 30 = 0 (page missing), a Missing Page fault is generated; if bit 30 = 1 (page present), the operation continues.

---

1. XED executes in NS mode only.

2. Write Control Field (bit 31). The PTW control field bit 31 provides for controlling a memory write operation to the page level by processors and IMX. Even though the segment containing the page may have flag field write permission, writing (altering) the page may be denied at the page level. Thus, a memory store (write) operation requires both segment descriptor flag field write permission and PTW control field write permission. If a PTW has write permission, but the segment descriptor does not, the segment write condition takes precedence, causing a Security fault, class 2.

   The segment descriptor write flag is checked during operand address preparation for a store-to-memory operation; if write permission is denied, the instruction is terminated and the PTW write control field is not checked.

   Thus, when a store-to-memory operation proceeds to the point where the PTW is obtained, PTW bit 31 is checked. If bit 31 = 1 (write permission), the operation continues; if bit 31 = 0 (write denied), the operation terminates with a Security fault, class 2.

3. Housekeeping Control Field (bit 32). (Processor only) - The PTW housekeeping bit is used by the operating system to enable allocation in page units of use attributes depending upon the processor mode. (Allocations in the three processor modes are described below.) The hardware checks the PTW housekeeping bit on all instruction fetches and stores, and all segment descriptor fetches and stores. Instructions and operands must be contained in a segment described with type T = 0, 2, 4, 6, 12, or 14 segment descriptor. The page may be either a housekeeping or nonhousekeeping page. The segment descriptors must be contained in a type T = 1 or 3 segment, and the page must be a housekeeping page.

   a. Privileged Master Mode

      When the processor is in Privileged Master mode, all instructions must be fetched from housekeeping pages of type T = 0 segments. An attempt to obtain an instruction from a nonhousekeeping page causes a Class 1 Security Fault. An exception applies for those instructions executed by an XEC or XED. Fetching and storing of operands may be performed for both housekeeping and nonhousekeeping pages.

      References to a type T = 0, 2, 4, 6, 12, or 14 segment to access or alter data other than instructions may be to either housekeeping or nonhousekeeping pages. The segment descriptors must be contained in a type T = 1 or 3 segment and the page must be a housekeeping page or a Class 1 Security Fault will be generated.

b. Master Mode

When the processor is in Master mode, instructions may be fetched from housekeeping or nonhousekeeping pages of type T = 0 segments; operands may be fetched from housekeeping or nonhousekeeping pages of type T = 0, 2, 4, 6, 12 or 14 segment. However, operands may not be stored on housekeeping pages (only Privileged Master mode instructions may modifiy these housekeeping pages); any attempt to modify a housekeeping page in Master mode causes a Class 1 Security Fault.

Because segment descriptors are not processed as operands, the SDRn and STDn instructions may be used to store DRn content in type T = 1 or 3 segments in housekeeping pages. All segment descriptor segment pages must be housekeeping pages or a Class 1 Security Fault occurs and the instruction is terminated.

c. Slave Mode

When the processor is in Slave mode, instructions must be fetched from nonhousekeeping pages of type T = 0 segments. Attempt to obtain an instruction from a housekeeping page results in a Class Security Fault. Operands must be fetched from or stored into nonhousekeeping pages of type T = 0, 2, 4, 6, 12, or 14 segments. Since descriptors in type T = 1 or 3 segments are not treated as operands, they may be stored or fetched from housekeeping pages in Slave mode. Thus, the SDRn and STDn instructions may store the contents of a DRn in a type T = 1 or 3 segment. In this case, the page must be a housekeeping page or a Class 1 Security Fault occurs. With the LDDn, LDPn, and CLIMB instructions, segment descriptors may be obtained from a type T = 1 or 3 segment. In this case, the page must be a housekeeping page or a Class 1 Security fault occurs.

d. All Modes

Instructions that may refer to type T = 1 or 3 segments (LDPn, LDDn, SDRn STDn, and CLIMB) must refer to a housekeeping page when fetching or storing the identified descriptor or safe store data; otherwise, a Class 1 Security Fault is generated.

Privileged instructions (such as LDSS, LDAS, and STSS) that load descriptors from type T = 0, 2, 4, 6, 12 or 14 segments into register, or store descriptors from registers into segments, do not require that the housekeeping bit be set ON.

Non privileged instructions (such as STAS, STPS, and STDn) that store descriptors from registers into T = 0, 2, 4, 6, 12, or 14 segments access normal memory areas and do not require the housekeeping bit. The STDn instruction accesses both normal memory areas and memory areas which contain segment descriptors.

4. IMX Page Present/Missing Control Field (bit 33). This bit is not monitored or changed by the processor hardware.

5. Page Modified Control Field (bit 34). Each time a processor performs a write (store) on a page and bit 34 of the PTW = 0, the hardware sets bit 34 of the associated PTW = 1 to indicate that the page has been modified. No fault is associated with bit 34.

6. Page Access Control Field (bit 35). Each time a page is accessed by a processor (either read or write) and bit 35 of the PTW = 0, the hardware sets PTW bit 35 = 1 to indicate that the page has been accessed. No fault is associatd with bit 35.

INTERRUPT PROCEDURES

The following is intended as a brief overview of the DPS 8000 interrupt procedures.

System Controller Interrupts

The SCU has an interrupt mask register and eight interrupt level queues. There are eight mask bits, one bit for each interrupt level, plus one "all" mask bit. The SCU maintains a queue for each interrupt level. The queue lengths are fixed at 256 entries per level. The SCU "senses" the interrupt level field of the received interrupt words to determine which queue to use and places the interrupt words in the selected queue. Interrupt words are normally sent by the IMX upon completion of an I/O service. A CPU can also initiate an interrupt.

The queueing scheme used by the SCU is based on a first-in first-out rule at each interrupt level. The SCU processes the queue in response to the Read Interrupt Word (RIW) instruction. Interrupt level queue zero has the highest priority and seven the lowest.

The SCU sends an interrupt to all CPUs that are unmasked when there are entries in the queue. The SCU fetches one queue entry per RIW request, starting with the oldest entry of the highest priority interrupt level that is not masked.

When GCOS issues the RIW command it obtains the interrupt queue words. The CPU receives each 2-word queue entry in the A and Q registers. With each RIW, GCOS tests the CPU's A and Q registers to determine whether all unmasked interrupt queue entries have been serviced.

Inward CLIMB Interrupts

An entry descriptoris "wired-in" to support the ICLIMB instruction for interupts. The second word of this ICLIMB instruction has the following parameters:

E bit - (no parameters)

C field

bit 18 - 0 (index register 0 is not changed)

bit 19 - Ignored. The Master mode bit of the indicator register is set ON but no descriptors are prepared.

bit 20 - Unused

bit 21 - Ignored

bit 22-23 - 0 (ICLIMB version)

S,D fields - Ignored. If an entry descriptor is not found at a fixed memory location, the processor generates a Backup fault.

(Refer to the CLIMB instruction format in Section 8.)

If an entry descriptor is not found at the fixed interrrupt vector location or if another fault occurs (e.g., a parity error) while the processor is attempting to CLIMB to the interrupt handler, the processor attempts to obtain an entry descriptor from the Backup fault vector location. If this second location does not contain an entry descriptor, the processor enters the HALT state. If the second fault occurs prior to the transfer of control to the new domain at the end of the ICLIMB, then the safe store frame will overlay the original frame (with the same information except for fault code). If the second fault occurs during the transfer of domains, such as a page fault when obtaining the next instruction, then a second frame will be filled specifying the new domain and the fault code of the type of fault that caused the backup condition.

The processor is placed in the Privileged Master mode for the execution of the "wired-in" ICLIMB instruction. Upon exiting the ICLIMB instruction, the processor will remain in the Privileged Master mode if flag bit 26 of the new instruction segment register (ISR) equals 1. If flag bit 26 of the new ISR equals 0, the processor will cycle to Master mode.

Multiword Instruction Interrupts

If an interrupt occurs during a multiword instruction, the processor sets bit 30 of the indicator register to 1. If the entry descriptor is type T = 11, the pointer and length registers are saved in the safe store frame. Indicator register bit 30 is reset to zero (OFF), but is safe stored as a 1 (ON) in word 4.

Eight 36-bit registers are used to store and load pointers for sending and receiving addresses and field lengths, and for other control information when a multiword instruction is interrupted.

IC VALUES STORED ON FAULTS AND INTERRUPTS

If the safe store bypass flag in the option register equals 0, a safe store is executed for any fault or interrupt. A description of the safe store stack is given in Figures 8-7 and 8-8.

The instruction is stored in word 2. Words 0,1 are defined as illustrated. In word 5, bit 8 is not used, but bits 17-18 contain 00. Word 47 is used for the timer register; word 5, bit 0 is for FRTRY; and words 48-51 contain mid-instruction interrupt recovery data for firmware.

The classes of faults and interrupts found in the safe store stack frame following a fault or interupt are described in Table 6-3. The designation of the fault group priorities is given in Table 6-1.

Table 6-3. Classes Of Faults And Interrupts (DPS 80)

SAFE STORE DATA	FAULT GROUP I	FAULT GROUP II - V						INTERRUPT		PROGRAMMED CLIMB
		FAULT 1 ALL OTHERS NOT IN 2-6	FAULT 2 DURING EIS	FAULT 3 DURING TRANSFER	FAULT 4 DURING TRANSFER IN CLIMB	FAULT 5 DURING CLIMB	FAULT 6 IN-LINE INSTR. FETCH	INTER. 1 NOT DURING EIS	INTER. 2 DURING EIS	
WORDS 0-3	INFORMATION REQUIRED BY PROCESSOR FOR RESTART AFTER FAULTS							N/A		
IC_{00-17} WORD 4	UNDEFINED	IC OF FAULTING INSTRUCTION			IC OF "TRANSFERRED TO" INSTR.	IC OF FAULTING INSTRUCTION		IC OF LAST COMPLETED INSTR. + 1	IC OF EIS INSTR.	IC OF CLIMB INSTR. + 2
IR_{30} WORD 4	1 OR 0	0	1	0					1	0
SEGID (IS) WORD 5	CURRENT IS				IS OF NEW INSTR.	IS PRIOR TO CLIMB	CURRENT IS			
DSAR, EWSN RVA WORDS 6-7	LAST VALUE OF DSAR: EWSN AND RVA CORRESPOND TO LAST SEGMENT ACCESSED									
ISR WORDS 8-9	CURRENT				ISR OF NEW DOMAIN	ISR PRIOR TO CLIMB	CURRENT			ISR PRIOR TO CLIMB
ASR WORDS 10-11 LSR WORDS 12-13 PSR WORDS 14-15	CURRENT				OF NEW DOMAIN	PRIOR TO CLIMB	CURRENT			PRIOR TO CLIMB
REGISTERS WORDS 16-47	LAST VALUE OF REGISTERS									
SAFE STORE OF P L WORDS 48-49	IF ENTRY DESCRIPTOR T=11									
EVEN INSTR IS FAULTING INSTR. IF SAFE STORED IC IS	UNDEFINED	$IC_{17}=0$					N/A	$IC_{17}=0$		IF $IC_{17}=0$ CLIMB WAS EVEN

NOTE: In general, DPS 80 will not change any register values on a faulting Instruction (including TSS or RET). The one execption is a fault occurring on a transfer at the end of the CLIMB. In this case, the Safestore data will reflect the new domain.

The definition of the classes of faults and interrupts contained in Table 6-3 follows:

FAULT 1 - A group II to V fault not covered by FAULT 2 through FAULT 6, including XECs and RPTs[1]. For XECs and RPTs, if a fault occurs on the "to" instruction, the faulting instruction is the XEC or RPT instruction

FAULT 2 - A group II to V fault caused by a multiword instruction

FAULT 3 - A group II to V fault that occurs while attempting to fetch "transferred to" instructions resulting from a TRA, TSXn, TSS, RET, or a satisfied conditional transfer

FAULT 4 - A group II to V fault that occurs while attempting to fetch "transferred to" instructions resulting from a CLIMB instruction

FAULT 5 - A group II to V fault that occurs on a CLIMB instruction prior to fetching "transferred to" instructions

FAULT 6 - A group II to V fault that occurs on an inline instruction fetch

INTER 1 - An interrupt that occurs any time except during an interruptible multiword instruction

INTER 2 - An interrupt that occurs during an interruptible multiword instruction

The effective working space number (EWSN) and relative virtual address (RVA) are not valid for MME and DRL instructions for faults and interrupts that are not generated by the virtual memory hardware, since the EWSN and RVA always reflect the last segment accessed and the last indirect word for the fault tag. If the virtual memory hardware detects the fault, the EWSN and RVA will reflect the faulting segment that is referenced.

The instruction counter (IC) values stored in bits 0-17 of word 4 of the safe store stack during faults and interrupts are described below:

1. Programmed CLIMB

   IC of CLIMB + 2

2. Interrupt during multiword instruction or Connect, or Timer Runout faults during multiword instruction

   IC of the first word of the multiword instruction

3. Interrupt after completed multiword or single-word instruction

   IC of the next instruction

---

1. RPT, RPD, RPL execute in NS mode only.

4. Fault while attempting to fetch "transferred to" instructions resulting from a CLIMB instruction

   IC of "transferred to" instruction

5. Safestore stack fault on programmed CLIMB

   IC of "transferred to" instruction

6. Execute fault

   IC undefined

7. Operation Not Completed, Lockup, or Bound faults

   IC of faulting instruction + 1

8. Connect or Timer Runout faults after completed multiword or single-word instruction

   IC of next instruction

9. Any other fault

   IC of faulting instruction + 1

SECTION 7

MACHINE INSTRUCTION FUNCTIONS

Many of the instructions available in the instruction repertoire are familiar to experienced users of large-scale computers. However, additional instructions have been provided to supply extended capability for character handling, decision making, and advanced programming techniques involving list processing. In addition, numerous instructions are provided that have capabilities for processing and moving bytes, BCD characters, packed decimal data, and bit strings, for vector operations, and for performing register to register operations.

SINGLE-WORD INSTRUCTIONS

Single-word instructions provide for multiple variations by permitting the user to specify not only the type of address modification desired, but also the source and/or destination registers associated with particular operation codes. For example, the operation field for a Transfer and Set Index Register n (TSXn) instruction specifies the index in the operation field, leaving full address modification capability free for destination calculation.

The processor performs efficient operations on 6-, 9-, 18-, 36-, and 72-bit operands.

The following operations are performed by single-word instructions:

- Address Register Instructions
- Boolean Operations
- Comparison Operations
- Data Movement Instructions
- Data Shifting Instructions
- Effective Address to Register Instructions
- Fixed-Point Arithmetic Instructions
- Floating-Point Arithmetic Instructions
- Quadruple-Precision Instructions
- Master Mode Instructions
- Miscellaneous Instructions
- ES Mode Instructions
- Special Processor Instructions
- Transfer Instructions

Address Register Instructions

Address register instructions allow for loading and storing of address registers. The number of bits loaded or stored depends upon whether the NS or ES mode is being used. Alter address register instructions are used to replace, increment, and decrement the content of the address register in word, character, or bit. These instructions perform operations between registers; they do not refer to memory. Special address register instructions, executable only in the NS mode, use the address registers to manipulate the address portion of numeric and alphanumeric operand descriptors. (Refer to the instructions specifications in Section 8).

Boolean Operations

The logical operations AND, OR, and EXCLUSIVE OR are permitted between storage and the index registers, A- and Q-registers, and the AQ-register.

Comparison Operations

Comparison operations do not alter the contents of storage or the specified register, but merely set or clear the appropriate indicators as the result dictates. The compare instructions enable the user to make many types of program decisions.

Fixed-point compare instructions permit comparison of absolute values, (algebraic or characters); provide for tests of word fields; permit searches for identical, selectable word fields; and permit searches for a value within selectable limits.

Floating-point compare instructions are included for single- and double-precision operations on absolute values and algebraic values. All compare instructions are repeatable using the RPT, RPD, or RPL instructions. (Repeat instructions execute in NS mode only.)

Data Movement Instructions

Character handling and manipulation are facilitated by "indirect and tally" (IT) address modification and by instructions for directly storing selected characters of the accumulator or quotient register. Instructions are also included for directly loading the index registers from either memory or the A- and Q-registers, directly storing any register into memory, and loading registers with the two's complement (negative) of the contents of the memory location specified.

Data Shifting Instructions

Shifting is accomplished using an algorithm in which long shifts are executed essentially as fast as short shifts. The A- and Q-registers can be shifted individually or as one unit. The shift commands include right- or left-shift arithmetic, right-shift logical, and left-shift rotate, (right-shift rotate is omitted because the high speed of the left-shift rotate makes the right-shift rotate unnecessary).

Effective Address To Register Instructions

The Effective Address to Register instructions permit the effective address of such an instruction to be placed in any of the index registers, in the A-register, or in the Q-register. Thus, any effective address referenced frequently in a program can be stored in a register and used without lost processing time in repeatedly redeveloping the effective address. Furthermore, the instructions provide the user with the capability of transferring data among any of the index registers and to the A-register and the Q-register.

Fixed-Point Arithmetic Instructions

Instructions for both fractional and integral multiplication and division afford the programmer freedom from scaling the results of such operations. Fractional multiplications are performed with the multiplicand in the A-register; the result appears in bit positions 0 through 70 of the AQ-register, automatically scaled with the binary point to the right of position 0. Integral multiplications are performed with the multiplicand in the Q-register; the result appears in bit positions 1 through 71 of the AQ-register, automatically scaled with the binary point to the right of position 71.

Fractional divisions use the full range of the AQ-register for the dividend; the quotient appears in the A-register with the remainder in the Q-register. The binary point is automatically scaled to the right of position 0. Integral divisions have the dividend in the Q-register, with the binary point to the right of position 35. After division, the quotient is in the Q-register with the binary point automatically placed to the right of position 35; the remainder is in the A-register.

Normally, integer operations of divide and multiply occur in the Q-register, and fractional operations of divide and multiply occur in the A-register. This convention permits easy programming of fixed-point arithmetic operations.

Instructions are provided for combining the contents of memory locations directly with the contents of registers and storing the results in the same locations, without recourse to separate store instructions. In all such cases, the programmer can use the 18-bit indexing registers, X0 through X7 in the NS mode, the 36-bit general indexing registers, GX0 through GX7 in the ES mode, and the 36-bit A- and Q-registers. In effect, the Add and Subtract to Storage instructions make arithmetic accumulators of all available memory locations. In all such cases, the register contents are undisturbed.

Floating-Point Arithmetic Instructions

Floating-point operations can be performed on both single- and double-precision data words; complete sets of data movement, arithmetic, and control instructions are provided for use in both types of operations. Unless otherwise specified by the programmer, the mantissas of all floating-point operation results, except divides, are automatically normalized by the hardware. In additions and subtractions, the operands are automatically aligned.

Operations on floating-point numbers are performed using an extended register composed of a 72-bit AQ-register, which holds the mantissa, and a separate 8-bit exponent register; operations on the exponent and mantissa are performed by two separate adders. The existence of separate exponent and mantissa registers and adders enables the programmer to efficiently intermix single- and double-precision instructions.

The floating-point instruction repertoire includes two special divide instructions: Floating Divide Inverted (FDI) and Double-Precision Floating Divide Inverted (DFDI). These instructions cause the contents of the memory location to be divided by the contents of the AQ-registers, the reciprocal of other divide instructions in the repertoire. Thus, regardless of whether the contents of the AQ-register must be a dividend or a divisor, the programmer can always perform a division without recourse to wasteful data movement operations.

Floating Negate, Normalize, Add to Exponent, and Single- and Double-Precision Compare instructions further facilitate effective programming.

The slave mode instructions providing rounded floating-point results include: FRD, DFRD, FSTR, and DFSTR.

The hexadecimal option may be used in floating-point operations to declare hexadecimal constants, either explicitly or by default. (Refer to Hexadecimal Floating-point Number in Section 2.)

Quadruple-Precision Floating-Point Instructions

Quadruple-precision floating-point instructions provide arithmetic operations for which the exponents are handled as powers of 16. In these operations, the AQ register and the operand register (LOR) handle mantissas and the E register handles exponents. Results of these operations are automatically normalized.

Privileged Master Mode Instructions

The following conditions must be satisfied for execution of these instructions.

- The Master Mode bit in the Indicator Register is ON.
- The privileged bit in the Instruction Segment Register (ISR) is ON.
- The housekeeping bit in the page table word for the instruction is ON. This bit is assumed as being ON in the Working Space 0 Addressing mode.

When these conditions are not met a Command fault or a Class 1 Security fault occurs. (Refer to the instruction specifications in Section 8.)

Miscellaneous Instructions

This catagory includes instructions which perform operations such as Binary-to-BCD conversion, programmed faults, repeat instructions, and no-operation instructions (e.g., NOP).

Special Processor Instructions

Slave mode instructions available to provide the operating system with program gating for multiprocessor configurations include: LDAC, LDQC, and SZNC. They provide for clearing the referenced memory cell to zero after the contents are transferred to the processor. The instruction STACQ provides for conditional storing in the referenced memory cell, based on the comparison of Q with the operand word.

Privileged master mode instructions providing system information and control are LCPR, SCPR, RSCR, SSCR, STTA, and STTD.

MULTIWORD INSTRUCTIONS

Multiword instructions fall into six general categories:

- Alphanumeric instructions
- Numeric instructions
- Bit string instructions
- Conversion instructions
- Edited Move Instructions

Alphanumeric Instructions

Alphanumeric instructions permit moving, transliteration, editing, and comparing of alphanumeric data. The operands for these instructions (with the exception of comparisons) can be any combination of alphanumeric types (9-bit, 6-bit, or 4-bit) and are translated as part of the instruction execution to permit the different types of character strings to be manipulated in the same instruction.

Numeric Instructions

Numeric instructions include decimal arithmetic functions in addition to moving, comparing, and editing of numeric data. Decimal add, subtract, multiply, and divide operations are permitted. The numeric instructions can be 2- or 3-operand instructions. The operands themselves can be either 9-bit or 4-bit packed decimal. The numbers employed as data can be floating-point with leading sign, scaled fixed-point with trailing sign, leading sign, or no sign. As with alphanumeric instructions, numeric instructions achieve these various characteristics within a single multiword instruction (in conjunction with associated operand descriptors).

Bit String Instructions

Bit string instructions allow two bit strings to be compared on a bit-by-bit basis and Boolean operations to be performed to combine strings and set indicators.

Conversion Instructions

Conversion instructions provide for decimal/binary and binary/decimal conversion.

Edited Move Instructions

Both alphanumeric and numeric edited move instructions (MVE, MVNE, and MVNEX) utilize micro operations (MOPS) to perform editing functions. The sequence of micro-steps to be executed is contained in memory and is referenced by the second operand descriptor of the edited move instructions.

Micro operations provide alphanumeric and numeric edited move instructions with the capability to edit strings on a character-by-character or digit-by-digit basis, or in concatenated series of characters and digits.

Micro operations are not altered by their execution; therefore, a sequence of micro operations can be set to describe a data field and then can be used repeatedly by the edit instructions. A single instruction can perform a complicated edit function with great speed.

The special edit characters are contained in a hardware edit table and table entries are modified using micro operations designed for this purpose. Refer to "Micro Operations For Edit Instructions MVE, MVNE, and MVNEX" later in this section for detailed information.

Multiword Instruction Capabilities

The capabilities of the multiword instructions are given below.

1. Decimal Arithmetic Capability

   a. Data types as packed decimal and direct ASCII (may be intermixed)

   b. Decimal arithmetic operands of 1 to 63 digits in length (including sign)

   c. Numeric data as fixed-point and/or floating-point (intermixed fixed- and floating-point data is allowed)

   d. A full set of decimal arithmetic instructions (each is a multiword instruction with either two or three descriptor words) including add, subtract, multiply, and divide

   e. All numeric instructions with a hardware rounding option

2. Data Manipulation Capability

   Five native data modes - ASCII, BCD, packed decimal (numeric only), bit string, and EBCDIC

3. Data Movement Capability

   a. Alphanumeric movement from left or right with character-fill

   b. Character moves from 9-bit-byte or 8-bit-byte fields

   c. Numeric move with fill and/or rounding and scale change

   d. Bit string manipulation using any of 16 different Boolean operations

   e. Radix conversion and transliteration instructions

4. Data Comparison Capability

   a. Alphanumeric comparison with fill

   b. Numeric comparisons between fields of the same or different format and character type

c. Bit string comparisons with fill

d. String scan for a match of one or two characters

5. Second-Level Indexing Capability

   Eight address registers providing for second-level indexing for all instructions (including single-word instructions)

ADDRESS REGISTER INSTRUCTIONS

ADDRESS REGISTER INSTRUCTIONS

This set of instructions provides the capability for using address registers to manipulate the address portion of numeric and alphanumeric descriptors. If an address register is to be used in address preparation, its usage is specified in the instruction word. All single-word instructions, to which address modification is applicable, have essentially the same machine instruction word format which hardware interprets differently depending on whether the processor is in the NS or the ES mode. (Refer to Section 5.)

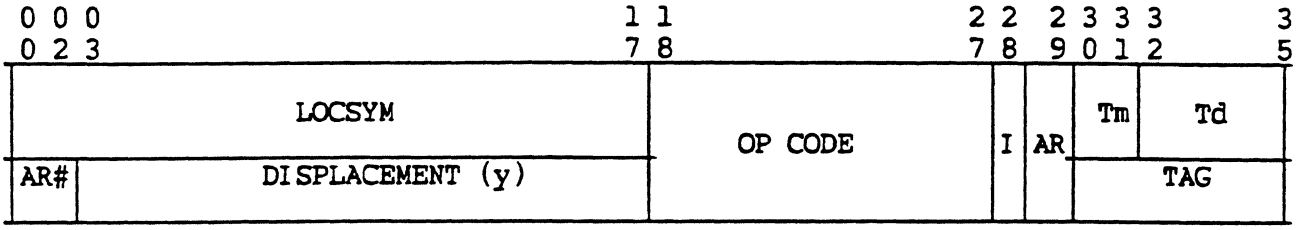


Figure 7-1. Single-word Instruction With Address Modification

AR# - One of eight address registers (0-7)

LOCSYM - Represents either address of operand or displacement from a base

DISPLACEMENT - (y) 15-bit displacement from the address register address (two's complement: values from -16,384 to +16,383)

OP CODE - A 10-bit operation code field

I - Program interrupt inhibit bit

AR - If bit 29 is 1, an address register is to be used and is specified by bits 0, 1, and 2 of the y field. If bit 29 is 0, no address register is used.

TAG - Tag field that controls all other address modification. If an address register is used on an instruction with indirect addressing, it is applied only on the fetch of the indirect word.

Tm - tag modifier
Td - tag designator

ADDRESS REGISTER INSTRUCTIONS

Address Register Load

LARn	76n (1)	Load Address Register n
LAREG	463 (1)	Load Address Registers

Address Register Store

SARn	74n (1)	Store Address Register n
SAREG	443 (1)	Store Address Registers

Alter Address Register Contents

This set of instructions provides the capability for replacing, incrementing, and decrementing the contents of an address register on either a word, character, or bit address basis. The operation is register-to-register, with no memory fetch involved.

The special instructions have the same instruction format:

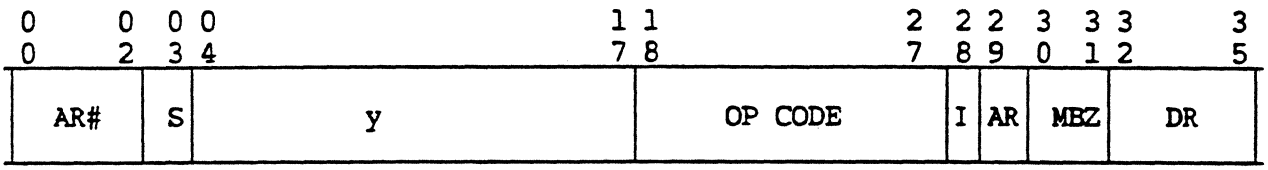


Figure 7-2. Alter Address Register Contents

AR# - Selects address register to be altered.

S - Sign bit. (Refer to Section 5 for differences betwen NS and ES modes.)

y - A word displacement (no character or bit position included) used along with the contents specified in the DR field to alter the contents of the specified address register. Bit 3 provides negative (two's complement) or positive word displacement.

OP CODE - 10-bit operation code field.

I - Program interrupt inhibit bit.

AR - Address register bit.

ADDRESS REGISTER INSTRUCTIONS

If bit 29 = 1, the sum of the DR (in characters, words, or bits) and the y field (in words) are added to or subtracted from the contents of the AR specified in bits 0-2.

If bit 29 = 0, the above described sum or its two's complement is loaded into the AR for addition or subtraction, respectively.

If the mnemonic is coded with X (for example, AWDX), bit 29 is forced to zero.

MBZ - Bits 30-31 must be zero.

DR - Displacement register. Specifies which register contains the displacement value. The register codes and register lengths are the same as those used in MF fields except that IC modification is illegal. (Refer to Table 5-2.) (Refer to "Multiword Modification Field" in this section.).

The operations for adding a value to the contents of an address register proceed as with effective operand address preparation from an operand descriptor, with the final results being stored in the specified address register.

The subtract operation differs only in that the contents of the register specified by the code in the DR field are first added to the y field. This result is then subtracted from the actual contents of the address register or from the implied zero contents and the result is placed in the address register. The codes for DU, DL, and IC are illegal for the DR field and cause an IPR fault.

The indicators are unaffected by these instructions.

A4BD(X)	502 (1)	Add 4-Bit Displacement to Address Register
A6BD(X)	501 (1)	Add 6-Bit Displacement to Address Register
A9BD(X)	500 (1)	Add 9-Bit Displacement to Address Register
ABD(X)	503 (1)	Add Bit Displacement to Address Register
AWD(X)	507 (1)	Add Word Displacement to Address Register
S4BD(X)	522 (1)	Subtract 4-Bit Displacement from Address Register
S6BD(X)	521 (1)	Subtract 6-Bit Displacement from Address Register
S9BD(X)	520 (1)	Subtract 9-Bit Displacement from Address Register
SBD(X)	523 (1)	Subtract Bit Displacement from Address Register
SWD(X)	527 (1)	Subtract Word Displacement from Address Register

ADDRESS REGISTER INSTRUCTIONS

Special Address Register Instructions

Special instructions provide use of address registers to manipulate the address portion of numeric and alphanumeric operand descriptors. These instructions may be used only in the NS mode. If an attempt is made to execute these instructions in the ES mode, an IPR fault occurs.

These special instructions have the following instruction format:

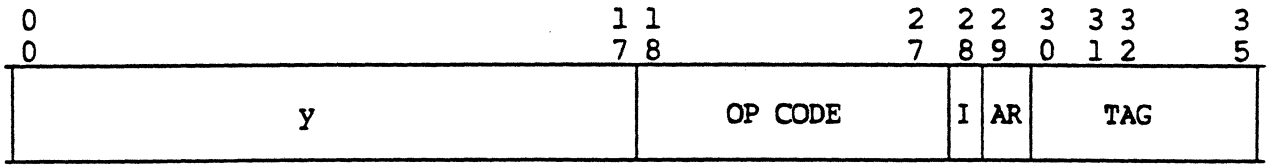


Figure 7-3. Special Address Register Instructions

AARn	56n (1)	Alphanumeric Descriptor to ARn
ARAn	54n (1)	ARn to Alphanumeric Descriptor
ARNn	64n (1)	ARn to Numeric Descriptor
NARn	66n (1)	Numeric Descriptor to ARn

BOOLEAN OPERATIONS

BOOLEAN OPERATION INSTRUCTIONS

The logical operations AND, OR, and EXCLUSIVE OR are permitted between storage and the index registers, A- and Q-registers, and the AQ-register. These instructions use the single-word instruction format.

Boolean Expressions

A Boolean expression is defined similarly to an algebraic expression except that the operators *, /, +, and - are interpreted as Boolean operators. Two types of boolean expressions are defined below:

1. The expression that appears in the variable field of a BOOL pseudo-operation uses Boolean operators.

2. The expression that appears in the octal subfield of the variable field of a VFD pseudo-operation uses Boolean operators.

Evaluation Of Boolean Expressions

A Boolean expression is evaluated following the same procedure used for an algebraic expression except that the operators are interpreted as Boolean.

In a Boolean expression, the operators +, -, *, and / have Boolean meanings, rather than their normal arithmetic meanings, as follows:

Operator	Meaning	Definition
+	OR, inclusive OR, union	0 + 0 = 0 0 + 1 = 1 1 + 0 = 1 1 + 1 = 1
-	EXCLUSIVE OR symmetric difference	0 - 0 = 0 0 - 1 = 1 1 - 0 = 1 1 - 1 = 0

BOOLEAN OPERATIONS

Operator	Meaning	Definition
*	AND, intersection	0 * 0 = 0 0 * 1 = 0 1 * 0 = 0 1 * 1 = 1
/	one's complement, complement, NOT	/0 = 1 /1 = 0

Although / is a unary operation involving only one term, by convention A/B is taken to mean A*/B. This is not regarded as an error by the assembler. Thus, the table for / as a two-term operation is:

```
0/0 = 0
0/1 = 0
1/0 = 1
1/1 = 0
```

and other conventions are:

```
+A = A+  = A
-A = A-  = A
*A = A*  = 0     (possible error, operand missing)
A/ = A/0 = A
```

Boolean AND

ANA	375 (0)	AND to A-Register
ANAQ	377 (0)	AND to AQ-Register
ANQ	376 (0)	AND to Q-Register
ANSA	355 (0)	AND to Storage from A-Register
ANSQ	356 (0)	AND to Storage from Q-Register
ANSXn	34n (0)	AND to Storage from Index Register n
ANXn	36n (0)	AND to Index Register n

BOOLEAN OPERATIONS

Boolean OR

ORA	275 (0)	OR to A-Register
ORAQ	277 (0)	OR to AQ-Register
ORQ	276 (0)	OR to Q-Register
ORSA	255 (0)	OR to Storage from A-Register
ORSQ	256 (0)	OR to Storage from Q-Register
ORSXn	24n (0)	OR to Storage from Index Register n
ORXn	26n (0)	OR to Index Register n

Boolean EXCLUSIVE OR

ERA	675 (0)	EXCLUSIVE OR to A-Register
ERAQ	677 (0)	EXCLUSIVE OR to AQ-Register
ERQ	676 (0)	EXCLUSIVE OR to Q-Register
ERSA	655 (0)	EXCLUSIVE OR to Storage with A-Register
ERSQ	656 (0)	EXCLUSIVE OR to Storage with Q-Register
ERSXn	64n (0)	EXCLUSIVE OR to Storage with Index Register n
ERXn	66n (0)	EXCLUSIVE OR to Index Register n

Boolean COMPARATIVE AND

CANA	315 (0)	Comparative AND with A-Register
CANAQ	317 (0)	Comparative AND with AQ-Register
CANQ	316 (0)	Comparative AND with Q-Register
CANXn	30n (0)	Comparative AND with Index Register n

Boolean COMPARATIVE NOT AND

CNAA	215 (0)	Comparative NOT AND with A-Register
CNAAQ	217 (0)	Comparative NOT AND with AQ-Register
CNAQ	216 (0)	Comparative NOT AND with Q-Register
CNAXn	20n (0)	Comparative NOT AND with Index Register n

FIXED POINT INSTRUCTIONS

FIXED-POINT INSTRUCTIONS

Data Movement Load

EAA	635 (0)	Effective Address to A-Register
EAQ	636 (0)	Effective Address to Q-Register
EAXn	62n (0)	Effective Address to Index Register n
LCA	335 (0)	Load Complement into A-Register
LCAQ	337 (0)	Load Complement into AQ-Register
LCQ	336 (0)	Load Complement into Q-Register
LCXn	32n (0)	Load Complement into Index Register n
LDA	235 (0)	Load A-Register
LDAC	034 (0)	Load A-Register and Clear
LDAQ	237 (0)	Load AQ-Register
LDI	634 (0)	Load Indicator Register
LDQ	236 (0)	Load Q-Register
LDQC	032 (0)	Load Q-Register and Clear
LDXn	22n (0)	Load Index Register n from Upper
LREG	073 (0)	Load Registers
LXLn	72n (0)	Load Index Register n from Lower

Data Movement Store

SBAR	550 (0)	Store Base Address Register
SREG	753 (0)	Store Registers
STA	755 (0)	Store A-Register
STAC	354(0)	Store A Conditional
STACQ	654 (0)	Store A Conditional on Q
STAQ	757 (0)	Store AQ-Register
STBA	551 (0)	Store 9-bit Bytes of A-Register
STBQ	552 (0)	Store 9-bit Bytes of Q-Register
STC1	554 (0)	Store Instruction Counter Plus 1
STC2	750 (0)	Store Instruction Counter Plus 2
STCA	751 (0)	Store 6-bit Characters of A-Register
STCQ	752 (0)	Store 6-bit Characters of Q-Register
STI	754 (0)	Store Indicator Register
STQ	756 (0)	Store Q-Register
STT	454 (0)	Store Timer Register
STXn	74n (0)	Store Index Register n in Upper
STZ	450 (0)	Store Zero
SXLn	44n (0)	Store Index Register n in Lower

FIXED-POINT INSTRUCTIONS

Data Movement Shift

ALR	775 (0)	A-Register Left Rotate
ALS	735 (0)	A-Register Left Shift
ARL	771 (0)	A-Register Right Logical Shift
ARS	731 (0)	A-Register Right Shift
LLR	777 (0)	Long Left Rotate
LLS	737 (0)	Long Left Shift
LRL	773 (0)	Long Right Logical Shift
LRS	733 (0)	Long Right Shift
QLR	776 (0)	Q-Register Left Rotate
QLS	736 (0)	Q-Register Left Shift
QRL	772 (0)	Q-Register Right Logical Shift
QRS	732 (0)	Q-Register Right Shift

Fixed-Point Addition

ADA	075 (0)	Add to A-Register
ADAQ	077 (0)	Add to AQ-Register
ADL	033 (0)	Add Low to AQ-Register
ADLA	035 (0)	Add Logical to A-Register
ADLAQ	037 (0)	Add Logical to AQ-Register
ADLQ	036 (0)	Add Logical to Q-Register
ADLXn	02n (0)	Add Logical to Index Register n
ADQ	076 (0)	Add to Q-Register
ADXn	06n (0)	Add to Index Register n
AOS	054 (0)	Add 1 to Storage
ASA	055 (0)	Add to Storage from A-Register
ASQ	056 (0)	Add to Storage from Q-Register
ASXn	04n (0)	Add to Storage from Index Register n
AWCA	071 (0)	Add With Carry to A-Register
AWCQ	072 (0)	Add With Carry to Q-Register

FIXED-POINT INSTRUCTIONS

Fixed-Point Subtraction

SBA	175 (0)	Subtract from A-Register
SBAQ	177 (0)	Subtract from AQ-Register
SBLA	135 (0)	Subtract Logical from A-Register
SBLAQ	137 (0)	Subtract Logical from AQ-Register
SBLQ	136 (0)	Subtract Logical from Q-Register
SBLXn	12n (0)	Subtract Logical from Index Register n
SBQ	176 (0)	Subtract from Q-Register
SBXn	16n (0)	Subtract from Index Register n
SSA	155 (0)	Subtract Stored from A-Register
SSQ	156 (0)	Subtract Stored from Q-Register
SSXn	14n (0)	Subtract Stored from Index Register n
SWCA	171 (0)	Subtract With Carry from A-Register
SWCQ	172 (0)	Subtract With Carry from Q-Register

Fixed-Point Multiplication

MPF	401 (0)	Multiply Fraction
MPY	402 (0)	Multiply Integer

Fixed-Point Division

DIV	506 (0)	Divide Integer
DVF	507 (0)	Divide Fraction

FIXED POINT INSTRUCTIONS

Fixed-Point Comparison

Fixed-point compare instructions permit comparison of absolute values, algebraic values, or characters; provide for test of word fields; permit searches for identical, selectable word fields; and permit searches for a value within selectable limits. Comparison instructions are repeatable using the RPT, RPD, or RPL instruction. (Repeat instructions are executable in NS mode only.)

CMG	405 (0)	Compare Magnitude
CMK	211 (0)	Compare Masked
CMPA	115 (0)	Compare with A-Register
CMPAQ	117 (0)	Compare with AQ-Register
CMPQ	116 (0)	Compare with Q-Register
CMPXn	10n (0)	Compare with Index Register n
CWL	111 (0)	Compare with Limits
SZN	234 (0)	Set Zero and Negative Indicators from Storage
SZNC	214 (0)	Set Zero and Negative Indicators from Storage and Clear

Fixed-Point Negate

NEG	531 (0)	Negate (A-Register)
NEGL	533 (0)	Negate Long (AQ-Register)

FLOATING POINT INSTRUCTIONS

FLOATING-POINT INSTRUCTIONS

Data Movement Load

DFLD	433 (0)	Double-Precision Floating Load
DFLP	532 (0)	Double-Precision Floating Load Positive
FLD	431 (0)	Floating Load
FLP	530 (0)	Floating Load Positive
LDE	411 (0)	Load Exponent Register

Data Movement Store

DFST	457 (0)	Double-Precision Floating Store
DFSTR	472 (0)	Double-Precision Floating Store Rounded
FST	455 (0)	Floating Store
FSTR	470 (0)	Floating Store Rounded
STE	456 (0)	Store Exponent Register

Floating-Point Addition

ADE	415 (0)	Add to Exponent Register
DFAD	477 (0)	Double-Precision Floating Add (Normalized)
DUFA	437 (0)	Double-Precision Floating Add (Unnormalized)
FAD	475 (0)	Floating Add (Normalized)
UFA	435 (0)	Floating Add (Unnormalized)

Floating-Point Subtraction

DFSB	577 (0)	Double-Precision Floating Subtract
DFSBI	467 (0)	Double-Precision Floating Subtract Inverted
DUFS	537 (0)	Double-Precision Unnormalized Floating Subtract
FSB	575 (0)	Floating Subtract
FSBI	465 (0)	Floating Subtract Inverted
UFS	535 (0)	Unnormalized Floating Subtract
UFTR	434 (0)	Unnormalized Floating Truncate Fraction

FLOATING-POINT INSTRUCTIONS

Floating-Point Multiplication

DFMP	463 (0)	Double-Precision Floating Multiply
DUFM	423 (0)	Double-Precision Unnormalized Floating Multiply
FMP	461 (0)	Floating Multiply
UFM	421 (0)	Unnormalized Floating Multiply

Floating-Point Division

DFDI	527 (0)	Double-Precision Floating Divide Inverted
DFDV	567 (0)	Double-Precision Floating Divide
FDI	525 (0)	Floating Divide Inverted
FDV	565 (0)	Floating Divide

Floating-Point Comparison

Floating-point compare instructions are used for single- and double-precision operations on absolute values and algebraic values. Compare instructions are repeatable using the RPT, RPD, or RPL instruction.

DFCMG	427 (0)	Double-Precision Floating Compare Magnitude
DFCMP	517 (0)	Double-Precision Floating Compare
FCMG	425 (0)	Floating Compare Magnitude
FCMP	515 (0)	Floating Compare
FSZN	430 (0)	Floating Set Zero and Negative Indicators from Storage

Floating-Point Negate

FNEG	513 (0)	Floating Negate

Floating-Point Normalize

FNO	573 (0)	Floating Normalize

Floating-Point Round

DFRD	473 (0)	Double-Precision Floating Round
FRD	471 (0)	Floating Round

Floating-Point Truncate Fraction

FTR	474 (0)	Floating Truncate Fraction

QUADRUPLE-PRECISION INSTRUCTIONS

QUADRUPLE-PRECISION INSTRUCTIONS

The quadruple-precision instructions permit exponents to be handled as powers of 16. The AQ register and LOR register handle the mantissas and the E register handles the exponents. The results of these operations are automatically normalized.

QFAD	476 (0)	Quadruple-Precision Floating Add
QFLD	432 (0)	Quadruple-Precision Floating Load
QFMP	462 (0)	Quadruple-Precision Floading Multiply
QFSB	576 (0)	Quadruple-Precision Floating Subtract
QFST	453 (0)	Quadruple-Precision Floating Store
QFSTR	466 (0)	Quadruple-Precision Floating Store Rounded
QSMP	460 (0)	Quadruple-Precision Floating Multiply with Double-Precision Operands

MULTIWORD INSTRUCTIONS

MULTIWORD INSTRUCTIONS

The format and terms which are common to all multiword instructions are described below.

Multiword Instruction Format

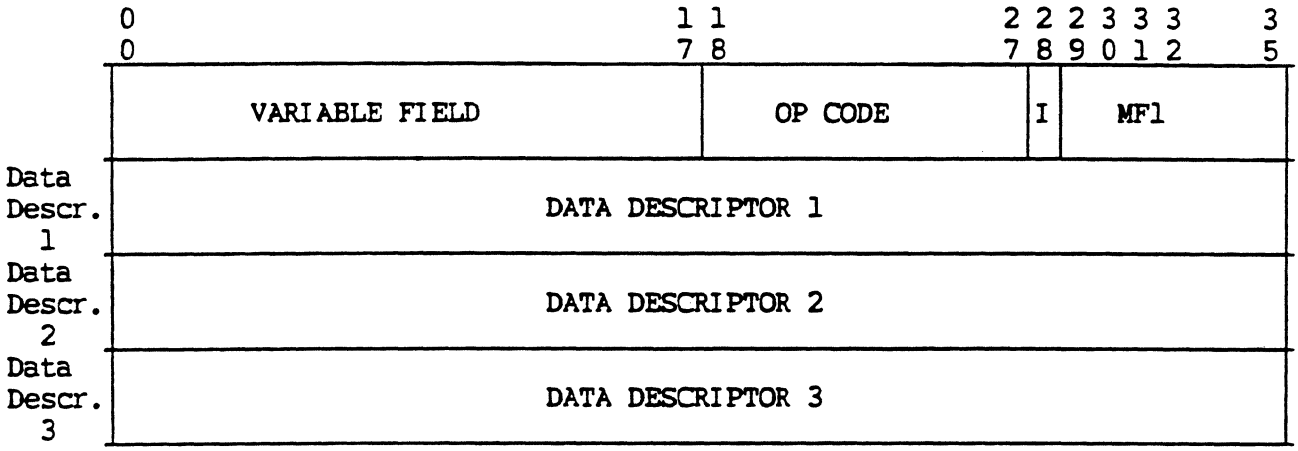


Figure 7-4. Multiword Instruction Format

Bits	Description
0-17	Contains variable information for the executed instruction function. The format of this field differs with each instruction. When data descriptors 2 and 3 exist, the corresponding MF2 and MF3 are located in bits 11-17 and 1-8, respectively, of the variable field to describe the address modification executed for the data descriptors. Refer to the individual instruction specifications in Section 8.
18-27	10-bit operation code
28	Interrupt inhibit bit
29-35	Modification field 1. Describes the address modification executed for data descriptor 1.

Data descriptors (2 or 3) follow the basic instruction word. The number of data descriptors is determined by each instruction. Data descriptors consist of the operand descriptor or the indirect word which points to the operand descriptor.

MULTIWORD INSTRUCTIONS

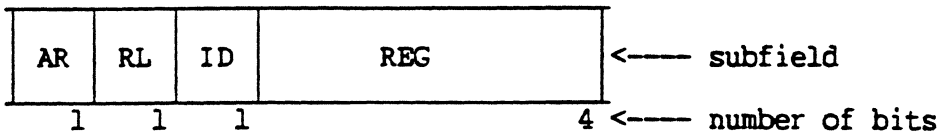
MULTIWORD MODIFICATION FIELD

Each modification field (MF) contained in a multiword instruction is a 7-bit field specifying address modification to be performed on the operand descriptors. The modification field is interpreted as follows:

2 3 4 5 through 8 <--- bits (MF3)

11 12 13 14 through 17 <--- bits (MF2)

29 30 31 32 through 35 <--- bits (MF1)



AR - Address Register Specifier

0- No address register used.

1- Bits 0-2 of the operand descriptor address field specify the address register to be used in computing the effective address of the operand. Bits 0 - 2 also specify the operand descriptor register that defines the segment containing the operand.

RL - Register or Length

0- Operand length is specified in the N field (bits 32-35) of the operand descriptor.

1- Length of operand is contained in the register specified by code in the N field (bits 32-35) of the operand descriptor, in the machine format of REG (the coding format is different).

ID - Indirect Operand Descriptor

0- The operand descriptor follows the instruction word in its sequential memory location.

1- The operand descriptor location contains an indirect word that points to the operand descriptor. Only one level of indirection is allowed.

MULTIWORD INSTRUCTIONS

REG - Address modification register selection for R-type modification of the operand descriptor address field. The REG codes are approximately the same as the single-word modifications. In addition, for indirect string length specification (RL = 1), the N field codes are similar to the REG field. A comparison of these codes is shown in Table 5-2.

Operand Descriptors And Indirect Words

The words following a multiword instruction word are either operand descriptors or indirect words to the operand descriptors. The interpretation of the words is performed according to the settings of the control bits in the associated modification field (MF).

OPERAND DESCRIPTOR INDIRECT WORD FORMAT

An indirect pointer to an operand descriptor is interpreted as shown in Figure 7-5 (also see "Indirect Word" in Section 5):

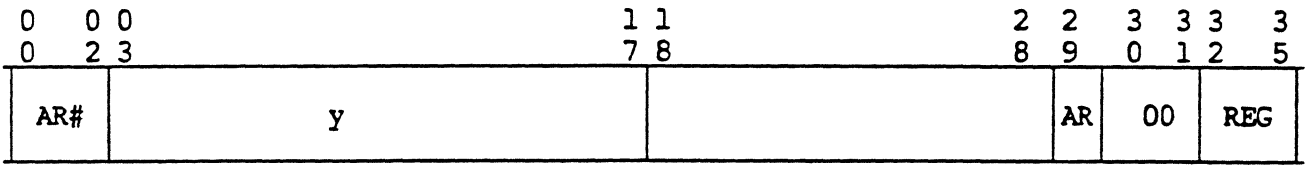


Figure 7-5. Operand Descriptor Indirect Word Format

AR# - A 3-bit pointer register number

y - An 18-bit main memory address or a 15-bit word offset

AR - Indirect via bit 29 flag that controls the interpretation of the y field of the indirect pointer

REG - The address modifier for the y field

Alphanumeric Instructions

Alphanumeric instructions permit moving, transliteration, editing, and comparing of alphanumeric data.

MULTIWORD INSTRUCTIONS

ALPHANUMERIC OPERAND DESCRIPTOR FORMAT

For any operand of a multiword instruction that requires alphanumeric data, the operand descriptor is interpreted as shown In Figure 7-6 (also see "Alphanumeric Operand Descriptors" in Section 5):

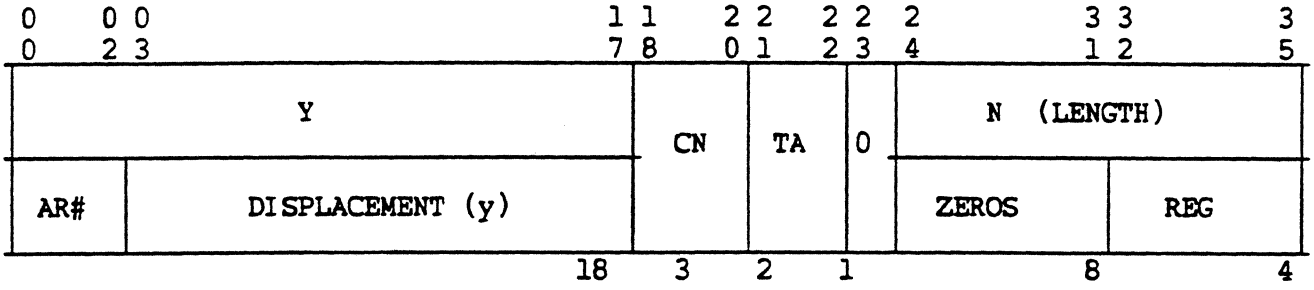


Figure 7-6. Alphanumeric Operand Descriptor Format

AR# - A 3-bit address register number

Y - Location or displacement value

DISPLACEMENT- (y) An 18-bit main memory address or a 15-bit word offset relative to the address register's content

CN - Character number. This field gives the character position within the word at y of the first operand character. Its interpretation depends on the data type (see TA below) of the operand. Table 7-1 shows the interpretation of the field. A digit in the table indicates the corresponding character position (see Section 2 for data formats). Invalid codes cause IPR faults.

Table 7-1. Alphanumeric Character Number (CN) Codes

C(CN)	Data type		
	4-bit	6-bit	9-bit
000	0	0	0
001	1	1	IPR
010	2	2	1
011	3	3	IPR
100	4	4	2
101	5	5	IPR
110	6	IPR	3
111	7	IPR	IPR

TA - Type alphanumeric. This is the data type code for the operand. The interpretation of the field is shown in Table 7-2. The code shown as Invalid causes an IPR fault.

Table 7-2. Alphanumeric Data Type (TA) Codes

C(TA)	Data type
00	9-bit
01	6-bit
10	4-bit
11	IPR

N - Operand length. If RL = 0 in the corresponding MF, this field contains the string length of the operand. (Refer to Multiword Modification Field in this section.) If RL = 1, this field contains the code for a register holding the operand string length (See "Register Codes", Table 5-2).

The alphanumeric operand descriptor is coded as follows:

```
1      8      16
       {ADSC9}  LOCSYM,CN,N,AM
       {ADSC6}                    (braces indicate a choice)
       {ADSC4}
```

where:

LOCSYM - An expression containing either the location of the data or an offset from the base.

CN - Character number (see above)

N - Symbol or decimal value containing either length or a register code

AM - Address register containing the base

ALPHANUMERIC COMPARE

CMPC	106 (1)	Compare Alphanumeric Character Strings
CMPCT	166 (1)	Compare Characters and Translate
SCD	120 (1)	Scan Characters Double
SCDR	121 (1)	Scan Characters Double in Reverse
SCM	124 (1)	Scan with Mask
SCMR	125 (1)	Scan with Mask in Reverse
TCT	164 (1)	Test Character and Translate
TCTR	165 (1)	Test Character and Translate in Reverse

ALPHANUMERIC MOVE

MLR	100 (1)	Move Alphanumeric Left to Right
MRL	101 (1)	Move Alphanumeric Right to Left
MVE	020 (1)	Move Alphanumeric Edited
MVT	160 (1)	Move Alphanumeric with Translation

Character Move To/From Register Instructions

Two instructions permit moves of one, two, three, or four 9-bit characters from a memory location to a register or from a register to memory. An indirect word cannot be used for the data descriptor of this instruction.

MULTIWORD INSTRUCTIONS

OPERAND DESCRIPTOR FOR CHARACTER MOVE INSTRUCTIONS

The word following the character move instruction word is the operand descriptor which specifies the origin or destination of the move, indicates the number of characters to be moved, and specifies whether 9-bit characters or 8-bit bytes are to be moved. This word is illustrated in Figure 7-7.

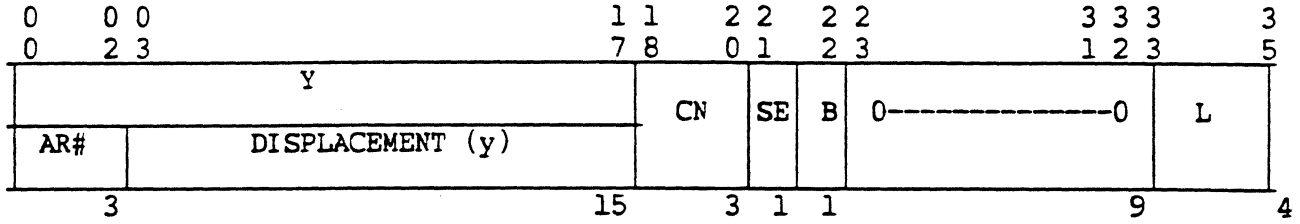


Figure 7-7. Character Move Descriptor Format

The character move operand descriptor is created by entering a one-line pseudo operation coded, SDSCn, following an MTR or MTM instruction. This descriptor serves a similar purpose as operand descriptors used with other multiword instructions. SDSCn creates a descriptor word to transfer 9-bit characters or 8-bit bytes for the MTR/MTM instruction depending upon the specification in n as described below.

```
1       8       16
_______________________________

        SDSCn   LOCSYM,CN,L,SE,AM
```

where:

n - when = 9, B (see descriptor format above) is set to 0 indicating 9-bit characters

when = 8, B is set to 1 indicating 8-bit bytes

LOCSYM - Address of word containing first character to be moved

CN - Character position of left end of operand within a word. Must be 0-3.

L - Number of characters to be moved. Must be 0-4. Defaults to 0.

SE - State of enlargement for character positions. Applies to MTR move only.

MULTIWORD INSTRUCTIONS

AM - Optional address register modification (AR#)

NOTE: Refer to specifications for MTR and MTM in Section 8.

The method of generating a start address for a character move by using the Y field is the same as in other multiword instructions. However, A, Q,X0-X7 or GX0-GX7 must be specified for REG modification when REG modification is used.

CHARACTER MOVE INSTRUCTION REPERTOIRE

MTM	365 (1)	Move to Memory
MTR	361 (1)	Move to Register

Numeric Instructions

The set of numeric instructions deals with sign and magnitude operands. Floating-point decimal zero is represented as + 0 * 10**127. If any computation is performed that would result in a zero representation other than this, the hardware forces the zero representation to this format, thus preventing loss of data during decimal point alignment.

All numeric operations are limited to final results not to exceed 63 characters (sign, digits, exponent). If any numeric move, compare, or calculation is specified involving either a number with more than 63 characters or a final product with more than 63 characters, the operation is performed as though 63 characters were specified and no fault occurs unless the specific description of an instruction states that such a fault occurs and/or that operation does not take place.

All characters are carried internally as 4 bits. The upper 5 bits of any 9-bit input character (TN = 0) are truncated. If a 9-bit output is specified, 00011 (ASCII numeric zone) is appended to form the numeric digits; standard ASCII plus minus characters (octal 053 and 055, respectively) are generated.

NUMERIC OPERAND DESCRIPTOR FORMAT

For any operand of a multiword instruction that requires numeric data, the operand descriptor is interpreted as shown in Figure 7-8 (also see "Numeric Operand Descriptors" in Section 5):

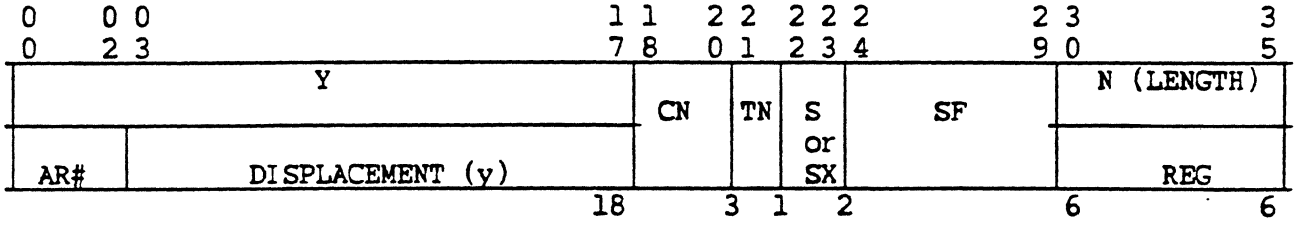


Figure 7-8. Numeric Operand Descriptor Format

AR# - A 3-bit address register number

Y - Location or displacement value

DISPLACEMENT - (y) An 18-bit main memory address or a 15-bit word offset relative to the address register's content.

CN - Character number. This field gives the character position within the word at y of the first operand digit. Its interpretation depends on the data type (see TN below) of the operand.

TN - Type numeric. This is the data type code for the operand. The codes are:

C(T)	Data Type
0	9-bit
1	4-bit

S - Sign and decimal type of data. The interpretation of the field is shown in Table 7-3.

MULTIWORD INSTRUCTIONS

Table 7-3. Sign And Decimal Type (S) Codes

C(S)	Sign and Decimal type
00	Floating-point, leading sign
01	Scaled fixed-point, leading sign
10	Scaled fixed-point, trailing sign
11	Scaled fixed-point, unsigned

SX - Sign and scaling

If TN = 0 (unpacked data)
00 leading sign, overpunched, fixed-point
01 leading sign, separate, fixed-point
10 trailing sign, separate, fixed-point
11 trailing sign, overpunched, fixed-point

If TN = 1, (packed data)
00 leading sign, separate, floating point
01 leading sign, separate, fixed-point
10 trailing sign, separate, fixed-point
11 no sign, fixed-point

(Refer to description of overpunched signs under MVNX in Section 8.)

SF - Scaling factor. This field contains the two's complement value of the base 10 scaling factor(i.e., the value of m for numbers represented as n * 10**m). The decimal point is assumed to the right of the least significant digit of n. Negative values of m move the decimal point to the left; positive values, to the right. The range of m is -32 to 31 treated as the powers of 10.

N - Operand length. If RL = 0 in MF, this field contains the operand length in digits. If RL = 1, it contains the REG code for the register holding the operand length and C(REG) is treated as a 0 modulo 64 number.

MULTIWORD INSTRUCTIONS

The numeric operand descriptor is coded as follows:

```
1       8       16
_____________________

       {NDSC9}  LOCSYM,CN,N,S,SF,AM
       {NDSC4}
```

where:

LOCSYM - An expression containing either the location of the data or an offset from the base

CN - Character number (see above)

N - A symbol or decimal value containing either the length for a register code.

S - The sign and decimal type in two bits:

Code	Description
0	Floating-point, leading sign
1	Scaled fixed-point, leading sign
2	Scaled fixed-point, trailing sign
3	Scaled fixed-point, unsigned

SX - Sign and scaling (see above).

SF - The scaling factor for scaled decimal numbers; range is -31 to +32 treated as the powers of 10

AM - Address register containing the base (AR#)

NUMERIC COMPARE

CMPN	303 (1)	Compare Numeric
CMPNX	343 (1)	Compare Numeric Extended

NUMERIC MOVE

MVN	300 (1)	Move Numeric
MVNX	340 (1)	Move Numeric Extended
MVNE	024 (1)	Move Numeric Edited
MVNEX	004 (1)	Move Numeric Edited Extended

MULTIWORD INSTRUCTIONS

Bit String Instructions

These instructions provide the capability of performing Boolean operations on bit strings. The Boolean Result (BOLR) control field (bits 5, 6, 7, and 8 of the instruction word) defines one of 16 possible logical operations to be performed. The four bits in this field are associated with the four possible combinations of bits from the two operands. The association rule is:

If first operand bit is:	and second operand bit is:	then result is from bit:
0	0	5
0	1	6
1	0	7
1	1	8

Boolean operations most commonly used are:

	BOLR Field Bits			
Operation	5	6	7	8
MOVE	0	0	1	1
AND	0	0	0	1
OR	0	1	1	1
NAND	1	1	1	0
EXCLUSIVE OR	0	1	1	0
Clear	0	0	0	0
Invert	1	1	0	0

The four bits contained in the Boolean control field are represented in the instruction format by one or two octal digits.

BIT STRING OPERAND DESCRIPTOR FORMAT

For any operand of a multiword instruction that requires bit string data, the operand descriptor is interpreted as shown in Figure 7-9 (also see "Bit String Operand Descriptor" in Section 5):

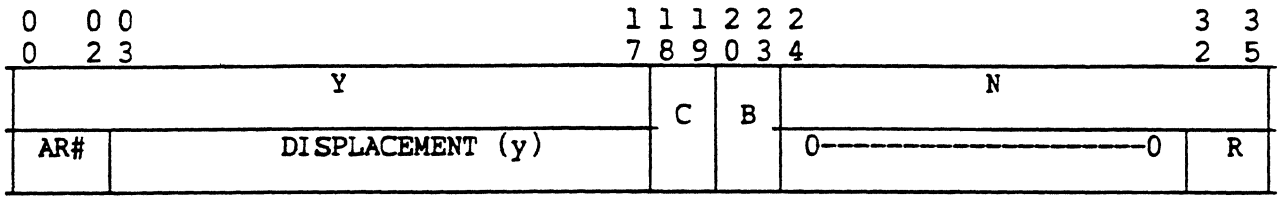


Figure 7-9. Bit String Operand Descriptor Format

AR# - A 3-bit address register number

Y - Location or displacement value

DISPLACEMENT - (Y) An 18-bit main memory address or a 15-bit word offset relative to the address register's content

C - The character number of the 9-bit character within the y field containing the first bit of the operand

B - The bit number within the 9-bit character, C, of the first bit of the operand

N - Operand length. If RL = 0 in MF, this field contains the string length of the operand. If RL = 1, this field contains the code for a register holding the operand string length.

R - Register containing data length

The bit string operand descriptor is coded as follows:

```
1       8       16
        BDSC    LOCSYM,N,C,B,AM
```

where:

LOCSYM - An expression containing either the location of the data or an offset from the base

MULTIWORD INSTRUCTIONS

N - Symbol or decimal value containing either length or a register code

C - Character position (0-3)

B - Bit within character (0-8)

AM - Address register containing the base (AR#)

BIT STRING COMBINE

CSL	060 (1)	Combine Bit Strings Left
CSR	061 (1)	Combine Bit Strings Right

BIT STRING COMPARE

CMPB	066 (1)	Compare Bit String

BIT STRING SET INDICATORS

SZTL	064 (1)	Set Zero and Truncation Indicators with Bit Strings Left
SZTR	065 (1)	Set Zero and Truncation Indicators with Bit Strings Right

Data Conversion Instructions

Conversion instructions are used for conversions between binary and decimal numbers where the binary number is stored as a character string, starting and ending on 9-bit character boundaries, and the decimal number is stored as a character string.

BTD	301 (1)	Binary-to-Decimal Convert
DTB	305 (1)	Decimal-to-Binary Convert

MULTIWORD INSTRUCTIONS

Arithmetic Instructions

DECIMAL ADDITION

AD2D	202 (1)	Add Using Two Decimal Operands
AD2DX	242 (1)	Add Using Two Decimal Operands Extended
AD3D	222 (1)	Add Using Three Decimal Operands
AD3DX	262 (1)	Add Using Three Decimal Operands Extended

DECIMAL SUBTRACTION

SB2D	203 (1)	Subtract Using Two Decimal Operands
SB2DX	243 (1)	Subtract Using Two Decimal Operands Extended
SB3D	223 (1)	Subtract Using Three Decimal Operands
SB3DX	263 (1)	Subtract Using Three Decimal Operands Extended

DECIMAL MULTIPLICATION

MP2D	206 (1)	Multiply Using Two Decimal Operands
MP2DX	246 (1)	Multiply Using Two Decimal Operands Extended
MP3D	226 (1)	Multiply Using Three Decimal Operands
MP3DX	266 (1)	Multiply Using Three Decimal Operands Extended

DECIMAL DIVISION

DV2D	207 (1)	Divide Using Two Decimal Operands
DV2DX	247 (1)	Divide Using Two Decimal Operands Extended
DV3D	227 (1)	Divide Using Three Decimal Operands
DV3DX	267 (1)	Divide Using Three Decimal Operands Extended

MICRO OPERATIONS FOR EDIT INSTRUCTIONS MVE, MVNE, AND MVNEX

The Move Alphanumeric Edited (MVE), Move Numeric Edited (MVNE), and Move Numeric Edited Extended (MVNEX) instructions require micro operations to perform the editing functions in an efficient manner. The sequence of micro operation steps to be executed is contained in memory and is referenced by the second operand descriptor of the instruction. Some micro operations require special characters for insertion into the string of characters being edited. These special characters are shown in the edit insertion tables in this section.

Micro Operation Sequence

The micro operation string operand descriptor points to a string of 9-bit bytes that specifies the micro operations to be performed during an edited move. Each of the 9-bit bytes defines a micro operation and has the format shown in Figure 7-10:

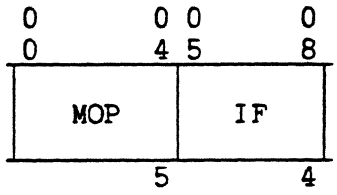


Figure 7-10. Micro Operation (MOP) Character Format

MOP 5-bit code specifying the micro operator
(Refer to the Micro Operation Repertoire.)

IF Information field containing one of the following:

1. A sending string character count. A value of 0 is interpreted as 16.

2. The index of an entry in the edit insertion table to be used. Permissible values are 1 through 8.

3. An interpretation of the "blank-when-zero" operation

Edit Insertion Tables

While executing an edit instruction, the processor provides a register of eight 9-bit bytes to hold insertion information. This register, called the edit insertion table, is not maintained after execution of an edit instruction. At the start of each edit instruction, the processor initializes the table to the values given in Table 7-4. For MVE and MVNE, the ASCII code is used for each initial value. For MVNEX, the BIT field in the instruction word determines the character set (ASCII, BCD, or EBCDIC) to be used for the initial values. (Refer to the Edit Insertion Table Entries in Table 7-5.)

Table 7-4. Default Edit Insertion Table Characters For MVE And MVNX

Table Entry Number	Character
1	space
2	*
3	+
4	-
5	$
6	,
7	.
8	0 (zero)

The relationship between the ASCII character bit positions and the table character positions is as follows:

```
0 1 2 3 4 5 6 7 8  Table character bit positions
-----------------------------------------------
9 8 7 6 5 4 3 2 1  ASCII character bit positions
```

where unused high-order bit positions of the character are zero-filled. One or all of the table entries may be changed by the Load Table Entry (LTE) or the Change Table (CHT) micro operation to provide different insertion characters.

MICRO OPERATIONS

Table 7-5. Edit Insertion Table Entries For MVNEX

Edit Insertion Table		Octal Code		
No.	Character	EBCDIC	BCD	ASCII
1	b (space)	100	020	040
2	* (asterisk)	134	054	052
3	+ (plus)	116	060	053
4	- (minus)	140	052	055
5	$ (dollar sign)	133	053	044
6	, (comma)	153	073	054
7	. (period)	113	033	056
8	0 (zero)	360	000	060

NOTE: The table entries may be changed by use of the Load Table Entry (LTE) or Change Table Entry (CHT) micro operations described on following pages.

MVNE, MVE, And MVNEX Differences

The processor executes MVNE and MVNEX in a slightly different manner than it executes MVE because of inherent differences in how numeric and alphanumeric data is handled. The following are brief descriptions of the basic operations.

NUMERIC EDIT (MVNE AND MVNEX)

1. Load the entire sending string number (maximum length 63 characters) into the decimal unit input buffer as 4-bit digits (high-order truncating 9-bit data). Strip the sign and exponent characters (if any), put them aside into special holding registers, and decrease the input buffer count accordingly.

2. Test sign and, if required, set the SN flag.

3. Execute micro operation string, starting with the first (4-bit) digit.

4. If an edit insertion table entry or MOP insertion character is to be stored, ANDed, or ORed into a receiving string of 4- or 6-bit characters, high-order truncate the character accordingly.

5. If the receiving string is 9-bit characters, high-order fill the (4-bit) digits from the input buffer with bits 0-4 of character 8 of the edit insertion table. If the receiving string is 6-bit characters, high-order fill the digits with "00".

ALPHANUMERIC EDIT (MVE)

1. Load the decimal unit input buffer with sending string characters. Data is read from memory in unaligned units (not modulo 8 boundary) of four double-words. The number of characters loaded is the minimum of the remaining sending string count, the remaining receiving string count, and 64.

2. Perform tests for zero on the four least significant bits of each character.

3. Execute micro operation string, starting with the first receiving string character.

4. If an edit insertion table entry or MOP insertion character is to be stored, ANDed, or ORed into a receiving string of 4- or 6-bit characters, use the lower 4 or 6 bits.

5. If the receiving string is 6- or 9-bit characters, the zero-fill is already supplied; do not append bits of any edit insertion table entry as the most significant bits.

Micro Operation Repertoire

MOP	Octal	Binary	Operation
CHT	21	10001	Change Table
ENF	02	00010	End Floating Suppression
IGN	14	01100	Ignore Source Characters
INSA	11	01001	Insert Asterisk on Suppression
INSB	10	01000	Insert Blank on Suppression
INSM	01	00001	Insert Table Entry One Multiple

MOP	Octal	Binary	Operation
INSN	12	01010	Insert On Negative
INSP	13	01011	Insert On Positive
LTE	20	10000	Load Table Entry
MFLC	07	00111	Move With Floating Currency Symbol Insertion
MFLS	06	00110	Move With Floating Sign Insertion
MORS	17	01111	Move and OR Sign
MSES	16	01110	Move and Set Sign
MVC	15	01101	Move Source Characters
MVZA	05	00101	Move With Zero Suppression and Asterisk Replacement
MVZB	04	00100	Move With Zero Suppression and Blank Replacement
SES	03	00011	Set End Suppresion

Micro Operations Descriptions

A description of the 17 micro operations (MOPs) follows. The descriptions are presented in the format shown below.

MOP	Operation	Binary Code

EXPLANATION: Describes how the operation functions

FLAGS: Describes the setting of the affected flags

NOTES: Describes any fault conditions

Checks for termination are made during and after each micro operation. All MOPs that make a zero test of a sending-string character, test only the four least-significant bits of the character.

Edit Flags

The processor provides the following four edit flags for use by the micro operations.

ES End suppression flag; initially OFF and set ON by a micro operation when zero-suppression ends. (This ES should not be confused with the ES mode.)

SN Sign flag; initially set OFF if the sending string has an alphanumeric descriptor or an unsigned numeric descriptor. If the sending string has a signed numeric descriptor, the sign is initially read from the sending string from the digit position defined by the sign and the decimal type field (S or SX); SN is set OFF if positive, ON if negative. If all digits are zero, the data is assumed positive and the SN flag is set OFF, even when the sign is negative.

Z Zero flag; initially set ON and set OFF whenever a sending string character that is not decimal zero is moved into the receiving string.

BZ Blank-when-zero flag; initially set OFF and set ON by either the ENF or SES micro operation. If, at the completion of a move (L1 exhausted), both the Z and BZ flags are ON, the receiving string is filled with character 1 of the edit insertion table.

CHT	Change Table	10001

EXPLANATION: The edit insertion table is replaced by the string of eight 9-bit characters immediately following the CHT micro operation.

FLAGS: None affected

NOTE: C(IF) is not interpreted for this operation.

ENF	End Floating Suppression	00010

EXPLANATION: Bit 0 of IF (IF_0) specifies the nature of the floating suppression.

Bit 1 of IF (IF_1) specifies if blank when zero option is used.

For IF_0 = 0 (end floating-sign operation):

If ES is OFF and SN is OFF, then edit insertion table entry 3 is moved to the receiving field and ES is set ON.

If ES is OFF and SN is ON, then edit insertion table entry 4 is moved to the receiving field and ES is set ON.

If ES is ON, no action is taken.

For IF_0 = 1 (end floating currency symbol operation):

If ES is OFF, then edit insertion table entry 5 is moved to the receiving field and ES is set ON.

If ES is ON, no action is taken.

For IF_1 = 1 (blank when zero): the BZ flag is set ON.

For IF_1 = 0 (no blank when zero): no action is taken.

FLAGS: (Flags not listed are not affected)

ES - If OFF, then set ON

BZ - If bit 1 of C(IF) = 1, then set ON; otherwise, unchanged

IGN	Ignore Source Characters	01100

EXPLANATION: IF specifies the number of characters to be ignored, where IF = 0 specifies 16 characters.

The next IF characters in the source data field are ignored and the sending tally is reduced accordingly.

FLAGS: None affected

INSA	Insert Asterisk on Suppression	01001

EXPLANATION: Same as INSB except that if ES is OFF, then edit insertion table entry 2 is moved to the receiving field.

FLAGS: None affected

NOTE: If C(IF) = 9-15, an IPR fault occurs.

INSB	Insert Blank on Suppression	01000

EXPLANATION: IF specifies which edit insertion table entry is inserted.

If IF = 0, the 9 bits immediately following the INSB micro operation are treated as a 9-bit character (not a MOP) and are moved or skipped according to ES:

If ES is OFF, then edit insertion table entry 1 is moved to the receiving field. If IF = 0, then the next 9 bits are also skipped. If IF is not 0, the next 9 bits are treated as a MOP.

If ES is ON and IF = 0, then the 9-bit character immediately following the INSB micro-instruction is moved to the receiving field.

If ES is ON and IF ≠ 0, then IF specifies which edit insertion table entry (1-8) is to be moved to the receiving field.

FLAGS: None affected

NOTE: If C(IF) = 9-15, an IPR fault occurs.

INSM	Insert Table Entry One Multiple	00001

EXPLANATION: IF specifies the number of receiving characters affected, where IF = 0 specifies 16 characters.

Edit insertion table entry 1 is moved to the next IF (1-16) receiving field characters.

FLAGS: None affected

INSN	Insert On Negative	01010

EXPLANATION: IF specifies which edit insertion table entry is inserted. If IF = 0, the 9 bits immediately following the INSN micro operation are treated as a 9-bit character (not a MOP) and are moved or skipped according to SN:

If SN is OFF, then edit insertion table entry 1 is moved to the receiving field. If IF = 0, then the next 9 bits are also skipped. If IF is not 0, the next 9 bits are treated as a MOP.

If SN is ON and IF = 0, then the 9-bit character immediately following the INSN micro-instruction is moved to the receiving field.

If SN is ON and IF $\neq$ 0, then IF specifies which edit insertion table entry (1-8) is to be moved to the receiving field.

FLAGS: None affected

NOTE: If C(IF) = 9-15, an IPR fault occurs.

INSP	Insert On Positive	01011

EXPLANATION: Same as INSN except that the responses for the SN values are reversed.

FLAGS: None affected

NOTE: If C(IF) = 9-15, an IPR fault occurs.

LTE	Load Table Entry	10000

EXPLANATION: IF specifies the edit insertion table entry to be replaced.

The edit insertion table entry specified by IF is replaced by the 9-bit character immediately following the LTE micro instruction.

FLAGS: None affected

NOTE: If C(IF) = 0 or C(IF) = 9-15, an Illegal Procedure fault occurs.

MFLC	Move with Floating Currency Symbol Insertion	00111

EXPLANATION: IF specifies the number of characters of the sending field upon which the operation is performed, where IF = 0 specifies 16 characters.

Starting with the next available sending field character, the next IF characters are individually fetched and the following conditional actions occur:

If ES is OFF and the character is zero, edit insertion table entry 1 is moved to the receiving field in place of the character.

If ES is OFF and the character is not zero, then edit insertion table entry 5 is moved to the receiving field, the character is also moved to the receiving field, and ES is set ON.

If ES is ON, the character is moved to the receiving field.

The number of characters placed in the receiving field is data-dependent. If the entire sending field is zero, IF characters are placed in the receiving field. However, if the sending field contains a nonzero character, IF+1 characters (the insertion character plus the characters from the sending field) are placed in the receiving field.

An IPR fault occurs when the sending field is exhausted before the receiving field is filled. In order to provide space in the receiving field for an inserted currency symbol, the receiving field must have a string length one character longer than the sending field. When the sending field is all zeros, no currency symbol is inserted by the MFLC micro operation and the receiving field is not filled when the sending field is exhausted. The user should provide an ENF (ENF,12) micro operation after a MFLC micro operation that has as its character count the number of characters in the sending field. The ENF micro operation is engaged only when the MFLC micro operation fails to fill the receiving field; then, it supplies a currency symbol to fill the receiving field and blanks out the entire field.

FLAGS: (Flags not listed are not affected)

ES - If OFF and any of C(Y) is less than decimal zero, then ON; otherwise, unchanged

NOTE: Since the number of characters moved to the receiving string is data-dependent, a possible IPR fault may be avoided by ensuring that the Z and BZ flags are ON.

MICRO OPERATIONS

MFLS	Move with Floating Sign Insertion	00110

EXPLANATION: IF specifies the number of characters of the sending field upon which the operation is performed, where IF = 0 specifies 16 characters.

Starting with the next available sending field character, the next IF characters are individually fetched and the following conditional actions occur:

If ES is OFF and the character is zero, edit insertion table entry 1 is moved to the receiving field in place of the character.

If ES is OFF, the character is not zero, and SN is OFF; then edit insertion table entry 3 is moved to the receiving field. The character is also moved to the receiving field, and ES is set ON.

If ES is OFF, the character is nonzero, and SN is ON; edit insertion table entry 4 is moved to the receiving field; the character is also moved to the receiving field, and ES is set ON.

If ES is ON, the character is moved to the receiving field.

The number of characters placed in the receiving field is data-dependent. If the entire sending field is zero, IF characters are placed in the receiving field. However, if the sending field contains a nonzero character, IF+1 characters (the insertion character plus the characters from the sending field) are placed in the receiving field.

An IPR fault occurs when the sending field is exhausted before the receiving field is filled. In order to provide space in the receiving field for an inserted sign, the receiving field must have a string length one character longer than the sending field. When the sending field is all zeros, no sign is inserted by the MFLS micro operation and the receiving field is not filled when the sending field is exhausted. The user should provide an ENF (ENF,4) micro operation after a MFLS micro operation that has as its character count the number of characters in the sending field. The ENF micro operation is engaged only when the MFLS micro operation fails to fill the receiving field; then, it supplies a sign character to fill the receiving field and blanks out the entire field.

FLAGS: (Flags not listed are not affected)

ES - If OFF and any of C(Y) is less than decimal zero, then ON; otherwise, unchanged

NOTE: Since the number of characters moved to the receiving string is data-dependent, a possible Illegal Procedure fault may be avoided by ensuring that the Z and BZ flags are ON.

MORS	Move and OR Sign	01111

EXPLANATION: IF specifies the number of characters of the sending field upon which the operation is performed, where IF = 0 specifies 16 characters.

Starting with the next available sending field character, the next IF characters are individually fetched and the following conditional actions occur:

If SN is OFF, the next IF characters in the source data field are moved to the receiving data field and, during the move, edit insertion table entry 3 is ORed to each character.

If SN is ON, the next IF characters in the source data field are moved to the receiving data field and, during the move, edit insertion table entry 4 is ORed to each character.

MORS can be used to generate a negative overpunch for a receiving field to be used later as a sending field.

FLAGS: None affected

MSES	Move and Set Sign	01110

EXPLANATION: IF specifies the number of characters of the sending field upon which the operation is performed, where IF = 0 specifies 16 characters.

For MVE, starting with the next available sending field character, the next IF characters are individually fetched and the following conditional actions occur:

Starting with the first character during the move, a comparative AND is made first with edit insertion table entry 3. If the result is nonzero, the first character and the rest of the characters are moved without further comparative ANDs. If the result is zero, a comparative AND is made between the character being moved and edit insertion table entry 4 If that result is nonzero, the SN indicator is set ON (indicating negative) and the first character and the rest of the characters are moved without further comparative ANDs. If the result is zero, the second character is treated like the first. This continues until one of the comparative AND results is nonzero or until all characters are moved.

For MVNE and MVNEX instructions, the sign (SN) flag is already set and IF characters are moved to the destination field (MSES is equivalent to the MVC instruction).

FLAGS: (Flags not listed are not affected)

SN - If edit insertion table entry 4 is found in C(Y-1), then ON; otherwise, unchanged

MVC	Move Source Characters	01101

EXPLANATION: IF specifies the number of characters to be moved, where IF = 0 specifies 16 characters.

The next IF characters in the source data field are moved to the receiving data field.

FLAGS: None affected

MVZA	Move with Zero Suppression and Asterisk Replacement	00101

EXPLANATION: Same as MVZB except that:

If ES is OFF and the character is zero, then edit insertion table entry 2 is moved to the receiving field.

FLAGS: (Flags not listed are not affected)

ES - If OFF and any of C(Y) is less than decimal zero, then ON; otherwise, unchanged

MVZB	Move with Zero Suppression and Blank Replacement	00100

EXPLANATION: IF specifies the number of characters of the sending field upon which the operation is performed, where IF = 0 specifies 16 characters.

Starting with the next available sending field character, the next IF characters are individually fetched and the following conditional actions occur:

If ES is OFF and the character is zero, then edit insertion table entry 1 is moved to the receiving field in place of the character.

If ES is OFF and the character is not zero, then the character is moved to the receiving field and ES is set ON.

If ES is ON, the character is moved to the receiving field.

FLAGS: (Flags not listed are not affected)

ES - If OFF and any of C(Y) is less than decimal zero, then ON; otherwise, unchanged

MICRO OPERATIONS

SES	Set End Suppression	00011

EXPLANATION: Bit 0 of IF (IF_0) specifies the setting of the ES switch.

Bit 1 of IF (IF_1) specifies the setting of the blank-when-zero option.

If IF_0 = 0, the ES flag is set OFF.

IF IF_0 = 1, the ES flag is set ON.

If IF_1 = 1, the BZ flag is set ON.

If IF_1 = 0, no action is taken.

FLAGS: (Flags not listed are not affected)

ES - Set by this micro operation

BZ - If bit 1 of C(IF) = 1, then ON; otherwise, unchanged

Micro Operation Code Assignment Map

Operation code assignments for the micro operations are shown in Table 7-6. Dashes (----) indicate an unassigned code. Unassigned codes cause an Illegal Procedure fault.

Table 7-6. Micro Operation Code Assignment Map

B_0 B_1 \ B_2 B_3 B_4	000	001	010	011	100	101	110	111
00	----	INSM	ENF	SES	MVZB	MVZA	MFLS	MFLC
01	INSB	INSA	INSN	INSP	IGN	MVC	MSES	MORS
10	LTE	CHT	----	----	----	----	----	----
11	----	----	----	----	----	----	----	----

Terminating Micro Operations

The micro-operation sequence is terminated normally when the receiving string length is exhausted. The micro-operation sequence is terminated abnormally (with an IPR fault) if an attempt is made to move from an exhausted sending string or to use an exhausted MOP string.

MICRO OPERATIONS EXAMPLES:

```
1       8       16                  32
        MVNE
        NDSC4   EPACK,5,11,2    PIC   S9(10)
        ADSC9   MOPLST,0,9
        ADSC6   PRTOUT+3,0,12   PIC   Z(7).999-
        USE     DETOUR
MOPLST  MICROP  (LTE,1),1H ,(MVZB,7),(SES,8)
        MICROP  (INSB),1H.,(MVC,3),(INSN)
        MICROP  1H-,(LTE,1),1H ,(MVZB,2),(MVC,1)
        USE

        MVNE
        NDSC4   FPACK,5,11,2    PIC   S9(10)
        ADSC9   MOPLST,0,9
        ADSC6   PRTOUT+6,0,12   PIC   Z(7).999-
        MVNE
        NDSC4   SEQPAK,5,3,3    PIC   999
        ADSC9   MOPLST+2,1,4
        ADSC6   PRTOUT+1,3,3    PIC   ZZ9
```

VIRTUAL MEMORY INSTRUCTIONS

VIRTUAL MEMORY INSTRUCTIONS

These instructions support segmentation and paging in the virtual memory environment. Except in the case of the CLIMB instruction, the format of these instructions is the same as the other single-word instructions.

Descriptor Register Instructions

These instructions provide the capability of loading or storing a descriptor register (DRn) with a new descriptor or modifying the descriptor currently contained in DRn. The LDDn instruction has a direct load option and a vector option.

LDDn	67n (1)	Load Descriptor Register n
SDRn	11n (1)	Save Descriptor Register n
STDn	05n (1)	Store Descriptor Register n

Pointer Register Instructions

LDPn	47n (1)	Load Pointer Register n
STPn	45n (1)	Store Pointer n
EPPRn	63n (1)	Effective Pointer to Pointer Register n
LDEAn	61n (1)	Load Extended Address n

Domain Transfer (CLIMB)

The CLIMB domain transfer instruction provides the software with a hardware mechanism for transferring control from one software function to another with a high level of software security. This 2-word instruction, described in detail in Section 8, has four versions which perform the functions of call, return, and co-routine invocations for intra- and inter-instruction segments and intra- and inter-domain references.

CLIMB	713 (1)	Domain Transfer

PRIVILEGED INSTRUCTIONS

PRIVILEGED INSTRUCTIONS

Privileged instructions are executed in Privileged Master mode. Three conditions must be met before the instructions can be executed:

1. The master mode bit in the indicator register must be ON.
2. The privileged bit in the instruction segment register must be ON.
3. The housekeeping bit in the page table word for the page containing the instruction must be ON; if the processor is in the working space zero addressing mode, this bit is assumed ON.

If any of the above conditions does not exist upon the attempted execution of a privileged instruction, a Command fault occurs.

CLEAR ASSOCIATIVE MEMORY PAGES

CAMP	532 (1)	Clear Associative Memory Pages

CLEAR CACHE

CCAC	011 (1)	Clear Cache

REGISTER LOAD

LDAS	770 (1)	Load Argument Stack Register
LDDSA	170 (1)	Load Data Stack Address Register
LDDSD	571 (1)	Load Data Stack Descriptor Register
LDPS	771 (1)	Load Parameter Segment Register
LDSS	773 (1)	Load Safe Store Register
LDWS	772 (1)	Load Working Space Registers
LPDBR	171 (1)	Load Page Table Directory Base Register

PRIVILEGED INSTRUCTIONS

REGISTER STORE

SPDBR	151 (1)	Store Page Table Directory Base Register
STAS	750 (1)	Store Argument Stack Register
STDSA	150 (1)	Store Data Stack Address Register
STDSD	551 (1)	Store Data Stack Descriptor Register
STPDW	155 (1)	Store PTWAM Directory Word
STPS	751 (1)	Store Parameter Segment Register
STPTW	157 (1)	Store PTWAM Register
STSS	753 (1)	Store Safe Store Register
STWS	752 (1)	Store Working Space Registers

MEMORY CONTROL

LIMR	553 (0)	Load Interrupt Mask Register
RIMR	233 (0)	Read Interrupt Mask Register

SYSTEM CONTROL

CIOC	015 (0)	Connect Input/Output Channel
DIS	616 (0)	Delay Until Interrupt Signal
LCON	016 (0)	Load Connect Table
LCPR	674 (0)	Load Central Processor Register
LDAT	336 (1)	Load Address Trap Register
LDT	637 (0)	Load Timer Register
LRMB	712 (0)	Load Reserve Memory Base
RCW	250 (0)	Read Connect Word Pair
RICHR	156 (1)	Restart IC History Register
RIW	412 (0)	Read Interrupt Word Pair
RMID	273 (0)	Read Memory ID Register
RMR	270 (0)	Read Memory Register
RPAT	611 (0)	Run PATROL
RRES	231 (0)	Read Reserved Memory
RSCR	413 (0)	Read System Control Register
SCPR	452 (0)	Store Central Processor Register
SICHR	154 (1)	Store IC History Register
SIW	451 (0)	Set Interrupt Word Pair
SMID	272 (0)	Set Memory ID Register
SMR	271 (0)	Set Memory Register
SSCR	057 (0)	Set System Control Register
STTA	553 (1)	Store Test Address Registers
STTD	550 (1)	Store Test Descriptor Registers

ALL MODE INSTRUCTIONS

ALL MODE INSTRUCTIONS

All mode instructions may be executed in any processor mode.

EPAT	412 (1)	Effective Pointer and Address to Test
PAS	176 (1)	Pop Argument Stack
RSW	231 (0)	Read Processor Model Characteristics

ES MODE INSTRUCTIONS

ES MODE INSTRUCTIONS

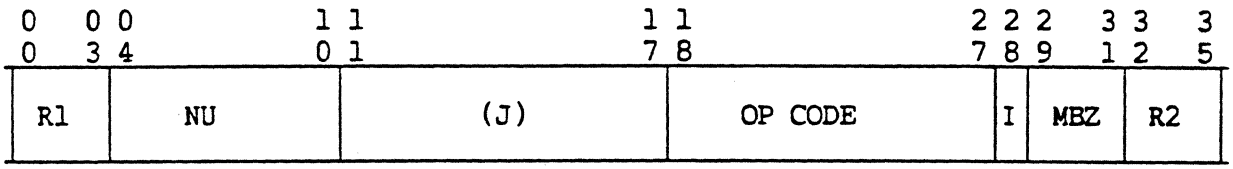
ES mode instructions are valid only in the ES mode (ISR bit 24=1). AN IPR fault occurs if an attempt is made to execute these instructions in the NS mode. Although these instructions are generated by some compilers in this release, they are not supported by the GMAP assembler.

Except for the AARn, NARn, ARAn, and ARN instructions, all instructions are valid in the ES mode. An IPR fault occurs if an attempt is made to execute these four instructions in the ES mode.

Register-to-Register Instructions

Register to Register instructions known as "RR" type instructions are valid only in the ES mode. An attempt to execute these instructions in the NS mode results in an IPR fault. RR type instructions permit movement, arithmetic operation, and shift of fixed-point data using the GXn, A and Q registers. An attempt to execute any RR type instruction by the RPT, RPD, or RPL instructions results in an IPR fault.

RR TYPE INSTRUCTION FORMAT



ES MODE INSTRUCTIONS

Bits	Field	Description
0 - 3	R1	Specifies a code indicating a register to be the destination of the result. The allowable codes follow:

Register Code	Result
0000	IPR
0001	IPR
0010	IPR
0011	IPR
0100	IPR
0101	A
0110	Q
0111	IPR
1000	GX0
1001	GX1
1010	GX2
1011	GX3
1100	GX4
1101	GX5
1110	GX6
1111	GX7

4 -10	NU	Not used. Should be set to 0.
11-17	J	Used only in a shift instruction. Specifies the shift number (immediate value). Must be 0 in all but shift instructions.
18-27	OP	Operation code
28	I	Interrupt inhibit bit
29-31	MBZ	Must be zero or an IPR fault occurs
32-35	R2	Specifies a code that indicates a source register. The codes for this register are the same as for R1.

ES MODE INSTRUCTIONS

NOTES: 1. Specifying a register code of 0000 in a shift instruction does not result in an IPR fault.

2. If a register pair appears in an instruction specification, the two registers are handled as linked. The list below indicates the register codes to be assocciated with the register pair.

Register Code	Result
0000	IPR
0001	IPR
0010	IPR
0011	IPR
0100	IPR
0101	A, Q
0110	A, Q
0111	IPR
100x	GX0, GX1
101x	GX2, GX3
110x	GX4, GX5
111x	GX6, GX7

where x means this bit is ignored by the hardware.

MOVEMENT AND ARITHMETIC INSTRUCTIONS

ADLR	435 (1)	Add Logical to Register
ADRR	434 (1)	Add Register to Register
ANRR	535 (1)	AND Register to Register
CMRR	534 (1)	Compare Register to Register
DVRR	533 (1)	Divide Register to Register
ERRR	537 (1)	Exclusive OR Register to Register
LDCR	431 (1)	Load Complement to Register
LDDR	433 (1)	Load Double Register to Register
LDPR	432 (1)	Load Positive Register to Register
LDRR	430 (1)	Load Register to Register
MPRR	530 (1)	Multiply Register-Pair to Register
MPRS	531 (1)	Multiply Register-Single to Register
ORRR	536 (!)	OR Register to Register
SBLR	437 (1)	Subtract Logical to Register
SBRR	436 (1)	Subtract Register to Register

ES MODE INSTRUCTIONS

SHIFT INSTRUCTIONS

GLLS	466 (1)	GXn Long Left Shift
GLRL	465 (1)	GXn Long Right Logic
GLRS	464 (1)	GXn Long Right Shift
GLS	462 (1)	GXn Left Shift
GRL	461 (1)	GXn Right Logic
GRS	460 (1)	GXn Right Shift

Fixed-Point Instructions

The fixed-point instructions concern movement and arithmetic operations on data in the GXn registers and memory. These instructions are valid only in the ES mode. An attempt to execute these instructions in the NS mode results in an IPR fault.

GLDD	32n (1)	Load Double to GXn (n = 0,2,4,6)
GSTD	14n (1)	Store Double from GXn (n = 0,2,4,6)
MPX	04n (1)	Multiply GXn (n = 0,1,...,7)

TRANSFER INSTRUCTIONS

TRANSFER INSTRUCTIONS

The program transfer instructions permit conditional and unconditional transfers. TSXn also permits the instruction counter to be stored in index registers X0 through X7 Conditional transfers on zero, plus, and carry also have the corollary transfers nonzero, minus, and no carry. The transfers on overflows and underflows are made to maskable fault routines. If the normal fault routine is masked, transfer is optional. As described in the individual descriptions in Section 8, the ISR and SEGID(IS) are affected by transfer of control instructions.

Conditional Transfer

TEO	614 (0)	Transfer on Exponent Overflow
TEU	615 (0)	Transfer on Exponent Underflow
TMI	604 (0)	Transfer on Minus
TMOZ	604 (1)	Transfer on Minus or Zero
TNC	602 (0)	Transfer on No Carry
TNZ	601 (0)	Transfer on Nonzero
TOV	617 (0)	Transfer on Overflow
TPL	605 (0)	Transfer on Plus
TPNZ	605 (1)	Transfer on Plus and Nonzero
TRC	603 (0)	Transfer on Carry
TRCTn	54n (1)	Transfer on Count
TRTF	601 (1)	Transfer on Truncation Indicator OFF
TRTN	600 (1)	Transfer on Truncation Indicator ON
TTF	607 (0)	Transfer on Tally Runout Indicator OFF
TTN	606 (1)	Transfer on Tally Runout Indicator ON
TZE	600 (0)	Transfer on Zero

Unconditional Transfer

RET	630 (0)	Return
TRA	710 (0)	Transfer Unconditionally
TSS	715 (0)	Transfer after Setting Slave
TSXn	70n (0)	Transfer and Set Index Register n

MISCELLANEOUS INSTRUCTIONS

MISCELLANEOUS INSTRUCTIONS

Option Register Instructions

LDO	172 (1)	Load Option Register
STO	152 (1)	Store Option Register

Binary-To-BCD Conversion

The Binary to Binary-Coded-Decimal (BCD) instruction converts the magnitude of a 33-bit or smaller binary number to its decimal equivalent in BCD form. The conversion is made automatically, one decimal digit per instruction execution, using previously stored conversion constants. The BCD form of the converted number is readily available for further operations.

BCD	505 (0)	Binary-to-BCD Convert

Execute Instructions

The Execute and Execute Double (XEC and XED) instructions allow remote instructions to be executed singly or in pairs. (XED executes only in NS mode.) A program will continue sequentially after the XEC or XED instructions are executed, as long as the referenced instructions do not alter the instruction counter. If a referenced instruction affects the instruction counter, a program transfer occurs.

XEC	716 (0)	Execute
XED	717 (0)	Execute Double

Gray-To-Binary-Conversion

The Gray-to-Binary (GTB) instruction converts a 36-bit word containing data in the Gray code (for example, coded analog information from an analog-to-digital input device) to its binary equivalent in only one execution of the instruction. This instruction enhances the use of the information system in real-time applications, such as telemetry. (This instruction executes in NS mode only.)

GTB	774 (0)	Gray-to-Binary Convert

Programmed Fault

DRL	002 (0)	Derail
MME	001 (0)	Master Mode Entry

MISCELLANEOUS INSTRUCTIONS

No Operation

NOP	011 (0)	No Operation
PULS1	012 (0)	Pulse One
PULS2	013 (0)	Pulse Two
SYNC	014 (0)	Gate Synchronize

Repeat Instructions

The RPT and RPD instructions permit execution of the next one or two instructions a selected number of times according to program requirements; they are especially useful for operating upon sequential lists in memory. (The repeat instructions execute only in NS mode.) For example, if RPT is used with any of several compare instructions to search a list, termination occurs when a "hit" is made according to conditions specified in the RPT instruction. The "hit" causes transfer to the next sequential instruction.

RPD	560 (0)	Repeat Double
RPL	500 (0)	Repeat Link
RPT	520 (0)	Repeat

Pointer And Length Instructions

LPL	467 (1)	Load Pointer and Length
SPL	447 (1)	Store Pointer and Length

CODING LIMITATIONS

CODING LIMITATIONS

Supplementary specification items and notes relating to the software that operates in the DPS 8000 is provided below.

1. Result of Fault Detection in the MLR/MRL instruction

   When an SCL1/SCL2/BND fault is detectd in the MLR/MRL instruction, the last several words (up to four words) preceding the fault may not be stored into memory.

2. Tally Runout Indicator

   If any instruction involving a tally word causes the tally count to be zero and sets the tally runout indicator to OFF, and a page fault subsequently occurs in this execution of this instruction, the value of the tally runout indicator in the safe store frame will represent the state of the indicator prior to the instructions. This permits the instruction to be retried. The value of the tally runout in the indicator register will indicate OFF.

3. Interrupt and Fault Entry Descriptor Locations

   The software-visible, fixed absolute memory locations for the interrupt and fault entry descriptors are defined by firmware values. These locations may be altered corresponding to the ECS firmware loaded into a CPU.

   The current entry descriptor locations are as follows:

Entry Descriptors	Word Location
Interrupt	30_8 - 31_8
Fault	32_8 - 33_8
System Entry (PMME)	34_8 - 35_8
Backup Fault	40_8 - 41_8

   The word location range available for these entry descriptors is $0-77_8$.

CODING LIMITATIONS

4. Timer Related Instructions

Instructions which store the timer register affect this value because the timer is stopped for one cycle. These instructions are

STT

CLIMB

DIS when PATROL is enabled

SECTION 8

MACHINE INSTRUCTION DESCRIPTIONS

FORMAT OF INSTRUCTION DESCRIPTION

Each instruction in the repertoire is described in this section. The descriptions are presented in the formats shown below.

The format for all instructions except vector instructions follows:

MNEMONIC	INSTRUCTION NAME	OPCODE

FORMAT: Figure or figure reference

CODING FORMAT: Text

PROCESSOR MODE: Text

SUMMARY: Text and/or bit transfer equations

EXPLANATION: Text

ILLEGAL ADDRESS MODIFICATIONS: Text

ILLEGAL REPEATS: Text

INDICATORS: Text and/or logic statements

NOTE: Text

EXAMPLE(S): If applicable

Line 1: MNEMONIC, INSTRUCTION NAME, OPCODE

This line has three parts that contain the following:

1. MNEMONIC -- The mnemonic code for the operation field of the assembler statement. The assembler recognizes this character string value and maps it into the appropriate binary pattern when generating the actual object code.

2. INSTRUCTION NAME -- The name of the machine instruction from which the mnemonic was derived.

3. OPCODE -- The octal value of the operation code for the instruction. A 0 or a 1 in parentheses following an octal code indicates whether bit 27 (opcode extension bit) of the instruction word is OFF or ON.

Line 2: FORMAT

The layout and definition of the subfields of the instruction word or words either as a figure or as a reference to a figure.

Line 3: CODING FORMAT

The format to be used in coding the instruction.

Line 4: OPERATING MODES

The modes in which the processor should be to execute the instruction. (Refer to Section 1, "Operating Modes".)

Line 5: SUMMARY

The change in the state of the processor affected by the execution of the instruction described in a short, symbolic form. If reference is made to the state of an indicator, it is the state of the indicator before the instruction is executed.

Line 6: EXPLANATION

In instances where more details are needed than supplied in a concise summary, this section describes how the operation functions.

Line 7: ILLEGAL ADDRESS MODIFICATIONS

A list of those modifiers that cannot be used with the instruction. An Illegal Procedure fault occurs when illegal address modification is used.

Line 8: ILLEGAL REPEATS

A list of the repeat instructions that cannot be used with the instruction.

Line 9: ILLEGAL EXECUTES

A list of operations or conditions that are prohibited with the instruction.

Line 10: INDICATORS

A list of only those indicators whose state can be changed by the execution of the instruction. In most cases, a condition for setting ON as well as one for setting OFF is stated. If only one of the two is stated, then the indicator remains unchanged if the condition is not met. Unless stated otherwise, the conditions refer to the contents of registers existing after instruction execution.

Line 11: NOTES

Notes regarding specific conditions, faults, and exceptions that affect the operation of the instruction upon the data.

Line 12: EXAMPLES

Any coding examples, if required for clarity.

ABBREVIATIONS AND SYMBOLS

The following abbreviations and symbols are used in the descriptions of the machine operations.

Symbol	Meaning
AM	Address register modification
AND	The Boolean connective AND
ARn	Address register n specifier in operand descriptor (n = 0, 1,...,7)
b	The original bit position within a 9-bit character
BOLR	Boolean results (4 bits). The BOLR field is used in bit string operations. The bits specify the resultant octal value for four combinations of two input sources.
:(BOLR):	A Boolean operation defined by the BOLR field
c	The original character position within a data word of 9-bit characters
C( )	The contents of ( ). C(string 1) represents the contents of string 1
C(R)	The complete contents of register R
$C(R)_i$	The contents of bit i of register R
$C(R)_{i-j}$	The contents of bits i through j of register R

Symbol	Meaning
CN	The original character number within the data word referred to by the original data word address
CS	Character set definition, EBCDIC (0) or ASCII (1)
DR	Displacement register (bits 32-35)
F	Bit value specifier (0 or 1) for bit string fill. Used when combining/comparing a short bit string with a long bit string to make the shorter string appear to be the same length as the longer string.
FILL	A character used when moving or comparing a short string of characters to a longer string to make the short string appear to be the same length as the longer string. (See note under MASK.)
GXn	General Index Registers 0,1,...7 (ES Mode only)
I	Program interrupt inhibit bit
ID	Indirect operand descriptor indicator
L	The actual length of the character or bit string, as determined by the register or length (RL) bit in the modification field and by N
LOCSYM	A symbol representing either the address of the operand or the displacement from a base
MASK	Bit pattern used in an instruction word. Each 1 bit in the mask causes that bit position in the two characters not to enter into the comparison (coded as octal digits). NOTE: FILL and MASK are 9-bit fields. When using 6- or 4-bit characters, the character must be right-justified in the 9-bit field.
MBZ	Must be zero
MF*n*	Modification field *n* describing address modification to be performed in operand descriptor *n*: MF1 = modification field 1 (bits 29-35) MF2 = modification field 2 (bits 11-17), if operand descriptor 2 is specified MF3 = modification field 3 (bits 2-8), if operand descriptor 3 is specified
N	Either the number of characters or bits in the data string or a 4-bit code (bits 32-35) that specifies a register that contains the number of characters or bits. (See L above.)

Symbol	Meaning
n	Register designation for those instructions that require a register specification to determine operation code.
NS	If 0, there is no effect upon the operation of the instruction. If 1, there is no effect upon the instruction unless TN = 0 and SX = 00 or 11, in which case (output is supposed to be overpunched sign) the appropriate overpunched sign character will not be placed in the specified field. Instead, the appropriate numeric (0-9) character will be placed in the specified field, independent of whether the calculated sign would have been plus or minus. This results in a no sign output. For other values of TN and SX, the NS bit is ignored. This procedure applies to both EBCDIC and ASCII. This usage of NS is not to be confused with NS used for Normal Segmentation mode.
OP CODE	Operation code field
OR	The Boolean connective OR (symbol V)
P	If P = 0, positive signed 4-bit results are stored with octal 14 as the plus sign If P = 1, positive signed 4-bit results are stored with octal 13 as the plus sign
R1,R2	General index registers, specified in ES mode only for register to register instructions
R_i	The ith bit, character, or byte position of R
R_{i-j}	Bit, character, or byte positions i through j of R
RD	Rounding numeric indicator flag: If RD = 0, no rounding takes place If RD = 1, rounding takes place as the final operation; the stored result is incremented by 1 at the least significant character if the most significant character of the truncated part is 5 or more
REG	Address modification register selection for R-type modification of the operand descriptor address field
RI	Distance between elements of vector data in vector operations
RL	Register or length indicator
RM	Register modification
RN	The register that holds the number of elements of vector data in vector operations
S	Sign and decimal type

Symbol	Meaning
SF	Scaling factor
SX	Sign and scaling
T	Truncation fault enable indicator: If T = 0, the truncation fault is disabled If T = 1, the truncation fault is enabled
TA	A code that defines the type of alphanumeric character used in the data
TAG	Tag field used to control address modification (bits 30-35)
TN	A code that defines which type of numeric character is used in the data
TR	Timer register
VA	Virtual address
Xn	Index Registers (0,1,...7)
XOR	The Boolean connective EXECLUSIVE OR
y	A 15-bit displacement from the address register address (with bit 29 = 1) or 18-bit address (with bit 29 = 0)
Y	The effective word address (18 bits for NS mode and 34-bits for ES mode) to the word level of the designated instruction
Y-pair	A symbol denoting that the effective address Y designates a pair of main memory locations (72 bits) with successive addresses, the smaller address being even. When Y is even, it designates the pair (Y, Y+1); when Y is odd, it designates the pair (Y-1, Y). The main memory location with the smaller (even) address contains the most significant part of a double-word operand or the first of a pair of instructions.
YC	The effective address for character data
YCB	The effective address for bit string data
Z	The temporary pseudo-result of a nonstore comparison operation
-->	Replace(s)
::	Is compared with. C(R) :: C(Y) means C(R) - C((Y)-->C(Z), C(R) and C(Y) unchanged invisible result C(Z) sets zero, negative and carry indicator as indicated in the instruction descriptions

Symbol	Meaning
≠	Not equal
$\overline{\Sigma}$	Sigma sign indicates summary.

COMMON ATTRIBUTES OF INSTRUCTIONS

Illegal Modification

If an illegal modifier is used with any instruction, an illegal procedure fault with a subcode class of illegal modifier occurs.

Parity Indicator

The parity indicator is turned ON at the end of a main memory access that has incorrect parity.

INSTRUCTION WORD FORMATS

Single-Word Instructions

The single-word instruction format is displayed in Figure 8-1.

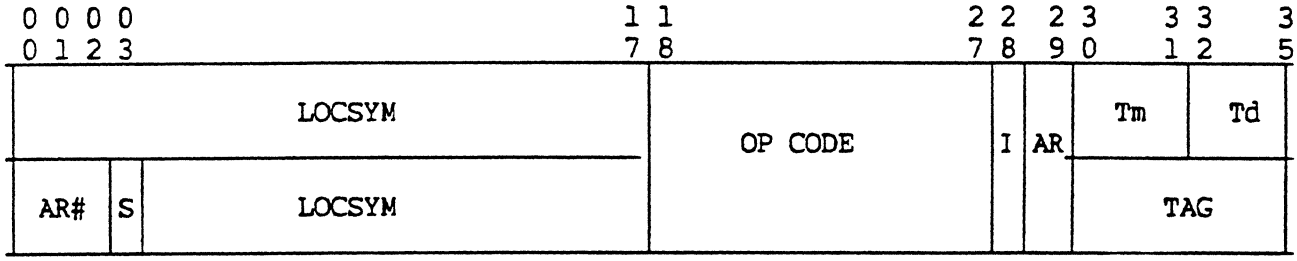


Figure 8-1. Single-Word Instruction Format

CODING FORMATS: 1 8 16 32

```
OPCODE  LOCSYM,RM,AM
OPCODEn LOCSYM,RM,AM          (n = 0,1,...,7)
OPCODE  n,LOCSYM,RM,AM
```

EXAMPLES:

```
                LDA     AB,X3,AR2           Instruction with no index
                                            involved

                LDX1    AB,X3,AR2           Format 1: instruction
                                            with index involved

                LDX     1,AB,X3,AR2         Format 2: instruction
                                            with index involved

          AB    OCT     0
```

AR# - Address register number, if bit 29 = 1.

S - Sign bit, if bit 29 = 1.

LOCSYM - Address field; bits 0-17 or bits 3-17, depending on the state of bit 29

OP CODE - 10-bit operation code field stated as a 3-digit octal number followed by the content of bit 27 (0 or 1) in parentheses

I - Program interrupt inhibit bit

AR - Address register bit. If bit 29 = 1, use address register specified in bits 0, 1, and 2 of Y field for address modification. Bit 3 (sign) is then extended to bits 0, 1, and 2. If bit 29 = 0, no address register modification is performed.

TAG - Tag field; used to control address modification.

Tm - (Bits 30-31) Type of address modification.
Td - (Bits 32-35) Index Register or modification variation designator

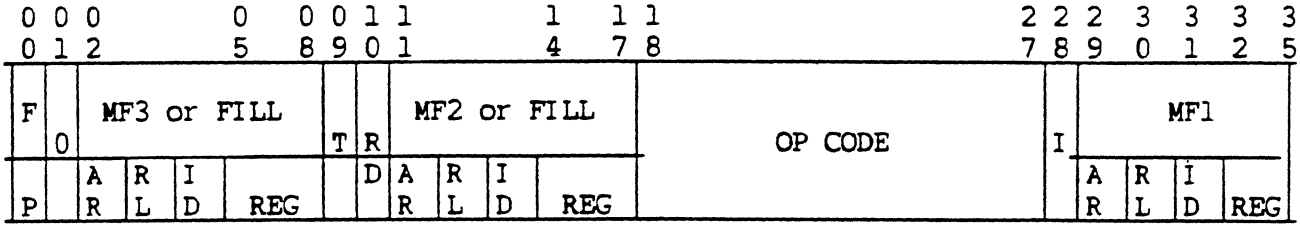
The Repeat (RPT), Repeat Double (RPD), and Repeat Link (RPL) machine instructions and variations of these instructions use special formats and have special tally, terminate, repeat, and other conditions associated with them. (The repeat instructions execute in NS mode only.) There is no address modification for the Repeat instructions. Address modifications for the repeated instructions are limited to R and RI with designators specifying X1,...,X7/GX1, ...,GX7. X0/GX0 is used to control terminate conditions and tally. Address Register (AR) modification is also permitted.

The Character Move and Translate instructions (MTR and MTM) use a variation of the single-word instruction format in which two registers are specified.

Indirect words, used for address modification, have the same general format as the instruction words; however, the fields are used in a somewhat different way.

Multiword Instructions

Alphanumeric, numeric, and bit string multiword instructions have the general machine format described in Figure 8-2.



The number of words and fields within the descriptor words will vary by instruction, but use the following general format.

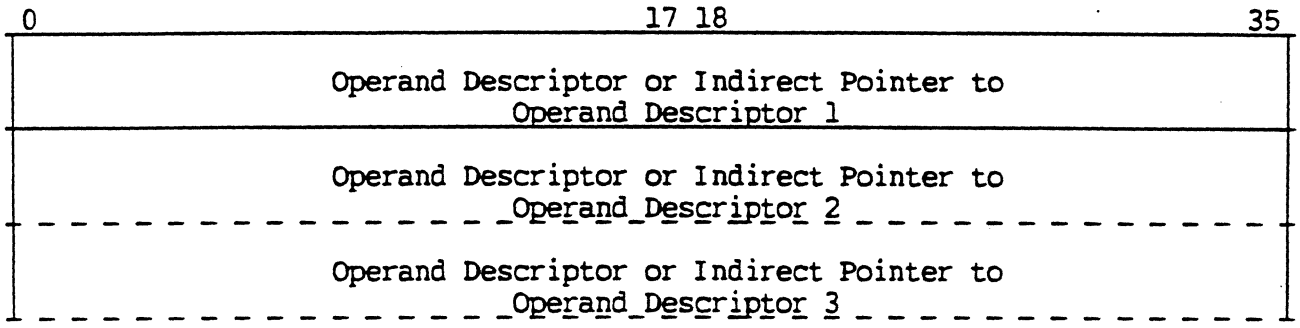


Figure 8-2. Multiword Instruction Format

The fields in the instruction word are defined below. The data fields in the operand descriptor words and the indirect word are discussed in detail in Section 5 under Operand Descriptors and additional detail including coding formats, is provided in Section 7 under Multiword Instructions.

F - Bit value specifier for bit string fill

P - Plus sign indicator (octal 13 or 14)

FILL - Fill character specifier

T - Truncation fault enable indicator

RD - Rounding indicator

MF1 - Modification field 1 (bits 29-35) denotes address modification to be performed for operand descriptor 1. (See "Multiword Modification Field" in Section 7.)

MF2 - Bits 11-17 describe address modification to be performed on this operand for operand descriptor 2

MF3 - Bits 2-8 describe address modification to be performed on this operand for operand descriptor 3

OP CODE - 10-bit operation code field. Octal representation consisting of three octal digits followed by the content of bit 27 (1) in parentheses.

I - Program interrupt inhibit bit

AR - Address register indicator

RL - Register containing length indicator

ID - Indirect operand descriptor indicator

REG - Type of register modification (A, AU, Q, QU, IC, DU, Xn/GXn)

Address Register Special Arithmetic Instructions

These instructions provide the capability for replacing, adding to, or subtracting from the contents of an address register on either a word, character, or bit address basis. The operation is register-to-register, with no memory fetch involved.

The special arithmetic instructions have the format shown in Figure 8-3:

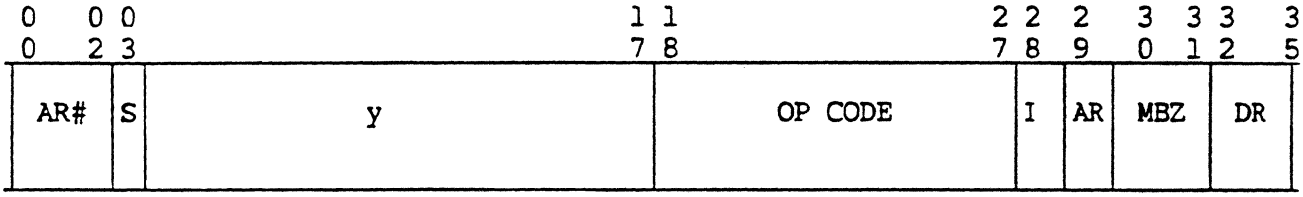


Figure 8-3. Address Register Special Arithmetic Instruction Format

AR# - Selects address register to be altered

S - Sign bit

y - Used as a word displacement (no character or bit position included) along with the contents specified in the DR field to alter the contents of the specified address register. Bit 3 provides negative or positive word displacement.

OP CODE - 10-bit operation code field. Octal representation consisting of three octal digits followed by the content of bit 27 (1) in parentheses.

I - Program interrupt inhibit bit

AR - Address register bit. If bit 29 = 1, the sum of the DR (in characters, words, or bits) and the y field (in words) are added to or subtracted from the contents of the AR specified in bits 0-2. If bit 29 = 0, the described sum or its two's complement is loaded into the AR for addition or subtraction, respectively. If the mnemonic is coded with X (for example, AWDX), bit 29 is forced to zero.

MBZ - Bits 30-31 must be zero. The operand length is contained in the register specified by DR.

DR - Displacement register. Specifies which register contains the displacement value. The register codes and register lengths are the same as those used in MF fields except that IC modification is illegal.

The operations for adding a value to the contents of an address register proceed identically as with effective operand address preparation from an operand descriptor, with the final results stored in the specified address register. The subtract operation differs only in that the contents of the register specified by the code in the DR field are first added to the y field. This result is then subtracted from the actual contents of the address register or from the implied zero contents and the result is placed in the address register. The codes for DU, DL, and IC are illegal for the DR field and cause an IPR fault.

No indicators are affected by these instructions.

Character Move To/From Register Instructions

Two instructions permit moves of one, two, three, or four 9-bit characters from a memory location to a register or from a register to memory. These instructions have the format shown in Figure 8-4.

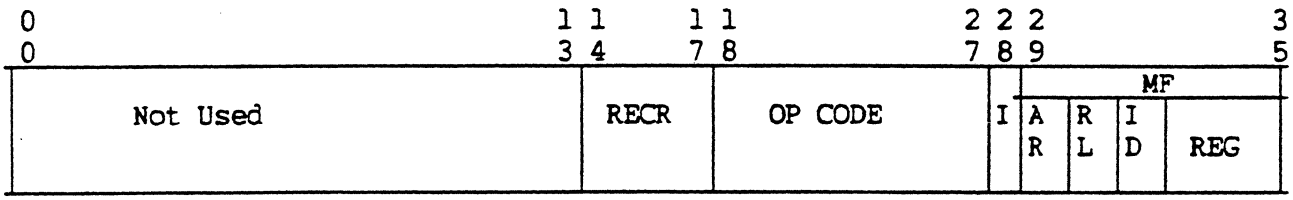


Figure 8-4. Character Move To/From Register Instruction Format

RECR - Specifies the register to which characters are moved (MTR), or from which characters are moved (MTM). (Refer to MTR/MTM instructions.)

OP CODE - 10-bit operation code field. Octal representation consisting of three octal digits followed by the content of bit 27 (1) in parentheses.

I - Program interrupt inhibit bit

AR - Address register indicator

RL - This field is ignored

ID - Indirect operand descriptor indicator

REG - Type of register modification (A, AU, Q, QU, IC, DU, Xn/GXn)

These instructions move one, two, three, or four 9-bit characters from (MTR) or to (MTM) a memory location to or from a register specified by the RECR field.

Register-to-Register Instructions

Register to Register instructions known as "RR" type instructions are valid only in the ES mode. An attempt to execute these instructions in the NS mode results in an IPR fault. RR type instructions permit movement, arithmetic operation, and shift of fixed-point data using the GXn, A and Q registers. An attempt to execute any RR type instruction by the RPT, RPD, or RPL instructions results in an IPR fault. The format for register to register instructions is shown in Figure 8-5.

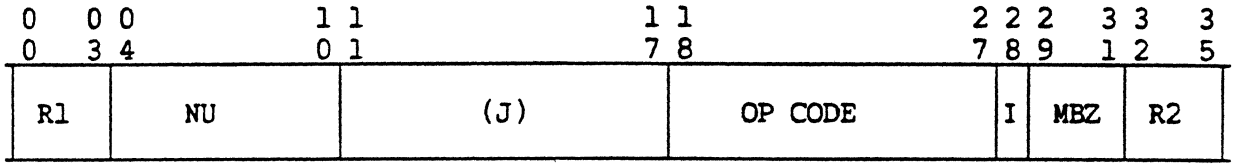


Figure 8-5. Register To Register Instruction Format

Bits	Field	Description
0 - 3	R1	A code indicating a register to be the destination of the result. The allowable codes follow:

Register Code	Result
0000	IPR
0001	IPR
0010	IPR
0011	IPR
0100	IPR
0101	A
0110	Q
0111	IPR
1000	GX0
1001	GX1
1010	GX2
1011	GX3
1100	GX4
1101	GX5
1110	GX6
1111	GX7

4 -10 NU Not used. Should be set to 0.

11-17 J Used only in a shift instruction. Specifies the shift number (immediate value). Must be 0 in all but shift instructions.

18-27 OP Operation code

28 I Interrupt inhibit bit

29-31 MBZ Must be zero or an IPR fault occurs

32-35 R2 A code indicating the source register. The codes for this register are the same as for R1.

NOTES:

1. Specifying a register code of 0000 in a shift instruction does not result in an IPR fault.

2. If a register pair appears in an instruction specification, the two registers are handled as linked. The list below indicates the register codes to be assocciated with the register pair.

Register Code	Result
0000	IPR
0001	IPR
0010	IPR
0011	IPR
0100	IPR
0101	A, Q
0110	A, Q
0111	IPR
100x	GX0, GX1
101x	GX2, GX3
110x	GX4, GX5
111x	GX6, GX7

where x means this bit is ignored by the hardware.

INSTRUCTION REPERTOIRE

The processor interprets a 10-bit field of the instruction word as the operation code. This field size yields 1024 possible instructions codes of which over half are implemented.

Detailed on the following pages are the processor instructions and operation codes sorted alphabetically on the mnemonic by function.

A4BD
A4BDX

A4BD A4BDX	Add 4-Bit Displacement to Address Register	502 (1)

FORMAT: Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

```
1       8       16
_____________________

        {A4BD }
        {A4BDX}  word displacement,R,AR
```

When the mnemonic is coded with an "X" (A4BDX), bit 29 is forced to zero.

OPERATING MODES: Any

EXPLANATION: NS Mode

The count of 4-bit characters contained in the register specified by the DR field is effectively divided by 8, producing a word count and a character count. The word count is added to the y field (bit 3 extended).

If bit 29 = 0, this sum replaces bits 0-17 of the specified AR, with the character count (from the divide) translated into bit string representation and replacing bits 18-23 of AR.

If bit 29 = 1, the sum of the word count (from the divide) and y field is added to bits 0-17 of the specified AR. The CHAR and BIT portions (bits 18-23) of the specified AR are forced to point to a 4-bit character boundary in bit string representation. The resulting character count is added to the character count from the divide operation, with the result being translated back into bit string representation. These formed values for the WORD, CHAR, and BIT fields are stored in bits 0-23 of the specified AR. With this addition, carry from the CHAR field is transferred to the WORD field.

ES Mode

The count of 4-bit characters contained in the register specified by the DR field is effectively divided by 8, producing a word count and a character count. The word count is added to the y field (bit 3 extended).

A4BD
A4BDX

If bit 29 = 0, this sum replaces bits 0-29 of the specified AR, with the character count (from the divide) translated into bit string representation and replacing bits 30-35 of AR.

IF bit 29 = 1, the sum of the word count (from the divide) and y field is added to bits 0-29 of the specified AR. The CHAR and BIT portions (bits 30-35) of the specified AR are forced to point to a 4-bit character boundary. The resulting character count is added to the character count from the divide operation, with the result translated back into bit string representation. These formed values for the WORD, CHAR, and BIT fields are stored in bits 0-35 of the specified AR. With this addition, carry from the CHAR field is transferred to the WORD field.

Effectively, the two bit string representations are added and the result is translated back to a format allowing 2 bits to represent the characters and 4 bits to represent bits. Any overflow of the 2 bits increments the address field and the 4-bit field is handled as mod-9. Any overflow of the 2-bit field increments the character (2-bit) field.

ILLEGAL ADDRESS MODIFICATIONS: When DU, DL, and IC are specified in the DR.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLES: (Applies to NS mode only)

```
1       8       16              32
_____________________________________________________

        EAX3    9
        A4BDX   2,3,5           AR5 octal contents  -  0 0 0 0 0 3 0 5
        A4BD    0,3,5           AR5 octal contents  -  0 0 0 0 0 4 2 0


        EAX4    6
        A4BDX   0,4,3           AR3 octal contents  -  0 0 0 0 0 0 6 0
        EAX5    9
        A4BD    4,5,3           AR3 octal contents  -  0 0 0 0 0 5 6 5
```

A6BD
A6BDX

A6BD A6BDX	Add 6-Bit Displacement to Address Register	501 (1)

FORMAT: Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT: 1 8 16

```
{A6BD }
{A6BDX} word displacement,R,AR
```

When the mnemonic is coded with an X (A6BDX), bit 29 is forced to zero.

OPERATING MODES: Any

EXPLANATION: NS Mode

The count of 6-bit characters contained in the register specified by the DR field is effectively divided by 6, producing a word count and a character count. The word count is added to the y field (bit 3 extended).

If bit 29 = 0, this sum replaces bits 0-17 of the specified AR, with the character count (from the divide) being translated into bit string representation and replacing bits 18-23 of AR.

If bit 29 = 1, the sum of the word count (from the divide) and y field is added to bits 0-17 of the specified AR. The CHAR and BIT portions (bits 18-23) of the specified AR are forced to point to a 6-bit character boundary. The resulting 6-bit character count is added to the character count from the divide operation, with the result being translated back into bit string representation. These formed values for the WORD, CHAR, and BIT fields are stored in bits 0-23 of the specified AR. With this addition, carry from the CHAR field (when carry + character count > 5) is transferred to the WORD field.

A6BD
A6BDX

ES Mode

The count of 6-bit characters contained in the register specified by the DR field is effectively divided by 6, producing a word count and a character count. The word count is added to the y field (bit 3 extended).

If bit 29 = 0, this sum replaces bits 0-29 of the specified AR, with the character count (from the divide) translated into bit string representation and replacing bits 30-35 of AR.

If bit 29 = 1, the sum of the word count (from the divide) and y field is added to bits 0-29 of the specified AR. The CHAR and BIT portions (bits 30-35) of the specified AR are forced to point to a 6-bit character boundary. The resulting 6-bit character count is added to the character count from the divide operation, with the result translated back into bit string representation. These formed values for the WORD, CHAR, and BIT fields are stored in bits 0-35 of the specified AR. With this addition, carry from the CHAR field (when carry + character count > 5) is transferred to the WORD field.

ILLEGAL ADDRESS MODIFICATIONS: When DU, DL, or IC are specified in DR.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None Affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

EXAMPLES: (Applies to NS mode only)

```
1       8       16                  32
        EAX2    8
        A6BDX   3,2,6               AR6 octal contents  -  0 0 0 0 0 4 2 3
        A6BD    2,2,6               AR6 octal contents  -  0 0 0 0 0 7 4 6

        EAX4    15
        A6BDX   0,4,7               AR7 octal contents  -  0 0 0 0 0 2 4 0
        A6BD    2,4,7               AR7 octal contents  -  0 0 0 0 0 7 0 0
```

A9BD
A9BDX

A9BD A9BDX	Add 9-Bit Displacement to Address Register	500 (1)

FORMAT: Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT: 1 8 16

```
{A9BD }
{A9BDX}  word displacement,R,AR
```

When the mnemonic is coded with an X (A9BDX), bit 29 is forced to zero.

OPERATING MODES: Any

EXPLANATION: NS Mode

The count of 9-bit characters contained in the register specified by the DR field is effectively divided by 4, producing a word count and a character count. This word count is then added to the y field (bit 3 extended).

If bit 29 = 0, the resulting sum of the word addresses and the character count (from the divide operation) replaces bits 0-19 of the specified AR.

If bit 29 = 1, the resulting sum of the word addresses is added to bits 0-17 of the specified AR and the character count (from the divide operation) is added to bits 18-19 of C(AR). These results are then stored in bits 0-19 of the specified AR. In either case, bits 20-23 of the specified AR are zeroed. Carry is transferred from bit 18 to bit 17 with this addition.

ES Mode

The count of 9-bit characters contained in the register specified by the DR field is effectively divided by 4, producing a word count and a character count. This word count is then added to the y field (bit 3 extended).

If bit 29 = 0, the resulting sum of the word addresses and the character count (from the divide operation) replaces bits 0-31 of the specified AR.

A9BD
A9BDX

If bit 29 = 1, the resulting sum of the word addresses is added to bits 0-29 of the specified AR and the character count (from the divide operation) is added to bits 30-31 of C(AR). These results are then stored in bits 0-31 of the specified AR. In either case, bits 32-35 of the specified AR are zeroed. Carry is transferred from bit 30 to bit 29 with this addition.

ILLEGAL ADDRESS MODIFICATIONS: When DU, DL, or IC are specified in the DR.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

EXAMPLES: (Applies to NS mode only)

```
1       8       16                  32
        EAX1    6
        A9BDX   2,1,2               AR2 octal contents - 0 0 0 0 0 3 4 0
        A9BD    2,,2                AR2 octal contents - 0 0 0 0 0 5 4 0

        EAX2    15
        A9BDX   4,2,6               AR6 octal contents - 0 0 0 0 0 7 6 0
        A9BD    0,2,6               AR6 octal contents - 0 0 0 0 1 3 4 0
```

AARn	Alphanumeric Descriptor To Address Register n	56n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        AARn    LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY: For n = 0, 1, ... or 7 as determined by op code

$C(Y)_{0-17}$ --> $C(ARn)_{0-17}$

$C(Y)_{18-20}$ translated $C(ARn)_{18-23}$

EXPLANATION: The alphanumeric descriptor is fetched from the computed effective address Y. The TA field, bits 21 and 22, is examined to determine the type of data described. If the TA code indicates 9-bit character data, bits 18 and 19 of the descriptor CN field go to the corresponding bit positions of ARn and zeros fill bits 20-23 of ARn. If the TA code indicates 6- or 4-bit character data, the descriptor CN field is appropriately translated into bit string representation and goes to bits 18-23 of ARn. In all cases, the word portion of the fetched descriptor is placed in the word portion (bits 0-17) of ARn.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: If this instruction is executed in ES mode.

INDICATORS: None affected

NOTES:
1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used or if the descriptor TA field contains code 11.

2. An IPR fault occurs if descriptor CN field contains xx1 for TA = 00, or 11x for TA = 01.

3. An IPR fault occurs if an attempt is made to execute this instruction in the ES mode.

EXAMPLES: (Applies to NS mode only)

```
1       8       16                  32
        AAR4    DESCR               load data string address into AR4
         .       .                  memory contents in octal
         .       .
         .       .
         .       .
DESCR   ADSC9   FLD1,3,1             001023600001 - descriptor
                                    AR4 octal contents - 0 0 1 0 2 3 6 0
```

ABD
ABDX

ABD ABDX	Add Bit Displacement to Address Register	503 (1)

FORMAT: Special arithmetic instruction format (see Figure 8-3).

CODING FORMAT:

1 8 16

{ABD }
{ABDX} word displacement,RM,AR

When the mnemonic is coded with an X (ABDX), bit 29 is forced to zero.

OPERATING MODES: Any

EXPLANATION: NS Mode

The bit string count in the register specified in the DR field is divided by 36. The quotient is taken as the word count and the remainder is taken as the bit count. The word count is added to the y field for which bit 3 of the instruction word is extended and the sum is taken.

If bit 29=0, the sum is loaded into bits 0-17 of the specified AR, and the character portion and the bit portion of the remainder are loaded into bits 18-23 of the specified AR.

If bit 29=1, the sum is added to bits 0-17 of the specified AR. The CHAR and BIT fields (bits 18-23) of the specified AR are added to the character portion and the bit portion of the remainder. WORD, CHAR and BIT fields generated in this manner are loaded into bits 0-23 of the specified AR. With this addition, carry from the BIT field (bit 20) and the CHAR field (bit 18) is transferred (when BIT field >8, CHAR field >3).

ES Mode

The bit string count in the register specified in the DR field is divided by 36. The quotient is taken as the word count and the remainder is taken as the bit count. The word count is added to the y field for which bit 3 of the instruction word is extended and the sum is taken.

If bit 29=0, the sum is loaded into bits 0-29 of the specified AR, and the character portion and the bit portion of the remainder are loaded into bits 30-35 of the specified AR.

If bit 29=1, the sum is added to the sign extended value of bits 0-29 of the specified AR. The CHAR and BIT fields (bits 30-35) of the specified AR are added to the character portion and the bit portion of the remainder. WORD, CHAR, and BIT fields generated in this manner are loaded into bits 0-35 of the specified AR. With this addition, carry from the BIT field (bit 30) and the CHAR field (bit 32) is transferred (when BIT field >8, CHAR field >3).

ILLEGAL ADDRESS MODIFICATIONS: When DU, DL, or IC are specified in the DR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

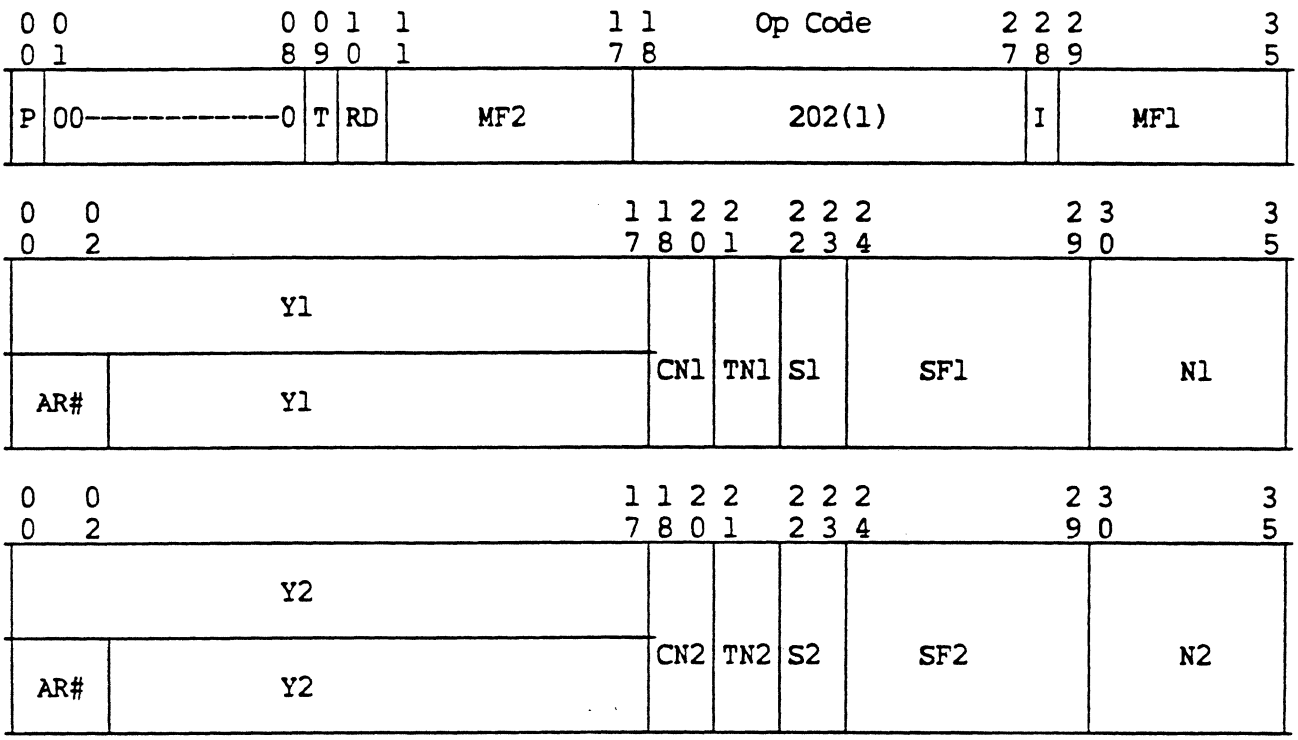
NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLES: (Applies to NS mode only)

1	8	16	32
	EAX6	85	
	ABDX	7,6,2	AR2 octal contents - 0 0 0 0 1 1 2 4
	ABD	2,6,2	AR2 octal contents - 0 0 0 0 1 5 5 0
	EAX1	74	
	EAX2	30	
	ABDX	4,1,3	AR3 octal contents - 0 0 0 0 0 6 0 2
	ABD	0,2,3	AR3 octal contents - 0 0 0 0 0 6 6 5

AD2D	Add Using Two Decimal Operands	202 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
AD2D    MF1),(MF2),RD,P,T
NDSCn   LOCSYM,CN,N,S,SF,AM
NDSCn   LOCSYM,CN,N,S,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) + (string 1) --> C(string 2)

Same as AD3D, except that the sum is stored using YC2, TN2, S2 and, if S2 indicates a scaled format, SF2.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Same as for AD3D

NOTES:

1. All notes for AD3D apply also to AD2D.
2. Illegal Procedure fault same as for MVN.
3. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLES:

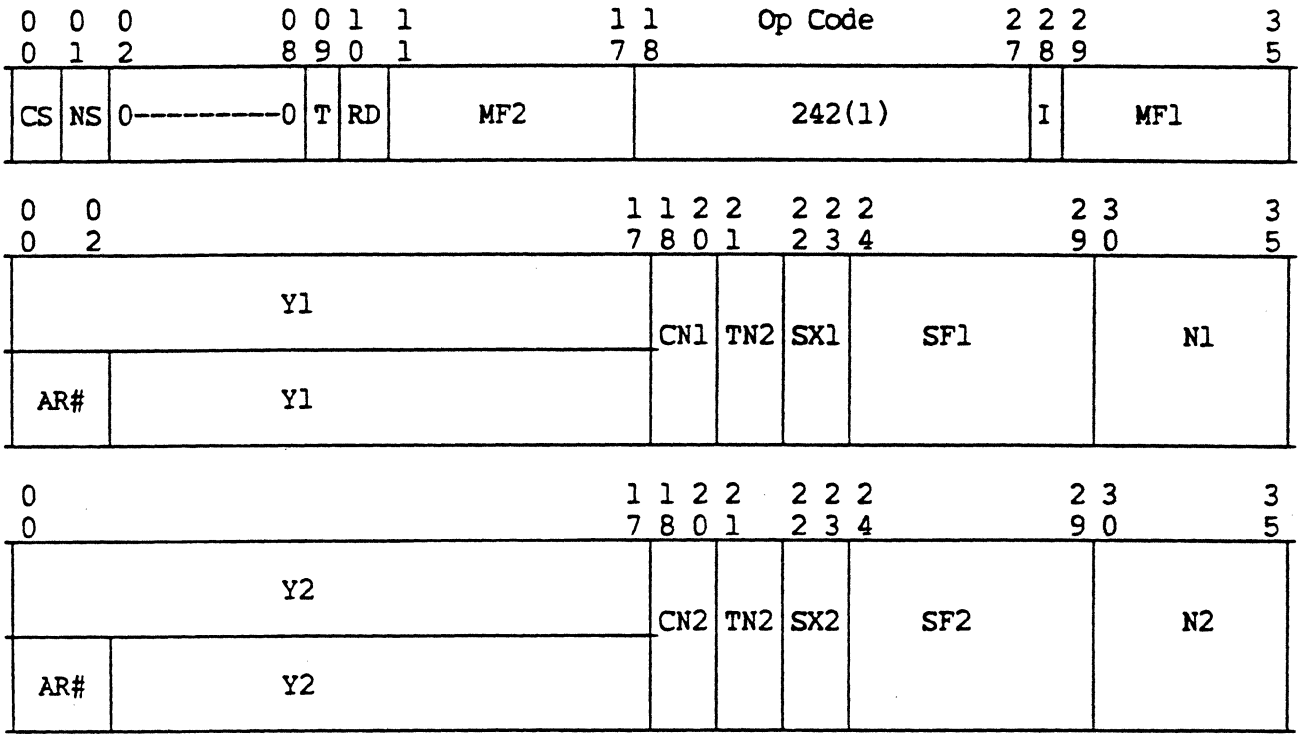
```
1       8       16                  32
        AD2D    ,,,,1               with truncation enable option
        NDSC4   FLD1,0,8,2,-2       FLD1 addend operand descriptor
        NDSC9   FLD2,0,6            FLD2 addend operand descriptor
        USE     CONST.              memory contents
FLD1    EDEC    8P123456+           0 1 2 3 4 5 6 +
FLD2    EDEC    6A+1E+2             + 0 0 0 1 2
        USE                         + 1 3 3 4 0  (Sum) (truncation fault)

        AD2D    ,,,1                with plus sign octal 13 option
        NDSC9   FLD1,0,4            FLD1 addend operand descriptor
        NDSC4   FLD2,1,7,2,-4       FLD2 addend operand descriptor
        USE     CONST.              memory contents
FLD1    EDEC    4A+99.              + 9 9 0
FLD2    EDEC    8P123456+           0 1 2 3 4 5 6 +
        USE                         0 1 1 3 4 5 6 +  (Sum) (overflow fault)
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
 1      8       16                32
_____________________________________________________________________________

        EAX1    1                 load character modifier into X1
        EAX7    7                 load FLD1 length into X7
        EAX4    FLD1              load FLD1 address into X4
        AWDX    0,4,4             put FLD1 address into AR4
        AD2D    (1,1,,X1),(,,1),1,1  rounding and plus sign options
        NDSC4   0,,X7,2,-2,4      FLD1 operand descriptor (FLD1,1,7,2,-2)
        NDSC9   INDSC2            pointer to FLD2 indirect operand descriptor

        USE     CONST.            memory contents
FLD1    EDEC    8P123450-         0 1 2 3 4 5 0 -
FLD2    EDEC    8A+9876E+2        + 0 0 9 8 7 6 2
INDSC2  NDSC9   FLD2,0,8          FLD2 indirect operand descriptor
        USE                       + 9 8 6 3 6 6 0  (Sum)
```

AD2DX	Add Using Two Decimal Operands Extended	242 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
AD2DX   (MF1),(MF2),RD,CS,T,NS
NDSCn   LOCSYM,CN,N,SX,SF,AM
NDSCn   LOCSYM,CN,N,SX,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) + C(string 1) --> C(string 2)

EXPLANATION: The decimal number of data type TN1, sign and decimal type SX1, and starting location YC1, is added to the decimal number of data type TN2, sign and decimal type SX2, and starting location YC2. The sum is stored starting in location YC2 as a decimal number of data type TN2 and sign and decimal type SX2.

- o If SX2 indicates a fixed-point format, the results are stored using scale factor SF2, which causes leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most significant digit overflow or least significant digit truncation to occur.
- o If SX2 indicates a floating-point format, the result is right-justified to preserve the most significant nonzero digits even if this causes least significant truncation.
- o The character set is defined by CS. Placement of an overpunched sign in the output is controlled by NS. (Refer to the introductory pages of this section for definition of NS.) If RD is 1, rounding takes place prior to storage. Provided that strings 1 and 2 are not overlapped, the contents of the decimal number that starts in location YC1 remains unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero - If result equals zero, then ON; otherwise, OFF

Negative - If result is negative, then ON; otherwise, OFF

Truncation - If in the preparation of the final result, one or more least significant digits (zero or nonzero) are lost and rounding is not specified, then ON; otherwise (i.e., no least significant digits lost or rounding specified), OFF.

Overflow - If data is lost in most significant positions, then ON; otherwise, unchanged

Exponent Overflow - If exponent of floating point result > 127, then ON; otherwise, unchanged

AD2DX

Exponent Underflow	If exponent of floating point result < - 128, then ON; otherwise, unchanged

NOTES:

1. Truncation fault occurs if the truncation indicator is set and the truncation fault enable (T) bit is a 1.

2. Illegal procedure faults occur when

   a. DU or DL modification in MF1 or MF2.

   b. The sign and numeric digits contains an unpermitted code.

   c. Though the operand descriptor indicates the presence of a sign or exponent, the value of N_1 or N_2 does not contain the number of characters required for the sign and exponent (when at least one digit is required).

   d. An illegal repeat is used.

3. Independent of the data type being used, either packed decimal or 9-bit numeric, floating point or fixed-point, significant digits of the result may be lost if the result field as defined by the result descriptor is not large enough to contain the calculated result after it has been aligned.

4. If an illegal digit or sign is detected, part or all of the receiving field may be changed before the IPR fault occurs.

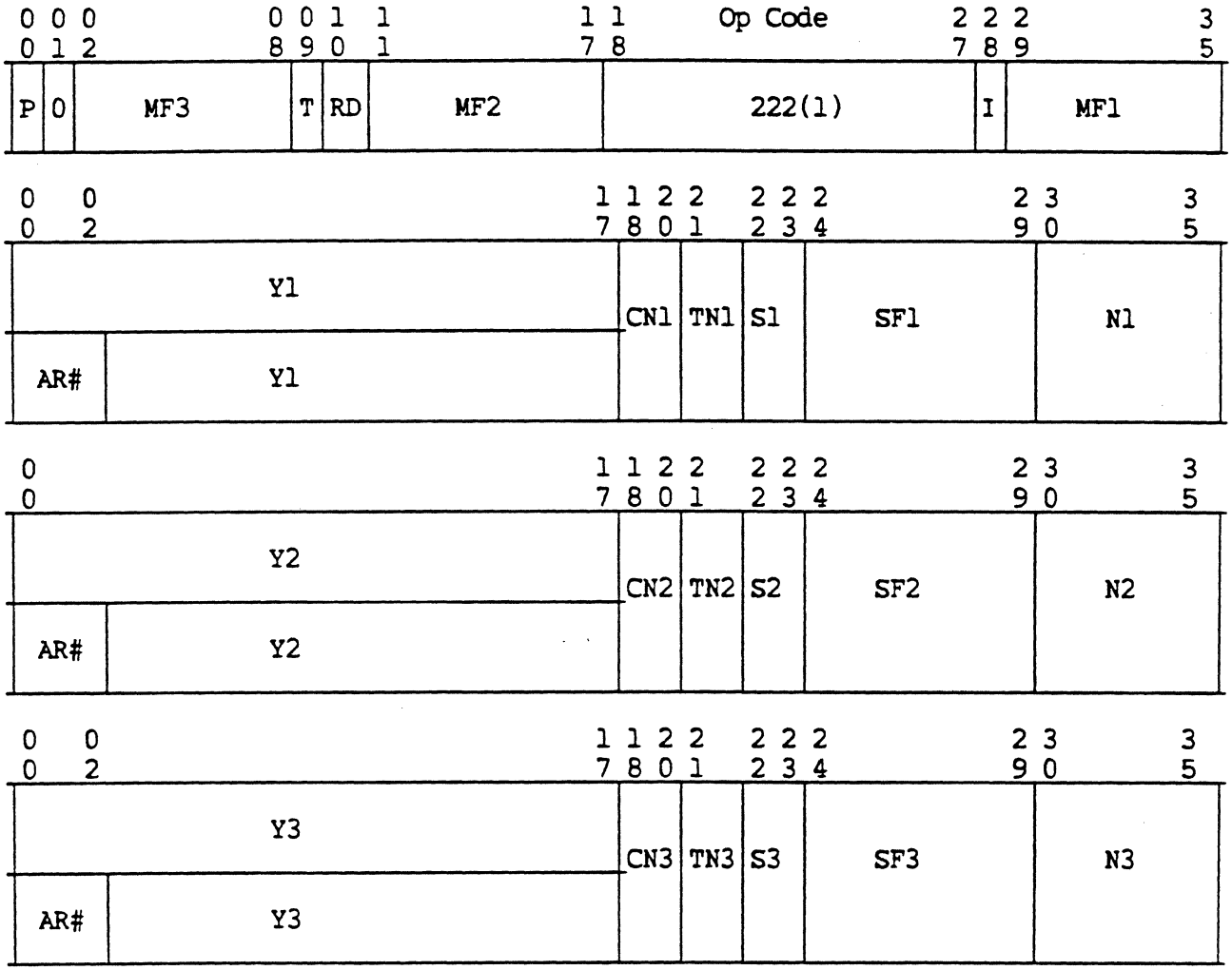
5. All notes for AD3D apply to AD2DX.

6. Refer to the specifications on MVNX for information on coding of overpunched signs.

7. An Illegal Procedure fault occurs if illegal address modification is used.

AD3D	Add Using Three Decimal Operands	222 (1)

FORMAT:



CODING FORMAT: The AD3D instruction is coded as follows:

```
1       8       16
        AD3D    (MF1),(MF2),(MF3),RD,P,T
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) + C(string 1) --> C(string 3)

EXPLANATION: The decimal number of data type TN1, sign and decimal type S1, and starting location YC1, is added to the decimal number of data type TN2, sign and decimal type S2, and starting location YC2. The sum is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type S3.

If S3 indicates a fixed-point format, the results are stored using scale factor SF3, which causes leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most significant digit overflow or least significant digit truncation to occur.

If S3 indicates a floating-point format, the result is right-justified to preserve the most significant nonzero digits even if this causes least significant truncation.

If P = 1, positive signed 4-bit results are stored using octal 13 as the plus sign. If P = 0, positive signed 4-bit results are stored with octal 14 as the plus sign. If RD is 1, rounding takes place prior to storage.

Provided that strings 1, 2, and 3 are not overlapped, the contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

The zero indicator is set when the decimal number is zero; it does not indicate the case in which all bits are zero.

If the result is given by a fixed-point, operations are performed by justifying the scaling factors (SF1, SF2, and SF3) of the operands 1, 2, and 3 as follows:

If SF1 > SF2

SF1 > SF2 >= SF3 --> Justify to SF2

SF3 > SF1 > SF2 --> Justify to SF1

SF1 >= SF3 > SF1 --> Justify to SF3 - 1

If SF2 > SF1

SF2 > SF1 >= SF3 --> Justify to SF1

SF3 > SF2 > SF1 --> Justify to SF2

SF2 >= SF3 > SF1 --> Justify to SF3 - 1

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS: RPT,RPD, RPL

INDICATORS:

Zero - If result equals zero, then ON; otherwise, OFF

Negative - If result is negative, then ON; otherwise, OFF

Truncation - If, in the preparation of the final result, one or more least significant digits (zero or nonzero) are lost and rounding is not specified, then ON. Otherwise (i.e., no least significant digits lost or rounding is specified), OFF

Exponent Overflow - If exponent of floating-point result is > 127, then ON; otherwise, unchanged

Exponent Underflow - If exponent of floating point result < - 128, then ON; otherwise, unchanged

Overflow - If data is lost in most significant positions, then ON; otherwise, unchanged

NOTES:

1. Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is 1.

2. Illegal procedure faults occur when

   a. DU or DL modification in MF1 or MF2.

   b. The sign and numeric digits contains an unpermitted code.

   c. Though the operand descriptor indicates the presence of a sign or exponent, the value of N_1 or N_2 does not contain the number of characters required for the sign and exponent (when at least one digit is required).

3. Independent of the data type being used (either packed decimal or 9-bit numeric floating-point or scaled) significant digits in the result may be lost if:

   a. The difference between the scaling factors (exponents) of the source operands is large enough to cause the expected length of the intermediate result to exceed 63 digits after decimal point alignment of source operands, followed by addition.

   b. The result field as defined by the result descriptor is not large enough to contain the calculated result after it has been aligned.

4. If an illegal digit or sign is detected, part or all of the receiving field may be changed before the IPR fault occurs.

EXAMPLES:

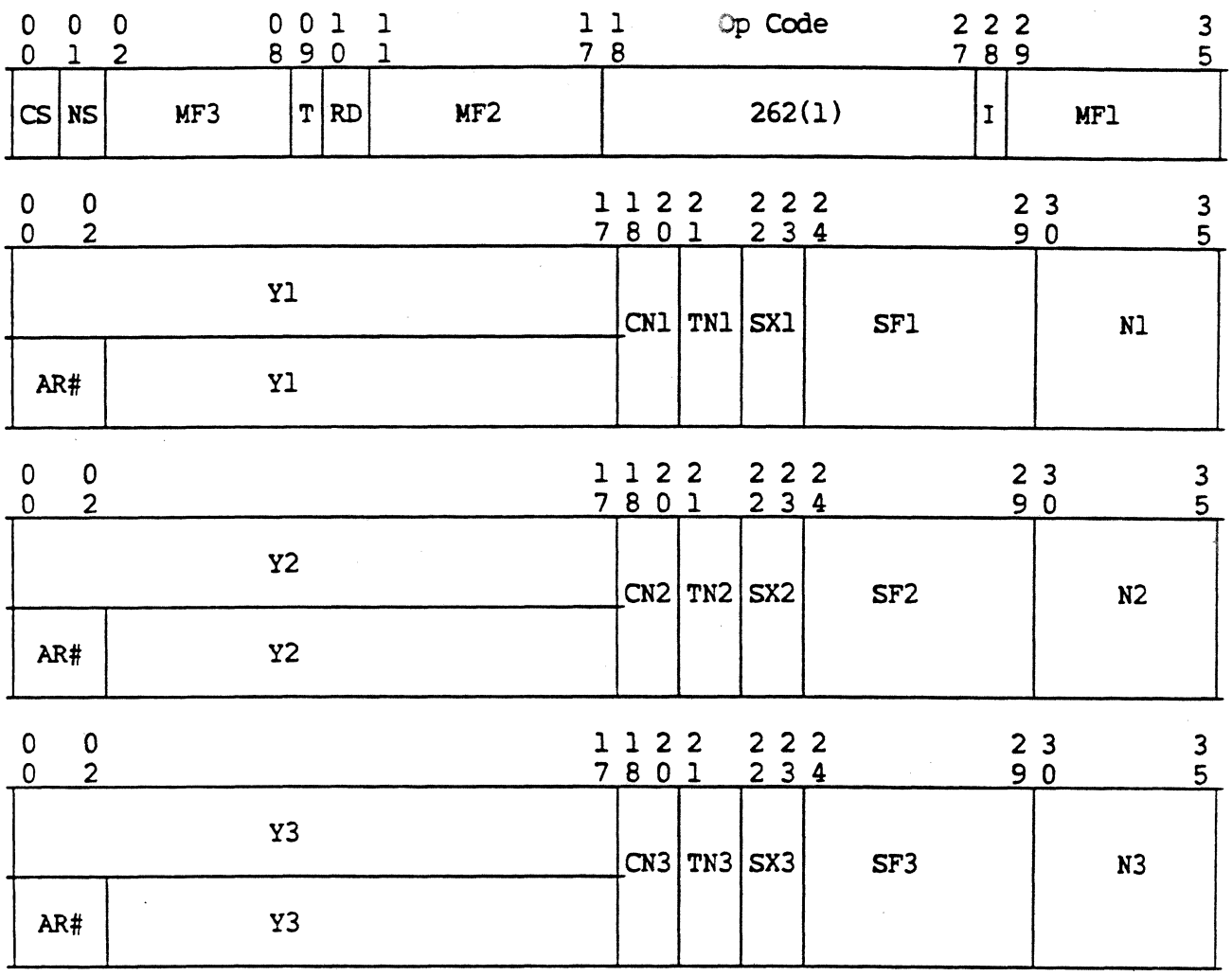
```
1       8       16              32
        AD3D    ,,,1,1          with rounding and plus sign options
        NDSC9   FLD1,0,4,3,-2   FLD1 addend operand descriptor
        NDSC9   FLD2,0,8,2,-2   FLD2 addend operand descriptor
        NDSC4   FLD3,2,6,1      operand descriptor for sum field
        USE     CONST.          memory contents
FLD1    EDEC    4A1234          1 2 3 4
FLD2    EDEC    8A654321+       0654321+
FLD3    BSS     1               xx+06556   (Sum)
        USE                     instruction fault?   no
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16              32

        EAX2    2               load character modifier into X2
        EAX6    6               load FLD1 length into X6
        EAX4    FLD1            load FLD1 address into X4
        AWDX    0,4,4           put FLD1 address into AR4
        AD3D    (1),(,1,,X2),(,,1),1,1
        NDSC9   0,0,4,          FLD1 operand descriptor (FLD1,0,4,0)
        NDSC4   FLD2,,X6,3,-2   FLD2 operand descriptor (FLD2,2,6,3,-2)
        ARG     DFLD3           pointer to FLD3 operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    4A-12E+2        - 1 2 2
FLD2    EDEC    8P123456        00123456
FLD3    BSS     1               xxx+0346   (Sum)
DFLD3   NDSC4   FLD3,3,5,1,-1   FLD3 sum operand descriptor
        USE                     instruction fault?   no
```

AD3DX	Add Using Three Decimal Operands Extended	262 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
        AD3DX   (MF1),(MF2),(MF3),RD,CS,T,NS
        NDSCn   LOCSYM,CN,N,SX,SF,AM
        NDSCn   LOCSYM,CN,N,SX,SF,AM
        NDSCn   LOCSYM,CN,N,SX,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) + C(string 1) --> C(string 3)

EXPLANATION: The decimal number of data type TN1, sign and decimal type SX1, and starting location YC1, is added to the decimal number of data type TN2, sign and decimal type SX2, and starting location YC2. The sum is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type SX3.

If SX3 indicates a fixed-point format, the results are stored using scale factor SF3, which causes leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most significant digit overflow or least significant digit truncation to occur.

If SX3 indicates a floating-point format, the result is right-justified to preserve the most significant nonzero digits even if this causes least significant truncation. The character set is defined by CS. Placement of overpunched sign in the output is controlled by NS. (Refer to the introductory pages of this section for definition of NS.) If RD is 1, rounding takes place prior to storage. Provided that strings 1, 2, and 3 are not overlapped, the contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2 and MF3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Zero - If result equals zero, then ON; otherwise, OFF

Negative - If result is negative, then ON; otherwise, OFF

Truncation - If, in the preparation of the final result, one or more least significant digits (zero or nonzero) are lost and rounding is not specified, then ON. Otherwise (i.e., no least significant digits lost or rounding specified), OFF.

Overflow - If data is lost in most significant positions, then ON; otherwise, unchanged.

Exponent Overflow - If exponent of floating-point result > 127, then ON; otherwise, unchanged.

Exponent Underflow If exponent of floating point result < - 128, then ON; otherwise, unchanged

NOTES:

1. Truncation fault occurs if the truncation indicator is set and the truncation fault enable (T) bit equals 1.

2. Illegal procedure faults occur when:

   a. DU or DL modification in MF1 or MF2.

   b. The sign and numeric digits contains an unpermitted code.

   c. Though the operand descriptor indicates the presence of a sign or exponent, the value of N_1 or N_2 does not contain the number of characters required for the sign and exponent (when at least one digit is required).

3. Independently of the data type being used (either packed decimal or 9-bit numeric, floating-point or scaled) significant digits of the result may be lost if the result field as defined by the result descriptor is not large enough to contain the actual calculated result after it has been aligned.

4. If an illegal digit or sign is detected, part or all of the receiving field may be changed before the IPR fault occurs.

5. For coding of overpunched signs, refer to MVNX.

6. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ADA	Add to A-Register	075 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(A) + C(Y) --> C(A); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of A is exceeded, then ON

Carry - If a carry out of bit 0 of C(A) is generated, then ON; otherwise, OFF

ADAQ	Add to AQ-Register	077 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(AQ) + C(Y-pair) --> C(AQ); C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of AQ is exceeded, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if an illegal address modification is used.

ADE	Add to Exponent Register	415 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: $C(E) + C(Y)_{0-7} \rightarrow C(E)$

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	-	Set OFF
Negative	-	Set OFF
Exponent Overflow	-	If exponent is > +127, then ON
Exponent Underflow		If exponent of floating point result < - 128, then ON; otherwise, unchanged

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification is used.
2. All data is handled as 0 when DL modification is specified in the NS mode.

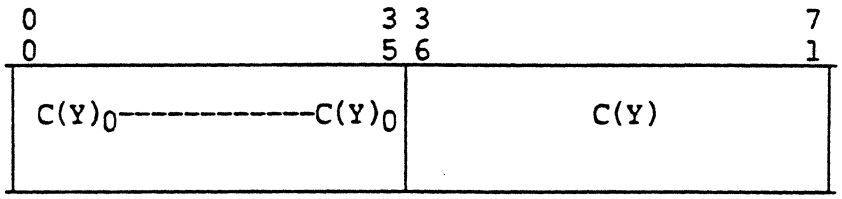
ADL	Add Low to AQ-Register	033 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(AQ) + C(Y, right-adjusted) --> C(AQ)

This instruction forms the following 72-bit number:



The lower half (bits 36 through 71) is C(Y). The bits in the upper half (bits 0 through 35) are equal to the C(Y) sign bit ($C(Y)_0$). This value is added to the AQ. If a carry is generated from Q as a result of this addition, it is passed on to A.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of AQ is exceeded, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

ADLA	Add Logical to A-Register	035 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(A) + C(Y) --> C(A); C(Y) unchanged

EXPLANATION: This instruction is identical to ADA with the exception that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(A) is generated, then ON; otherwise, OFF. When the carry indicator is ON, the range of A has been exceeded.

ADLAQ	Add Logical to AQ-Register	037 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(AQ) + C(Y-pair) --> C(AQ); C(Y-pair) unchanged

This instruction is identical to ADAQ except that the overflow indicator is not affected and an overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF. When the carry indicator is ON, the range of AQ has been exceeded.

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

ADLQ	Add Logical to Q-Register	036 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Q) + C(Y) --> C(Q); C(Y) unchanged

EXPLANATION: This instruction is identical to ADQ except that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

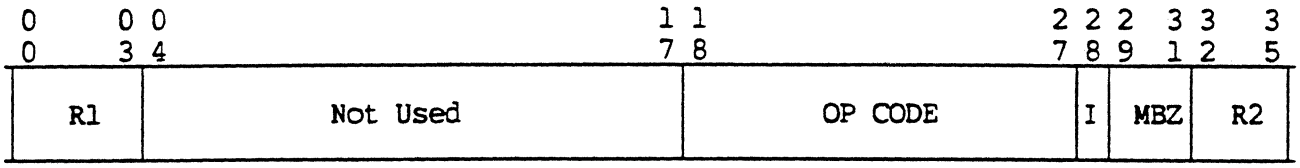
Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(Q) is generated, then ON; otherwise, OFF. When the carry indicator is ON, the range of Q has been exceeded.

ADLR	Add Logical Register to Register	435 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
        ADLR    R1,,R2
```

OPERATING MODES: Executes in ES mode only

SUMMARY: R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) + C(R2) --> C(R1); C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS:

Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(R1) is generated, then ON; otherwise, OFF.

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

ADLXn	Add Logical to Index Register n	02n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1..., 7 as determined by op code

$C(Xn) + C(Y)_{0-17}$ --> C(Xn); C(Y) unchanged

ES Mode

For n = 0,1..., 7 as determined by op code

C(GXn) + C(Y) --> C(GXn); C(Y) unchanged

EXPLANATION: This instruction is identical to ADXn with the exception that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of ADLX0

INDICATORS:

Zero - If C(Xn)/(GXn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)/(GXn)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(Xn)/(GXn) is generated, then ON; otherwise, OFF. When the carry indicator is ON, the range of Xn/GXn been exceeded

NOTES:

1. All data is handled as 0 when DL modification is specified for the NS mode.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ADQ	Add to Q-Register	076 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Q) + C(Y) --> C(Q); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(Q) = 0, then ON; otherwise, OFF

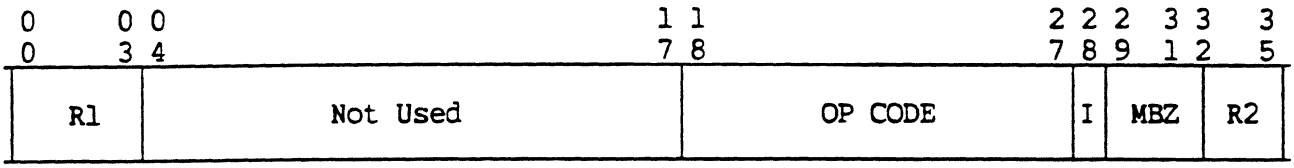
Negative - If $C(Q)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of Q is exceeded, then ON

Carry - If a carry out of bit 0 of C(Q) is generated, then ON; otherwise, OFF

ADRR	Add Register to Register	434 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
        ADRR    R1,,R2
```

OPERATING MODES: Executes in ES mode only

SUMMARY: R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) + C(R2) --> C(R1); C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS:

Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0 = 1$, then ON; otherwise, OFF

Overflow - If the range of R1 is exceeded, ON.

Carry - If a carry out of bit 0 of C(R1) is generated, then ON; otherwise, OFF.

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

ADXn	Add to Index Register n	06n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0, 1..., or 7 as determined by op code

$C(Xn) + C(Y)_{0-17} \rightarrow C(Xn)$; C(Y) unchanged

ES Mode

For n = 0, 1..., or 7 as determined by op code

$C(GXn) + C(Y) \rightarrow C(GXn)$; C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of ADX0

INDICATORS:

Zero - If $C(Xn)/(GXn) = 0$, then ON; otherwise, OFF

Negative - If $C(Xn)/(GXn)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of Xn/GXn is exceeded, then ON

Carry - If a carry out of bit 0 of C(Xn/GXn) is generated, then ON; otherwise, OFF

NOTES:

1. All data is handled as 0 when DL modification is specified in the NS mode.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ALR	A-Register Left Rotate	775 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: NS Mode

Rotate C(A) left the number of positions indicated by bits 11-17 of Y (Y modulo 128); enter each bit leaving bit position 0 in bit position 35.

ES Mode

Rotate C(A) left the number of positions indicated by bits 27-33 of Y (Y modulo 128); enter each bit leaving bit position zero in bit position 35.

The rotate count in the instruction must be a decimal number. To "right-rotate" n bits, use ALR 36-n.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ALS	A-Register Left Shift	735 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: NS Mode

Shift C(A) left the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with zeros.

ES Mode

Shift C(A) left the number of positions indicated by bits 27-33 of Y (Y modulo 128); fill vacated positions with zero.

The shift count in the instruction must be a decimal number.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

Carry - If $C(A)_0$ changes during the shift, then ON; otherwise, OFF. When the Carry indicator is ON, the algebraic range of A has been exceeded.

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ANA	AND to A-Register	375 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35, $C(A)_i$ AND $C(Y)_i$ --> $C(A)_i$;

C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0$ = 1, then ON; otherwise, OFF

ANAQ	AND to AQ-Register	377 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 71, $C(AQ)_i$ AND $C(Y\text{-pair})_i$ --> $C(AQ)_i$;

C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

ANQ	AND to Q-Register	376 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35, $C(Q)_i$ AND $C(Y)_i$ --> $C(Q)_i$;

ILLEGAL ADDRESS MODIFICATIONS: None

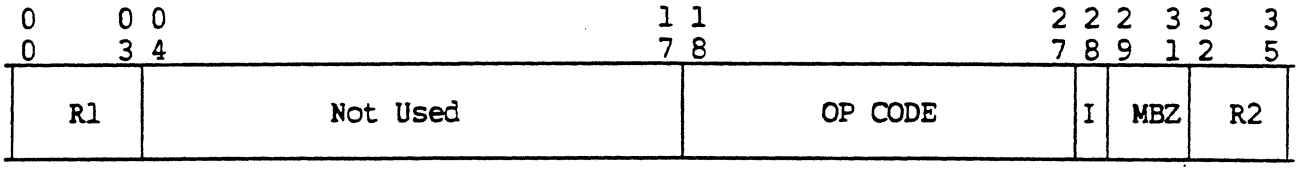
ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0$ = 1, then ON; otherwise, OFF

ANRR	AND Register to Register	535 (0)

FORMAT:



CODING FORMAT: 1 8 16

```
        ANRR    R1,,R2
```

OPERATING MODES: Executes in ES mode only

SUMMARY: R1, R2, = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

$C(R1)_i$ AND $C(R2)_i$ --> $C(R1)_I$ i = 0, 1, 2, ... , 35

C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

ANSA	AND to Storage from A-Register	355 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35, $C(A)_i$ AND $C(Y)_i$ --> $C(Y)_i$;

C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification or if an illegal repeat is used.

ANSQ	AND to Storage from Q-Register	356 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY For i = 0 to 35, $C(Q)_i$ AND $C(Y)_i$ --> $C(Y)_i$;

C(Q) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ANSX$\underline{n}$	AND to Storage from Index Register $\underline{n}$	34$\underline{n}$ (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0, 1,..., 7 as determined by op code

For i = 0 to 17, $C(Xn)_i$ AND $C(Y)_i$ --> $C(Y)_i$;

C(Xn) and $C(Y)_{18-35}$ unchanged

ES Mode

For n = 0, 1,..., 7 as determined by op code

For i = 0 to 35, $C(GXn)_i$ AND $C(Y)_i$ --> $C(Y)_i$;

C(GXn) is unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL, RPT, or RPD of ANSX0

INDICATORS: NS Mode

Zero - If bits $C(Y)_{0-17} = 0$, then ON; otherwise, OFF

Negative - If $C(Y)_0 = 1$, then ON; otherwise, OFF

ES Mode

Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0 = 1$, then ON; otherwise, OFF.

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ANXn	AND to Index Register n	36n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0, 1,..., or 7 as determined by op code

For i = 0 to 17, $C(Xn)_i$ AND $C(Y)_i$ --> $C(Xn)_i$

ES Mode

For n = 0, 1,..., or 7 as determined by op code

For i = 0 to 35, $C(GXn)_i$ AND $C(Y)_i$ --> $C(GX)_i$

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of ANX0

INDICATORS: NS Mode

Zero - If C(Xn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)_0$ = 1, then ON; otherwise, OFF

ES Mode

Zero - If C(GXn) = 0, then ON; otherwise, OFF

Negative - If $C(GXn)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. All data is handled as 0 when DL modification is specified in the NS mode.

AOS	Add One to Storage	054 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Y) + 0...01 --> C(Y)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of Y is exceeded, then ON

Carry - If a carry out of bit 0 of C(Y) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ARAn	Address Register n to Alphanumeric Descriptor	54n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        ARAn    LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY: For n = 0, 1,..., or 7 as determined by op code

$C(ARn)_{0-17}$ --> $C(Y)_{0-17}$

$C(ARn)_{18-23}$ --/translated\--> $C(Y)_{18-20}$

$C(Y)_{21-35}$ unchanged

EXPLANATION: This instruction is the converse of AARn. The alphanumeric descriptor is fetched from the computed effective address Y. The TA field code is examined to determine the type of data. Bits 18-23 of ARn are appropriately translated and replace bits 18-20 of the descriptor, and the word address (0-17) of ARn replaces bits 0-17. The updated descriptor is then stored back into location Y.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPD, RPT, RPL

ILLEGAL EXECUTES: Execution in ES mode

INDICATORS: None

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used, or if the descriptor TA field contains code 11.

2. AN IPR fault occurs if an attempt is made to execute this instruction in the ES mode.

EXAMPLE:

```
1       8       16                  32
        ARA6    DESCR               AR6 octal contents - 5 0 1 0 2 4 0 7
          .       .
          .       .
          .       .                 memory contents in octal
DESCR   ADSC9   ,,4                 5 0 1 0 2 4 0 0 0 0 0 4  - DESCR after
```

ARL	A-Register Right Logical Shift	771 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

Shift C(A) right the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with zeros

ES Mode

Shift C(A) right the number of positions indicated by bits 27-33 of Y (Y modulo 128); fill vacated positions with zeros.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ARNn	Address Register n to Numeric Descriptor	64n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        ARNn    LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY: For n = 0,1,...,7 as determined by op code

$C(ARn)_{0-17}$ --> $C(Y)_{0-17}$

$C(ARn)_{18-23}$ /translated\ $C(Y)_{18-20}$

Bits 21-35 of C(Y) unchanged

EXPLANATION: This instruction is the converse of NARn. The numeric descriptor is fetched from the computed effective address Y and the TN field bit is examined. Bits 0-17 of ARn replace the descriptor bits 0-17. Bits 18-23 of ARn are appropriately translated and replace bits 18-20 of the descriptor. The updated descriptor is then stored back in location Y.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. An IPR fault occurs if an attempt is made to execute this instruction in ES mode.

ARS	A-Register Right Shift	731 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

Shift C(A) right the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with bit 0 of C(A).

ES Mode

Shift C(A) right the number of positions indicated by bits 27-33 of Y (Y modulo 128); fill vacated positions with bit 0 of C(A).

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ASA	Add To Storage From A-Register	055 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(A) + C(Y) --> C(Y); C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of Y is exceeded, then ON

Carry - If a carry out of bit 0 of C(Y) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ASQ	Add To Storage From Q-Register	056 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Q) + C(Y) --> C(Y); C(Q) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of Y is exceeded, then ON

Carry - If a carry out of bit 0 of C(Y) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ASXn	Add To Storage From Index Register n	04n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,...,7 as determined by op code

$C(Xn) + C(Y)_{0-17} \rightarrow C(Y)_{0-17}$; C(Xn) and $C(Y)_{18-35}$ unchanged

ES Mode

For n = 0,1,...,7 as determined by op code

C(GXn) + C(Y) --> C(Y); C(GXn) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of ASX0

INDICATORS: NS Mode

Zero - If $C(Y)_{0-17} = 0$, then ON; otherwise, OFF

Negative - If $C(Y)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of Y_{0-17} is exceeded, then ON

Carry - If a carry out of bit 0 of C(Y) is generated, then ON; otherwise, OFF

INDICATORS: ES Mode

Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of Y is exceeded, then ON

Carry - If a carry out of bit 0 of C(Y) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

AWCA	Add with Carry to A-Register	071 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: If carry indicator is OFF, then C(A) + C(Y) --> C(A); C(Y) unchanged

If carry indicator is ON, then C(A) + C(Y) + 00...01 --> C(A); C(Y) unchanged

EXPLANATION: This instruction operates similarly to the ADA instruction except that if the carry indicator is ON prior to the execution of the instruction, a 1 is added to the least significant position of the A-register.

This instruction is intended for use with multiword precision binary arithmetic and for calculating checksums. The positive 1 added when the carry indicator is ON represents the carry from the next less significant word of the multiword addition.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of A is exceeded, then ON

Carry - If a carry out of bit 0 of C(A) is generated, then ON; otherwise, OFF

EXAMPLE: (Checksum Calculation)

```
1       8       16                  32
        LDI     =11324,DL
        LDA     INCARD
        EAX2    INCARD+2
        EAX3    =0
        RPDA    22,1
        ADLA    0,2
        AWCA    0,3
        CMPA    INCARD+1
        TNZ     ERROR
        LDI     =0500000,DL
```

AWCQ	Add with Carry to Q-Register	072 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: If carry indicator is OFF, then C(Q) + C(Y) --> C(Q); C(Y) unchanged

If carry indicator is ON, then C(Q) + C(Y) + 00...01 --> C(Q); C(Y) unchanged

EXPLANATION: This instruction operates similarly to the ADQ instruction except that if the carry indicator is ON prior to the execution of the instruction, a 1 is added to the least significant position of the Q-register.

This instruction is intended for use with multiword precision binary arithmetic and for calculating checksums. The positive 1 added when the carry indicator is ON represents the carry from the next less significant word of the multiword addition.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of Q is exceeded, then ON

Carry - If a carry out of Q_0 is generated, then ON; otherwise, OFF

EXAMPLE: (Triple-precision Binary Fixed-point Addition)

```
1       8       16              32
        STI     C               save overflow and overflow mask
        LXL0    C
        ANX0    =0044000,DU
        STX0    REST
        LDA     =1B24,DL        set overflow mask ON
        ORSA    C
        LDI     C
        LDQ     A+2             add low-order bits
        ADLQ    B+2
        STQ     C+2
        LDQ     A+1             add intermediate bits
        AWCQ    B+1
        STQ     C+1
        STI     C               restore overflow and overflow mask
        LDA     =0733777,DL
        ANA     C
REST    ORA     **,DL
        STA     C
        LDI     C
        LDQ     A               add high-order bits
        AWCQ    B
        STQ     C
```

AWD
AWDX

AWD AWDX	Add Word Displacement to Address Register	507 (1)

FORMAT: Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT: 1 8 16

```
{AWD }
{AWDX}   word displacement,R,AR
```

When the mnemonic is coded with X (AWDX), bit 29 is forced to zero.

OPERATING MODES: Any

SUMMARY: NS Mode

If bit 29 = 0: $y + C(DR) \rightarrow AR\underline{n}_{0-17}$

If bit 29 = 1: $C(AR\underline{n})_{0-17} + y + C(DR) \rightarrow AR\underline{n}_{0-17}$

In either case, zeros $\rightarrow AR\underline{n}_{18-23}$

ES Mode

If bit 29 = 0: $[(se)y + C(DR)]_{6-35} \rightarrow C(AR)_{0-29}$

If bit 29 = 1: $[(se)C(ARn) + (se)y + C(DR)]_{6-35} \rightarrow C(AR)_{0-29}$

(se) indicates sign extension.

In either case, zeros $\rightarrow ARn_{30-35}$

EXPLANATION: NS Mode

The y field (with bit 3 extended) is added to the contents of the register specified by the code in the DR field. Then, if bit 29 = 0, this value replaces bits 0-17 of the AR specified by bits 0-2 of the y field. If bit 29 = 1, this value is added to bits 0-17 of the specified AR and the resulting sum is stored in bits 0-17 of the specified AR. In either case, bits 18-23 of the specified AR are zeroed.

AWD
AWDX

ES Mode

The y field (with bit 3 extended) is added to the contents of the register specified by the code in the DR field. Then, if bit 29 = 0, this value replaces bits 0-29 of the AR specified by bits 0-2 of the y field. If bit 29 = 1, this value is added to the sign extended value of the specified AR bits 0-29 and the sum loaded into the specified AR bits 0-29. In either case, bits 30-35 of the specified AR are zeroed.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, and Ic specified in DR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLES: (Example applies to NS mode only)

```
1       8       16                  32
-------------------------------------------------------------

FLD1    BOOL    20100
        EAX4    FLD1        X4 octal contents  - 0 2 0 1 0 0
        AWDX    0,4,7       AR7 octal contents - 0 2 0 1 0 0 0 0
        AWD     2,,7        AR7 octal contents - 0 2 0 1 0 2 0 0

FLD2    BOOL    10000
        EAX2    FLD2        X2 octal contents  - 0 1 0 0 0 0
        EAX3    512         X3 octal contents  - 0 0 1 0 0 0
        AWDX    0,2,4       AR4 octal contents - 0 1 0 0 0 0 0 0
        AWD     1,3,4       AR4 octal contents - 0 1 1 0 0 1 0 0
```

BCD	Binary-to-BCD Convert	505 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY:

Shift C(A) left 3 positions;
|C(A)| / C(Y) --> 4-bit quotient;
C(A) - (C(Y) * quotient) --> remainder

Shift C(Q) left 6 positions;
00 --> $C(Q)_{30-31}$
4-bit quotient --> $C(Q)_{32-35}$
remainder --> C(A)

EXPLANATION:

The BCD instruction carries out one step of an algorithm for the conversion of a binary number to the equivalent binary-coded decimal, which requires the repeated short division of the binary number or last remainder by a 36-bit constant from memory.

$$c_i = 8^i * 10^{n-i} \text{ (for } i = 1, 2, \ldots),$$

with n being defined by $10^{n-1} \leq | \text{ number } | \leq 10^{n-1}$

For base K other than 10:

$$c_i = 8^i * K^{n-1}, \text{ where } K^{n-1} \leq | \text{ number } | \leq K^{n-1}.$$

One 6-bit character is produced each time the BCD instruction is executed. The character produced represents a decimal digit from 0 to 9.

The BCD instruction converts the magnitude of the contents of the accumulator to the binary-coded decimal equivalent. The method employed is to effectively divide a number by a constant, place the result in bits 30-35 of the quotient register, and leave the remainder in the accumulator. The execution of the BCD instruction allows the user to convert a binary number to BCD, one digit at a time, with each digit coming from the high-order part of the number. The address of the BCD instruction refers to a constant to be used in the division; a different constant is needed for each digit. In the process of the conversion, the number in the accumulator is shifted left three positions. The quotient register is shifted left six positions before the new digit is stored.

BCD

The values in Table 8-1 are the conversion constants to be used with the binary-to-BCD instruction. Each vertical column represents the set of constants to be used depending on the initial value of the binary number to be converted to its decimal equivalent. The instruction is executed once per digit, using the constant appropriate to the conversion step with each execution.

An alternate use of the table for conversion involves the use of the constants in the row corresponding to conversion step 1. If, after each conversion, the contents of the accumulator are shifted right three positions, the constants in the conversion step 1 row may be used one at a time in order of decreasing value until the conversion is complete.

Table 8-1. Binary-To-BCD Conversion Constants

Starting Range of C(AR)		$-10^{10}+1 \rightarrow 10^{10}-1$	$-10^{9}+1 \rightarrow 10^{9}-1$	$-10^{8}+1 \rightarrow 10^{8}-1$	$-10^{7}+1 \rightarrow 10^{7}-1$
	1	$8^1 \times 10^9$	8×10^8	8×10^7	8×10^6
	2	$8^2 \times 10^8$	$8^2 \times 10^7$	$8^2 \times 10^6$	$8^2 \times 10^5$
Conversion	3	$8^3 \times 10^7$	$8^3 \times 10^6$	$8^3 \times 10^5$	$8^3 \times 10^4$
Step	4	$8^4 \times 10^6$	$8^4 \times 10^5$	$8^4 \times 10^4$	$8^4 \times 10^3$
	5	$8^5 \times 10^5$	$8^5 \times 10^4$	$8^5 \times 10^3$	$8^5 \times 10^2$
	6	$8^6 \times 10^4$	$8^6 \times 10^3$	$8^6 \times 10^2$	$8^6 \times 10^1$
	7	$8^7 \times 10^3$	$8^7 \times 10^2$	$8^7 \times 10^1$	8^7
	8	$8^8 \times 10^2$	$8^8 \times 10^1$	8^8	
	9	$8^9 \times 10^1$	8^9		
	10	8^{10}			

Table 8-1 (cont). Binary-To-BCD Conversion Constants

$-10^6+1 \rightarrow 10^6-1$	$-10^5+1 \rightarrow 10^5-1$	$-10^4+1 \rightarrow 10^4-1$	$-10^3+1 \rightarrow 10^3-1$	$-10^1+1 \rightarrow 10^2-1$	$-10^1+1 \rightarrow 10^1-1$
$8^1 \times 10^5$	8×10^4	8×10^3	8×10^2	8×10^1	8
$8^2 \times 10^4$	$8^2 \times 10^3$	$8^2 \times 10^2$	$8^2 \times 10^1$	8^2	
$8^3 \times 10^3$	$8^3 \times 10^2$	$8^3 \times 10^1$	8^3		
$8^4 \times 10^2$	$8^4 \times 10^1$	8^4			
$8^5 \times 10^1$	8^5				
8^6					

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If prior to execution bit 0 of C(A) = 1, then ON; otherwise, OFF

NOTES:

1. The largest number that can be converted with the BCD instruction is that represented by 33 bits.
2. A 6-bit character is generated in the Q-register each time this instruction is executed.
3. The generated character represents one digit of the values 0-9.
4. One full 36-bit word cannot be directly converted by the BCD instruction.
5. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

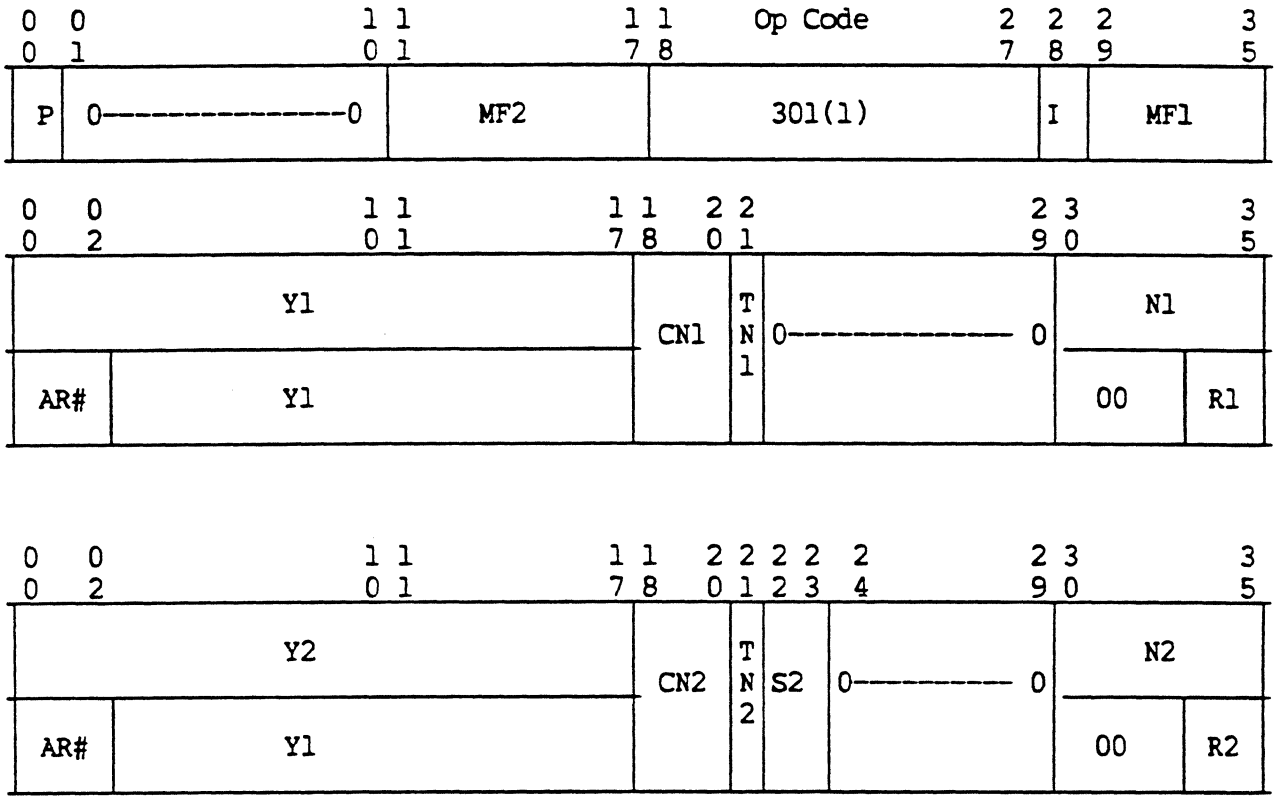
BCD

EXAMPLE:

```
1       8       16
        LDA     =15,DL
        LDQ     0,DL
        BCD     =80,DL
        BCD     =64,DL
```

BTD	Binary-to-Decimal Convert	301 (1)

FORMAT:



CODING FORMAT:

```
1       8       16
BTD     (MF1),(MF2),P
NDSC9   LOCSYM,CN,N,,,AM
NDSCn   LOCSYM,CN,N,S,,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY:

```
                converted
C(string 1) ---------> C(string 2)
```

EXPLANATION: The two's complement binary integer starting at location YC1 is converted into a signed string of decimal characters of data type TN2, sign and decimal type S2 (S2 = 00 is illegal), and scale factor 0; and is stored, right-justified, as a string of length L2 starting at location YC2. If the string generated is longer than L2, the high-order excess is truncated and the overflow indicator is set. If strings 1 and 2 are not overlapped, the contents of string 1 remain unchanged. The length of string 1 (L1) is given as the number of 9-bit segments that make up the string. L1 is equal to or is less than 8. Thus, the binary string to be converted can be 9, 18, 27, 36, 45, 54, 63, or 72 bits long. CN1 designates a 9-bit character boundary. If P=1, positive signed 4-bit results are stored using octal 13 as the plus sign. If P=0, positive signed 4-bit results are stored with octal 14 as the plus sign.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero - If the result is zero, then ON; otherwise, OFF

Negative - If the resultant sign is negative, then ON; otherwise, OFF

Overflow - If L2 is less than the length of the string generated, then ON; otherwise, unchanged

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2 or if an illegal repeat is used.

2. An IPR fault occurs if L1 is less than 1 or greater than 8, if CN1 does not contain a legal code, if S2 = 00, or if N2 is not large enough to specify at least one digit excluding sign.

EXAMPLES:

```
1       8       16                  32
        BTD
        NDSC9   FLD1,2,2            binary operand descriptor
        NDSC9   FLD2,0,4,1          decimal operand descriptor
        USE     CONST.              memory contents in octal
FLD1    DEC     -512                7 7 7 7 7 7 7 7 7 0 0 0
FLD2    BSS     1                   0 5 5 0 6 5 0 6 1 0 6 2
        USE                         any indicators set?     negative

        BTD
        NDSC9   FLD1,3,1            binary operand descriptor
        NDSC9   FLD2,1,3,2          decimal operand descriptor
        USE     CONST.              memory contents in octal
FLD1    DEC     255                 0 0 0 0 0 0 0 0 0 3 7 7
FLD2    BSS     1                   0 0 0 0 6 5 0 6 5 0 5 3
        USE                         any indicators set?     overflow
```

CAMP	Clear Associative Memory Pages	532 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode.

EXPLANATION: This instruction provides the capability to set the PTWAM ON or OFF, to clear the entire PTWAM, and to selectively clear the PTWAM. The instructions options are based on the instruction word tag, the effective address bits 16, and 17, and the ON/OFF state of the PTWAM.

When the instruction tag = 00 the following is executed within the CPU that is executing the instruction.

- When PTWAM is ON

  If $EA_{16,17}$ = 00 or 10, the PTWAM is cleared.

  If $EA_{16,17}$ = 01, the PTWAM is set OFF; the PTWAM is not cleared.

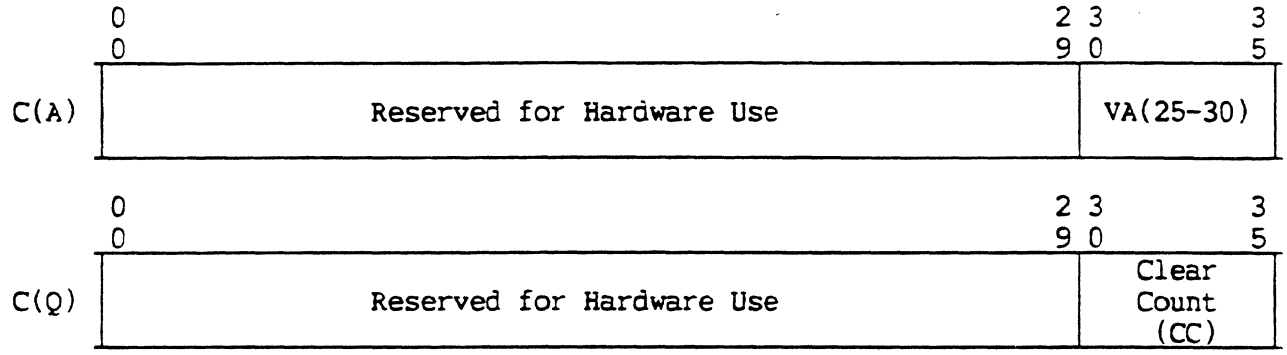
  If $EA_{16,17}$ = 11, the PTWAM is not affected.

- When PTWAM is OFF

  If $EA_{16,17}$ = 10, the PTWAM is cleared; the PTWAM is set ON.

  If $EA_{16,17}$ = 00, 01, or 11, the PTWAM is not affected.

When the instruction word tag = 01 a selective clear is done within the processor that executes this instruction according to the contents of the A and Q registers.



The VA corresponds to the lower six bits of the page number.

PTWAM entries having the lower 6 bits of the page number beginning at C(A) through C(A) + the CC in C(Q) are cleared.

When the instruction word tag = 2

A selective clearing of PTWAM is done in all processors depending upon the contents of the A and Q registers as shown above.

If clearing of all processors does not occur within 16ms, bit 0 of the A register is set to 1 in the processor that executes this instruction; otherwise this bit is unchanged.

The CAMP instruction is transmitted to the other procesors through the control SCU. The SCU selected is the control SCU.

When the instruction word tag = 3

The entire contents of PTWAM in all processors are cleared.

If clearing of all processors does not occur within 16 ms, bit 0 of the A register is set to 1 in the processor that executes this instruction; otherwise, this bit is unchanged.

The CAMP instruction is transmitted to the other processors through the control SCU.

ILLEGAL ADDRESS MODIFICATION: Only 00, 01, 02, or 03 allowed

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: None

NOTES:

1. The issuing CPU firmware builds an address that is transmitted to the SCU. This address is developed from the contents of the A and Q registers and the CAMP instruction type.

2. The issuing CPU also stores data, based on the contents of the A and Q registers and the CAMP instruction type, in reserved memory location 13x and resets reserved memory location 12x (where x is the processor number).

   13x contains data which is read by the receiving CPU, defining the clear operation. Each receiving CPU executes the CAMP instruction when the next interruptible point in its instruction stream is reached. The interrupt inhibit bit of the instruction is disregarded in this determination. If a receiving CPU determines that the contents of 13x are null, no action is taken in the execution of the CAMP instruction and a return is made to the next instruction. 12x is set by the receiving processors when their clear is complete. The CPU that issued the CAMP monitors the contents of 12x.

3. The reserved memory locations are accessed in absolute address mode relative to the Reserve Memory Base Register (RMBR). The RMBR also defines which CPUs are currently active, and thus from which CPU numbers, responses are anticipated in memory location 12x. Initialization firmware loads the RMBR with zero. On release of a CPU, software should also set the RMBR in that CPU to zero. If the RMBR is zero, a CPU will not respond to a broadcast CAMP. Thus the released CPU is not called upon to execute CAMPs.

4. A Command fault results when Slave or Master mode is used for execution of this instruction.

5. An IPR fault results when illegal address modification or illegal repeats are executed.

CANA	Comparative AND with A-Register	315 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY For i = 0 to 35, $C(Z)_i = C(A)_i$ AND $C(Y)_i$

C(A) and C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

Negative - If $C(Z)_0 = 1$, then ON; otherwise, OFF

CANAQ	Comparative AND with AQ-Register	317 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 71, $C(Z)_i = C(AQ)_i$ AND $C(Y\text{-pair})_i$

C(AQ) and C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

Negative - If $C(Z)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

CANQ	Comparative AND with Q-Register	316 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35, $C(Z)_i = C(Q)_i$ AND $C(Y)_i$

C(Q) and C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

Negative - If $C(Z)_0 = 1$, then ON; otherwise, OFF

CANXn	Comparative AND with Index Register n	30n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,...,7 as determined by op code

For i = 0 to 17, $C(Z)_i = C(Xn)_i$ AND $C(Y)_i$
C(Xn) and C(Y) unchanged

ES Mode

For n = 0,1,...,7 as determined by op code

For i = 0 to 35, $C(Z)_i = C(GXn)_i$ AND $C(Y)_i$
C(GXn) and C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of CANX0

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

Negative - If $C(Z)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged illegal by the assembler but executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CCAC

CCAC	Clear Cache	011 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master Mode

SUMMARY: This instruction functions as NOP.

EXPLANATION: The presence of CCAC in the instruction repertoire is for compatability only. All instructions reference cache except for the load and clear and the store compare instructions. These always bypass the cache and do not cause a block load on a directory hit.

ILLEGAL ADDRESS MODIFICATIONS: None. Address modification is not executed.

ILLEGAL REPEATS: None

INDICATORS: None affected

NOTE: A Command fault occurs if the processor is not in the Privileged Master mode for the execution of this instruction.

CIOC	Connect Input/Output Channel	015 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Priviledged Master mode

SUMMARY: $C(A)_{0-8}//0_{9-10}//Log.Ch.No._{11-17}//Scratch\ Pad_{24-35}$ --> Connect Word 1

Abs. Addr. $Y_{0-27}//00_{28-35}$ ---> Connect Word 2

$C(A)_{0-8}$ = a control field
$C(A)_{9-17}$ = unused
$C(A)_{18-35}$ = a logical channel number and a table entry

When $C(A)_{18-35}$ = 0-7, the logical channel number field = 0.

EXPLANATION: A double-word write to the designated control SCU occurs. The SCU stores the double-word in the port connect queue and informs the receiving port. The double-word is formed from the contents of the CPU A register, an entry in the CPU scratch pad, and the developed absolute address. The scratch pad content known as the connect table, consists of twelve 12-bit entries. The connect table is created external to software at initialization time.

The connect table entries are selected based on the contents of A_{18-35} as follows.

C(A)18-35	Recv'g Unit	Log. Chan. No.	Table Entry Number
0-3	Unused	N/A	4-7
4	CPU-0	N/A	8
5	CPU-1	N/A	9
6	CPU-2	N/A	10
7	CPU-3	N/A	11
8-135	IMX-0	0-127	0
136-263	IMX-1	0-127	1
264-391	IMX-2	0-127	2
392-519	IMX-3	0-127	3

CIOC

The connect table entries are located in the PATROL half of scratch pad memory at locations 74-77. A secondary connect table is located at 0-3 and is used to support system component reconfiguration. These entries define the following:

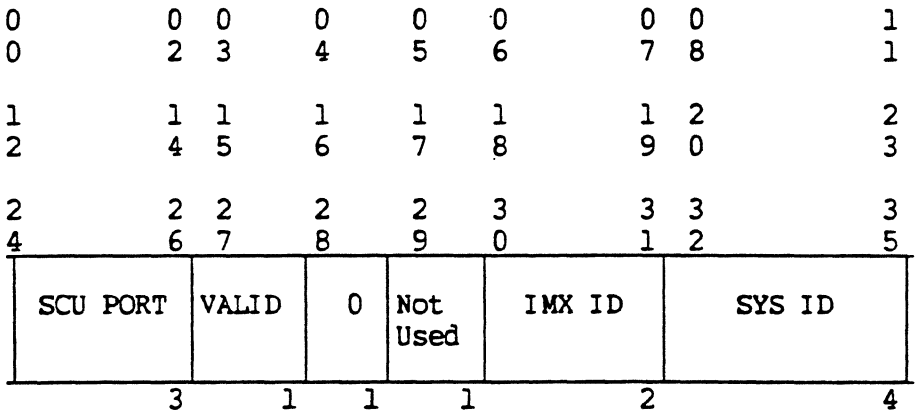
SCU PORT - Port and queue number of the unit that is to receive the connect

IMX ID - Used by the central systems software

SYS ID - Reserved for the central systems software

VALID - Valid connect word; 1 = valid.

The four primary entry words contain three 12-bit scratch entries in bits 0-11, 12-23, and 24-35. The format of the scratch pad data follows:



ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

CIOC

NOTES:

1. An IPR fault occurs if the use of this instruction is attempted by a processor in the Slave mode or Master mode.

2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

3. If the VALID bit in the connect table entry does not equal 1, a Command fault occurs.

4. The developed absolute address points to a 24-word mail box in main memory beginning at a 0 mod 8 address. The entire mailbox must reside within the same page. The first 8 words of this mailbox contain the basic information needed to execute the I/O, including a List Pointer Word (LPW) that points to the relative address of a Data Control Word (DCW) list. The DCW list is located in main memory. The mailbox also provides four different base addresses or Pointer Words (PTW) and related size/bounds information to be applied to the address fields of the various control words during address develoipment by the IMX. There may also be an optional "link word" to another mailbox.

   Upon termination of any I/O, the IMX stores the termination status word, DCW residue information, LPW residue, word counts, and extended status in the same 24-word mailbox. Table 8-6 illustrates the format of the standard mailbox for an indirect channel.

CIOC

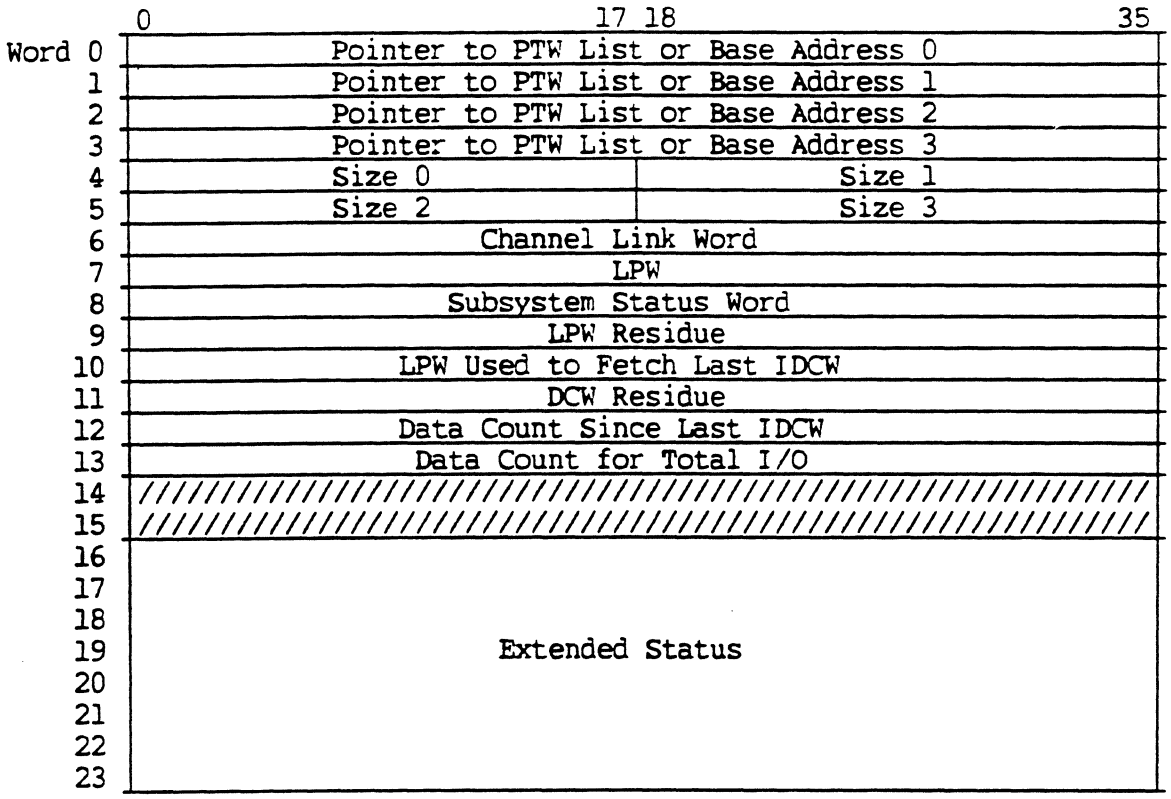
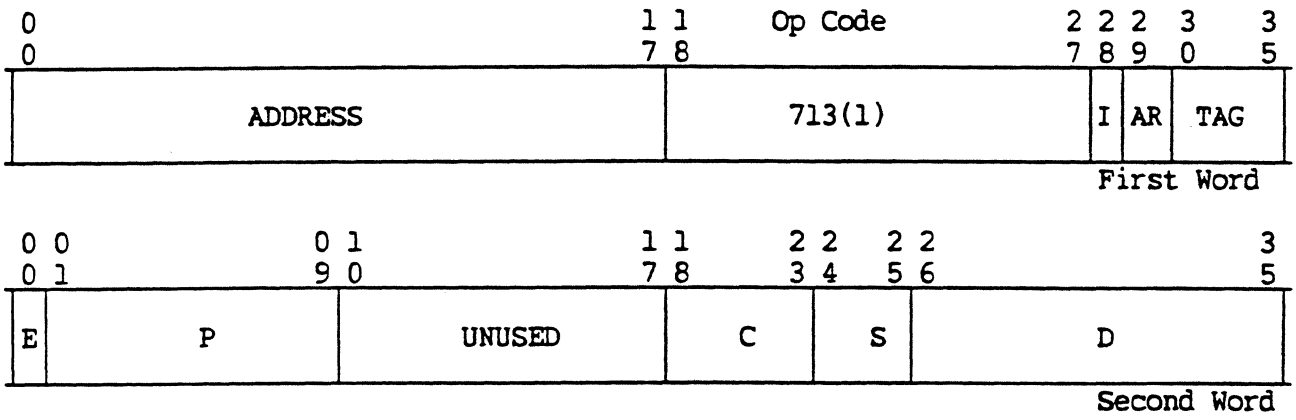


Figure 8-6. Standard I/O Mailbox

CLIMB	Domain Transfer	713 (1)

FORMAT:



The first word has the standard single-word instruction format (see Figure 8-1). The second word of the CLIMB instruction contains four control fields: C (actually made up of two fields, C_{22-23}, and C_{18-19}), E and P, and S and D. Bits 10-17 and 20-21 are not interpreted.

OPERATING MODES: Any

EXPLANATION: This instruction has four variations and performs functions of call, return, and common routine calls both within the same instruction segment and to a different instruction segment and also within the same domain and to a different domain reference.

The instruction word bit 28 (interrupt inhibit bit) does not accept interrupt for three of the four functions whether it is set to zero or to one. Bit 28 determines acceptance of interrupt for the other function.

The AR bit (bit 29) specifies whether or not the address register is to be used for generation of effective addresses. The tag field is also for address generation.

CLIMB

Versions of the CLIMB instruction include:

Mnemonic	Meaning
ICLIMB (Inward CLIMB - CALL)	Call another procedure which may reside in another domain
OCLIMB (Outward CLIMB - RET)	Return to calling domain
GCLIMB (Lateral Transfer - LTRAS)	Transfer to another procedure with passed arguments and parameters which may reside in another domain
PCLIMB (Lateral Transfer - LTRAD)	Transfer to another procedure which may be in another domain
PMME (System Entry CLIMB)	Privileged Master mode entry (This is a form of Inward CLIMB.)

The four control fields of the second word are defined as follows:

$C_{22,23}$ Instruction Version

This field determines one of the five (counting PMME) versions of the instruction to be executed:

00: Inward CLIMB (ICLIMB) Version - functions as a CALL, (i.e., a procedure invokes another procedure to accomplish a task and expects return of control from that other procedure.) Additional descriptors may be passed in a new parameter segment; an empty argument segment is created and placed in the argument stack. The processor state is saved (safe stored) if the SSF flag of the option register = 1. If S,D = 0,1760, this is the PMME version (System Entry). If S,D $\neq$ 0,1760, this is the ICLIMB version.

01: Outward CLIMB (OCLIMB) Version (RET) - functions as a return to the caller. The processor state is restored to the last safe store frame.

CLIMB

10: Lateral Transfer with same Parameter and Argument Segments (LTRAS). This version functions as an unconditional transfer, giving the callee the same visibility as the caller. The processor state is not saved. LTRAS is also called GCLIMB.

11: Lateral Transfer with new Parameter and Argument Segments (LTRAD). This version functions the same as the CALL version, except that the processor state is not saved. LTRAD is also called PCLIMB.

The terms inward, outward, and lateral refer to use of the stack segments. Inward means push the safe store frame on the safe store stack (saving the present processor state), frame a new parameter segment (PS), and open a new (empty) argument segment (AS). Outward means pop the safe store frame off the safe store stack (restoring the former processor state) and return PSR, ASR, LSR, ISR, IC, IR, SEGID(IS), DSAR, and, if specified, AR0-AR7, SEGID0-SEGID7, DR0-DR7, X0-X7, A, Q, E, and the Pointer/Length registers to their prior settings. Lateral means leave the safe store stack unchanged. The LTRAS version (10) keeps the PSR and ASR unchanged, while the LTRAD version (11) activates new PSR and ASR values in the same manner as an Inward CLIMB.

C_{18} X0/GX0 Control

For a CALL, LTRAS, or LTRAD, the C_{18} bit allows the caller to load the effective address of the CLIMB instruction into X0/GX0 if C_{18} = 1 and if an entry descriptor is referenced during execution of the CLIMB. For a RET, only the condition C_{18} = 1 is required to load X0/GX0 with the effective address of the CLIMB. If C_{18} = 0, X0/GX0 is not loaded, regardless of CLIMB version.

If the mode changes during a CLIMB (CALL, LTRAS, and LTRAD) the contents of X0 or GX0 are changed at the end of the CLIMB, to track each other. If bit 18 of the C field of the CLIMB instruction equals zero, or if the CLIMB was not an inter-domain transfer (an entry descriptor was not accessed) the register modifications are as follows:

Mode Change	Register Load
NS to ES	$0 \longrightarrow C(GX0)_{0-17}$
	$C(X0) \longrightarrow C(GX0)_{18-35}$
ES to NS	$C(GX0)_{18_{35}} \longrightarrow C(X0)$

If bit 18 of the C field equals 1 and the CLIMB is a domain transfer, the effective address specified by the CLIMB is loaded into X0 or GX0.

Mode	Register Load
NS to NS	EA_{0-17} ---> C(X0)
ES to NS	EA_{16-33} ---> C(X0)
NS to ES	0 ---> $C(GX00)_{0-17}$
	EA_{0-17} ---> $C(GX0)_{18-35}$
ES to ES	0 ---> $C(GX0)_{0-1}$
	EA_{0-33} ---> $C(GX)_{2-35}$

The X0 or GX0 loading is also done for a RETURN CLIMB.

In any CLIMB or RETURN CLIMB instruction in which the mode changes and the loading of Xn or GXn, (n = 1-7), is not specified, the contents of these registers are undefined.

o C19, Slave Mode

For a CALL, LTRAS, or LTRAD, the C_{19} bit allows Slave mode to be set. For an RET, C_{19} is ignored. If the CLIMB is the result of a fault interrupt, or invokes the System Entry (PMME), the C_{19} bit is overridden, and the Master Mode indicator is set.

Otherwise, for CALL, LTRAS, or LTRAD

if C_{19} = 0; 0 --->$C(IR)_{28}$
if C_{19} = 1; no change to $C(IR)_{28}$

If a CALL, LTRAS, or LTRAD attempts to transfer to a privileged segment (flag bit 26 = 1) and C_{19} = 0, an SCL1 or Security Fault, class 1 occurs.

CLIMB

- E and P Argument Passing

  The E and P fields are interpreted only for the ICLIMB (CALL) and PCLIMB (LTRAD) versions of the CLIMB instruction.

  If E = 1, P+1 descriptors are passed to the called routine. These descriptors are either prepared (shrunk and pushed onto the argument stack) by the instruction, or found in a descriptor segment, depending on the contents preset by the caller in DR0. When DR0 refers to an operand segment, a vector list is interpreted by the instruction to prepare descriptors; when DR0 refers to a descriptor segment, the descriptors are in the segment. In both cases, the PSR is loaded with a type 1 descriptor, framing the P+1 descriptors of parameters (or one parameter, if the P field is zero).

  If E = 0, no parameters are passed. The P field is ignored.

  In both cases, the ASR is updated in such a way that it locates the next available even-word location of the descriptor stack. The bound field is set to zero. The flag bit 27 is set to zero to indicate an empty segment. Details related to the PSR and the ASR are provided later in the CLIMB discussion.

  The E and P fields are not interpreted for the RET and LTRAS versions of the CLIMB instruction.

- S, D Field

  For CALL, LTRAS, or LTRAD, this field indicates the origin (SEGID) of the the descriptor that determines the destination of the CLIMB, or that the CLIMB is a System Entry (PMME).

  For the outward climb (RET), this field is ignored.

Instruction Variations

CLIMB variations determined by the settings in bits 22 and 23 of the C field are described below. When the CLIMB instruction is executed, a number of checks must be performed before the CPU state is altered.

Inward CLIMB (CALL/ICLIMB) C field bits 22 and 23 = 00

1. The S and D fields are interpreted in the same manner as the S and D fields of the vector in the LDDn instruction, except that, in this instance, the values S = 0 and D = 1760 (octal) define a PMME. If S = 0 and D = 1761 or 1763-1767 (octal), an IPR fault occurs.

   a. When S = 0, D = 1760_8, a special system entry is started at the same level as fault and interrupt. Hardware obtains the segment descriptor (this must be an Entry Descriptor) from a fixed memory location. The Master Mode indicator is always set to ON and the C field bit 19 is ignored. After the entry descriptor is obtained from the fixed memory location, execution of the CLIMB instruction is continued as when a normal entry descriptor is obtained. When there is no entry descriptor in the fixed memory location, an IPR fault occurs.

   b. If the CLIMB is a result of a fault or interrupt, this is an interdomain transfer, requiring an entry descriptor, which is obtained from locations in the operating system as follows:

      Interrupt: 30-31 octal
      Fault: 32-33 octal
      PMME: 34-35 octal

2. The CLIMB instruction S and D fields are used to access the specified segment descriptor segment or register and obtain the segment descriptor. The referenced descriptor must be one of the following types in order to continue execution of the CLIMB instruction:

   - Standard Descriptor (T = 0)
   - Descriptor Segment Descriptor (T = 1 or 3)
   - Entry Descriptor (T = 8, 9, or 11)

   If the CLIMB instruction is a result of an interrupt, the processor will attempt to obtain an entry descriptor from the operating system location 30-31 (octal).

CLIMB

If the CLIMB instruction has not yet been linked to one of the preceding descriptors, the obtained descriptor may be a dynamic linking descriptor (T = 5). In this case, the CLIMB instruction is terminated and a Dynamic Linking fault is generated. All other descriptor types (T = 2, 4, 6, 7, 10, or 12-15) terminate the CLIMB instruction and cause an IPR fault.

Given a descriptor segment descriptor, an entry descriptor, or a standard descriptor, the activity varies as follows:

a. Standard Descriptor (T=0)

When the descriptor referenced by the S and D fields is a standard descriptor, the CLIMB instruction is an intradomain transfer and the linkage segment register is not changed.

The obtained descriptor becomes the new instruction segment descriptor. Flag bits 25, 27, and 28 are checked and must be 1; otherwise, an appropriate fault occurs. The base and bound are checked for modulo 32 bytes; if the test fails, an IPR fault occurs.

b. Descriptor Segment Descriptor (T = 1 or 3)

When a type 1 or 3 descriptor is referenced by the S and D fields of the CLIMB instruction, the base of the type 1 or 3 descriptor is used as a pointer to an entry descriptor. Flag bits 20, 27, and 28 must be 1 and the bound field must be >= 7 bytes; otherwise, a Bound fault occurs. If the obtained descriptor is not an entry descriptor nor a dynamic linking descriptor, an IPR fault occurs.

If a dynamic linking descriptor is obtained, a Dynamic Linking fault occurs.

c. Entry Descriptor (T = 8, 9, or 11)

When an entry descriptor is referenced by the S and D fields of the CLIMB instruction (either directly or indirectly), the CLIMB instruction is an interdomain transfer. Entry descriptors may be of type T = 8, 9, or 11. The type of entry descriptor determines how much data (register contents) will be safe stored, and how the renewal of the pointer register will be processed.

CLIMB

Using the entry descriptor, the new instruction segment descriptor is obtained from the new linkage segment described by the entry descriptor. The new linkage segment is assumed to be present in real memory, because the entry descriptor does not have a flags field to indicate this, and the hardware attempts to obtain the new instruction segment descriptor.

The obtained instruction segment descriptor must be a standard descriptor with T = 0 and flag bits 25, 27, and 28 must be 1. If flag bit 25 is 0, a Security Fault, Class 2 occurs; if flag bit 28 = 0, a Missing Segment fault occurs; if flag bit 27 = 0, an STR fault occurs. The hardware also checks the base and bound of the new instruction segment descriptor for modulo 32 bytes; if the test fails, the instruction terminates in an IPR fault. If T is not 0, an IPR fault occurs.

3. A new parameter segment is prepared as described below.

The E bit of the second word of the CLIMB instruction is checked. If the E bit = 0, the segment descriptor is not passed (no parameter segment is prepared) and the operation proceeds to the safe store.

If the E bit = 1, the segment descriptor is passed. The operation that follows depends upon the type of the segment descriptor in DR0. An IPR fault occurs if the type for this segment is 3, 5, 7-11, 13 or 15.

a. Descriptor Type in DR0 = 1

If the descriptor type contained in DR0 is 1, the descriptors to be passed as parameters have already been prepared and are the last P+1 descriptors in this descriptor segment. Thus, the hardware does not prepare any descriptors but frames these last P+1 descriptors with the parameter segment register. In this case, hardware performs a bound check and if P + 111 > DR0, a bound fault occurs.

b. Descriptor Type in DR0 = 0, 2, 4, 6, 12, or 14

If the descriptor type contained in DR0 is 0, 2, 4, 6, 12, or 14, the hardware prepares descriptors. The vector list is located by pointer register zero (i.e., AR0 and DR0 combined). The descriptor identified by the S and D fields of each vector is obtained, prepared exactly as described in the definition of the LDDn instruction, and placed in the next available location in the argument

CLIMB

segment as described in the definition of the SDRn instruction. This procedure is continued until all P+1 descriptors have been prepared and placed in the argument segment. Various faults may occur during this operation as described in the definitions of the LDDn and SDRn instructions. Note that a vector with an S and D field of S = 0, D = 1760 or 1761 (octal) causes an IPR fault; S and D field values of S = 0, D = 1763 or 1764 (octal) require that the processor be in Privileged Master mode (as described in LDDn), which in this case refers to the processor mode at the beginning of the CLIMB instruction.

If a vector specifies that a data stack descriptor is to be formed and the associated bit in the option register specifies that the stack space is to be cleared, the CLIMB instruction performs the clear function.

If several data stack shrinks are specified, the second and subsequent data stack shrink operations are performed using the previously changed new value of the data stack address register (DSAR).

4. Safe Store Operation

The safe store operation differs depending upon the type of the segment descriptor referenced with the ICLIMB S and D fields. The size of the generated safe store frame and the stored data is determined by the referenced segment descriptor. The SSR base indicates the starting address of the last frame of the stored data prior to this CLIMB. The size of the last frame must therefore be added to the SSR base before the new frame is stored. In relation to the SSR, a 2-bit hardware control register, called the stack control register (SCR) is used. The SCR contains a code indicating the size of the last frame placed in the safe store stack.(The SCR, is initialized to 11_2 or 10_2 (binary) when the LDSS instruction is executed.) (Refer to details for the LDSS instruction.) The following displays the flow of safe store operation. When the safe store bypass flag (option register bit 19) is ON (zero), safe store is bypassed and processing proceeds to change the register contents as described under Loading the Registers.

a. The SSR base is increased, and the bound decreased, as follows based on the SCR content.

SCR	SSR Base	SSR Bound
00_2	+16 words	-16 words
01_2	+24 words	-24 words
10_2	+80 words	-80 words
11_2	+64 words	-64 words

The SSR base indicates the start of the newly generated safe store frame as a result of this operation.

NOTE: When hardware adds the SSR base, no check is performed to check for carry. Software must ensure that the base value initially loaded into the SSR is not at the end of the working space.

A safe store stack fault occurs in conjunction with a Inward (programmed) climb instruction, or in conjunction with the wired-in CLIMB instruction that results from a fault or interrupt. This fault indicates that the safe store stack has only one or two 64-word frames remaining.

After completing the safe store on a Inward CLIMB (SSR base and bound have been updated), if the SSR bound < 239 + 3 bytes, the hardware will not access the instruction pointed bo by the new ISR and IC, but will execute the Safe Store Stack fault. This causes another safe store stack frame to be stored. This frame will contain the "transferred to" domain registers from the inward CLIMB. Word 5, bit 10 (SSSF) will be set to 1 and the fault code in bits 12-16 of word 5 will be set to 00000.

When executing a fault or interrupt CLIMB, the hardware updates the SSR base and bound while generating the safe store frame, to determine whether a Safe Store Stack fault should be indicated in the safe store frame with the original fault or interrupt. If the SSR bound < 239 words + 3 bytes, the hardware will set word 5, bit 10 (SSSF) = 1 and leave the original fault code or interrupt cell number in word 5, bits 12-16. The Safestore Stack fault will not be executed; a separate safestore stack frame will not be stored.

NOTE: The operating system software must monitor word 5, bit 10 (SSSF) in each fault or interrupt frame in the safe store stack and to initiate appropriate action whenever this bit = 1.

b. The SCR content is saved in the new safe store frame.

c. The new SCR value is determined as follows, with the lower two bits of the type field (T) of the first word of the last segment descriptor referenced by the CLIMB instruction S and D fields, and the value of bit 24 of the ISR prior to the start of the CLIMB instruction.

T Field	ISR Bit 24	SCR
0 or 8	0 or 1	00_2
9	0 or 1	01_2
11	0	11_2
11	1	10_2

d. The amount of stored data (register content) is determined by the SCR value at this time (as described in item c above). The value of the SCR at this time is determined by the type of segment descriptor referenced by this CLIMB instruction and the ISR bit 24). As illustrated in Figures 8-7 and 8-8, 16, 24, 64, or 80 words are stored in accordance with the SCR content.

When the frame size is 64 or 80 words, the actual number of words stored may depend on the state of the indicator register bit 30 (multiword instruction interrupt or fault). The processor hardware sets IR bit 30 = 1 when a multiword EIS instruction is interrupted or faulted. IR bit 30 may also be set to 1 by the operating system software. The actual number of words stored when the frame size is 64 words is 48 words, unless IR bit 30 = 1, in which case 52 words are stored. When 52 words are stored, the pointer and length registers are included in the 64-word frame starting at word 48. Word 4, bit 30 is stored as a 1, and then IR bit 30 is set to 0, as though an SPL instruction had been executed.

e. Since the SCR is created by the hardware on the ICLIMB, the mode and SCR should be consistent on the RETURN CLIMB. If software modifies the SCR content of the safe store frame such that the mode, NS or ES, is inconsistent with the safe store frame size, an IPR fault will occur.

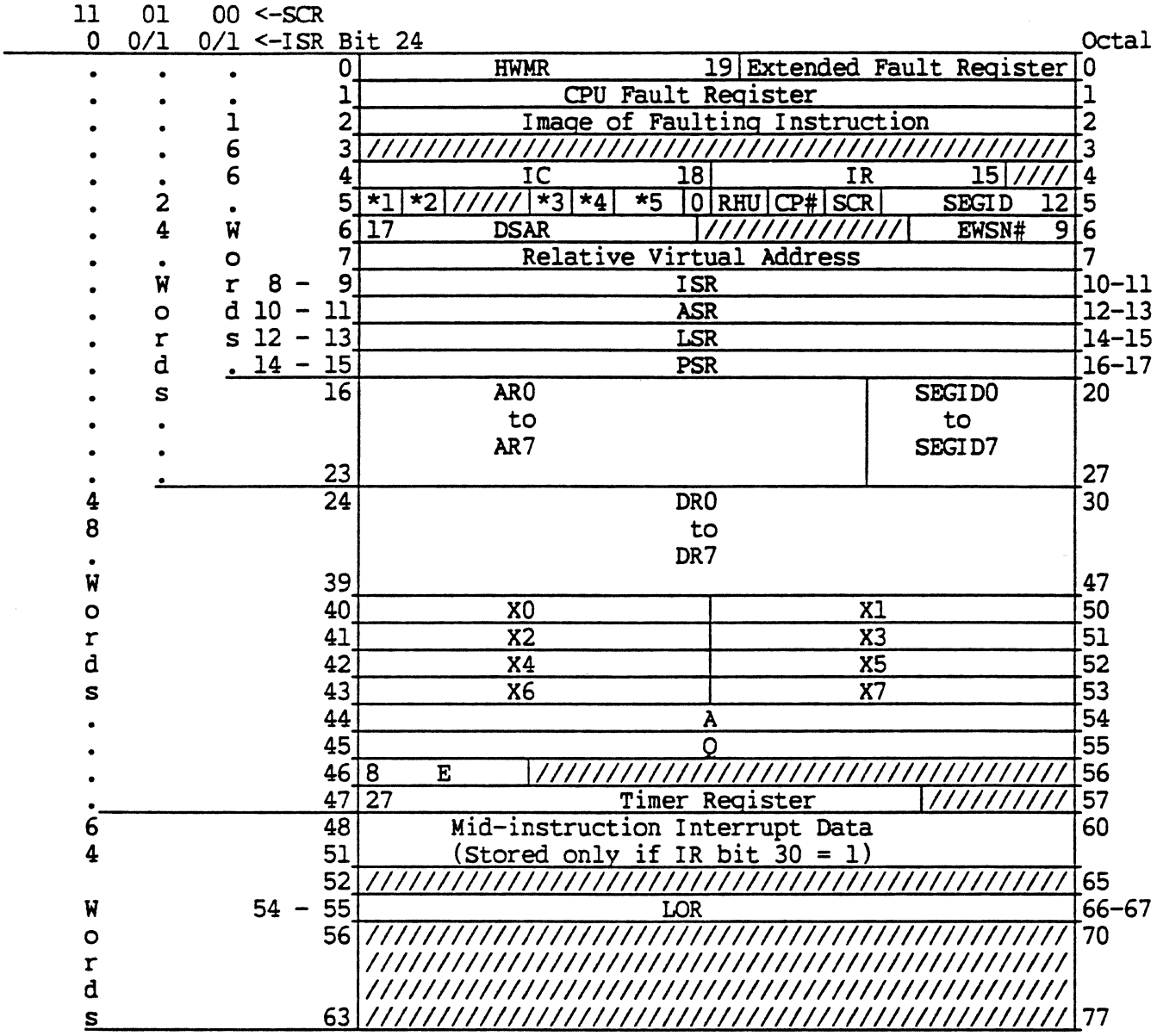


Figure 8-7. Safe Store Stack Format - NS Mode

*1: Fault Retry Flag - bit 1
*2: Fault Tally Flag - bit 2
*3: Safe Store Stack Fault Flag - bit 9
*4: Fault/Interrupt - bit 10
0 = fault
1 = interrupt
*5: Fault/Interrupt Code - bits 11-17

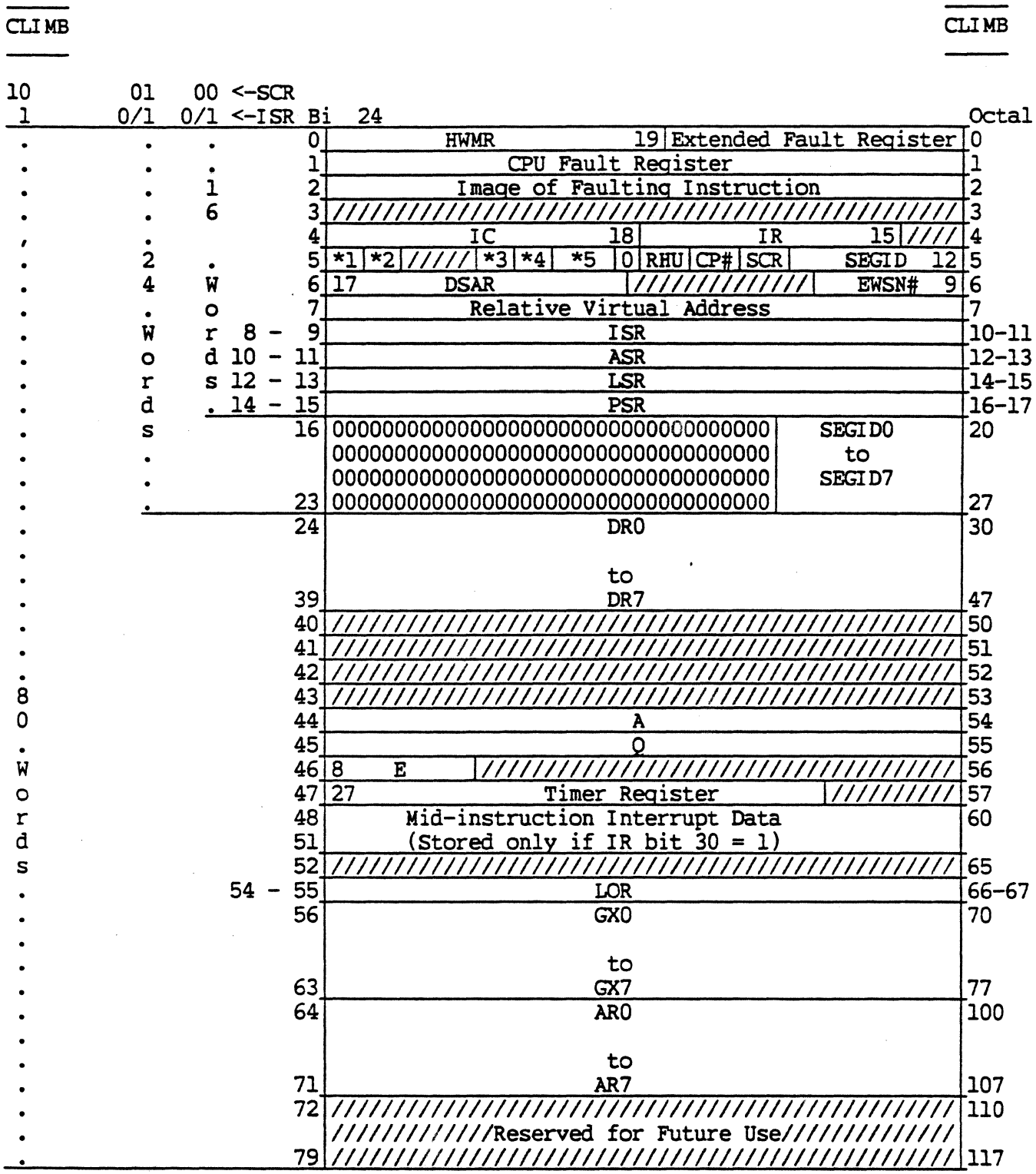


Figure 8-8. Safe Store Stack Format - ES Mode

*1: Fault Retry Flag - bit 1
*2: Fault Tally Flag - bit 2
*3: SSSF Flag - bit 9
*4: Fault/Interrupt - bit 10
0 = fault 1 = interrupt
*5: Fault/Interrupt Code - bits 11-17

CLIMB

Some of the fields shown in Figures 8-7 and 8-8 are stored only with a CLIMB instruction executed by hardware in response to faults or interrupts, and are meaningless when using the programmed CLIMB instruction.

The following discussion explains the contents of the safe store stack as illustrated in Figures 8-7 and 8-8.

Word 0 Bits 0-19

High Water Mark Register (HWMR) stored. The content of the HWMR is the value in the register when the CLIMB instruction started.

Bits 20 to 35 Extended Fault Register

Word 1:

Upon occurrence of a fault, the CPU fault register

Word 2:

Upon occurrence of a fault, the image of the faulting instruction

Word 3:

Reserved for hardware use

Word 4: Bits 0 to 17

The instruction counter (IC) value is stored as follows:

IC = IC + 2

Refer to Section 6 for the description of the value stored when a fault or interrupt occurs.

Bits 18 to 32

Indicator register (IR) contents.

Bits 33 to 35 Not used.

Word 5: Bits 0 to 9, 18, 19

CLIMB

Reserved for hardware use. When an interrupt or a Connect, Timer Runout, Shutdown, or Missing Page fault occurs, a 1 is stored in word 5, bit 9 to indicate the recoverable type. When other faults occur, a 0 is stored in word 5, bit 9.

Bit 10

SSF (Safe Store Stack fault flag). Refer to Section 6 for description of faults.

Bit 11 to 17

These bits are meaningful only when faults or interrupts occur. (Refer to Section 6, Faults and Interrupts for details.)

Bits 19 to 20

CPU number

Bits 22, 23

Stack Control Register (SCR)
Bits 24 to 35

SEGID(IS)

Word 6: Bits 0 to 16

The value stored here is the DSAR content when the CLIMB instruction started.

Bits 17 to 26

Reserved for hardware use.

Bits 27 to 35

When a Missing Page fault occurs, the hardware stores the effective working space number of the virtual address which caused the fault. It is not used in other cases.

Word 7:

When a missing page fault occurs, the hardware stores the relative virtual address which caused the fault. It is not used in other cases.

CLIMB

Words 8-47, 54-71:

As illustrated in Figure 8-7 for NS mode and Figure 8-8 for ES mode, the hardware stores register contents. These contents consist solely of values at the beginning of the CLIMB instruction. In particular, when a segment descriptor is pushed onto the argument stack during execution of the CLIMB instruction, the safe stored ASR bound value is that before the push operation.

When SCR = 10, bits 0 to 23 of the words 16 - 23, are all zero, and the values of words 40 to 43 are undefined.

When the ISR bit 24 immediately before the CLIMB instruction equals 1 with the 24-word stack, bits 0 to 23 of words 16 to 23 are all zero.

Words 48-53:

Hardware stores information for restart of instruction execution only in response to faults and interrupts.

The information stored in this area is normally the content of the pointers and lengths register when a fault or interrupt occurs during execution of an interruptible multiword instruction (when saved with the IR bit 30 set to ON). Even when the IR bit 30 is not set to ON, information is stored in this area, for example, for a Missing Page fault. The content of this area must not be changed by software.

When software does not specify type T = 11 as the entry descriptor for a fault or interrupt, the system cannot return correctly.

Words 72-79: Reserved for future used.

5. Loading the Registers

After the state is saved in the safe store stack, the registers are changed as described below.

CLIMB

a. Loading the Instruction Segment Register (ISR)

For an intradomain transfer, the standard descriptor referenced by the S and D fields of the instruction is placed in the ISR. If the S and D fields referenced a DRn (177n), then SEGIDn --> SEGID(IS); otherwise, S and D --> SEGID(IS).

For an interdomain transfer, the descriptor pointed to by the ISEGNO field of the entry descriptor is loaded into the ISR. SEGID(IS) is set to S = 3, D = ISEGNO.

b. Loading the Instruction Counter (IC)

For an intradomain transfer, an effective address is formed using the address field of the CLIMB instruction and applying the indicated AR and/or tag field modification. This 18-bit effective address is placed in the instruction counter.

c. Loading the Linkage Segment Register (LSR)

For an intradomain transfer, the linkage segment does not change.

For an interdomain transfer, a standard descriptor from the entry descriptor is placed in the LSR as follows:

- Base = Linkage base (LBASE) with zeros in the 10 most significant bit positions
- Size = Linkage bound (LBOUND) extended with three 1 bits on the right and with zeros in the 7 most significant bit positions
- WSR = WSR (working space register)
- T = 1
- Flags - Bits 20, 22, 23, 27, and 28 = 1
  Bits 21, 24, 25, and 26 = 0

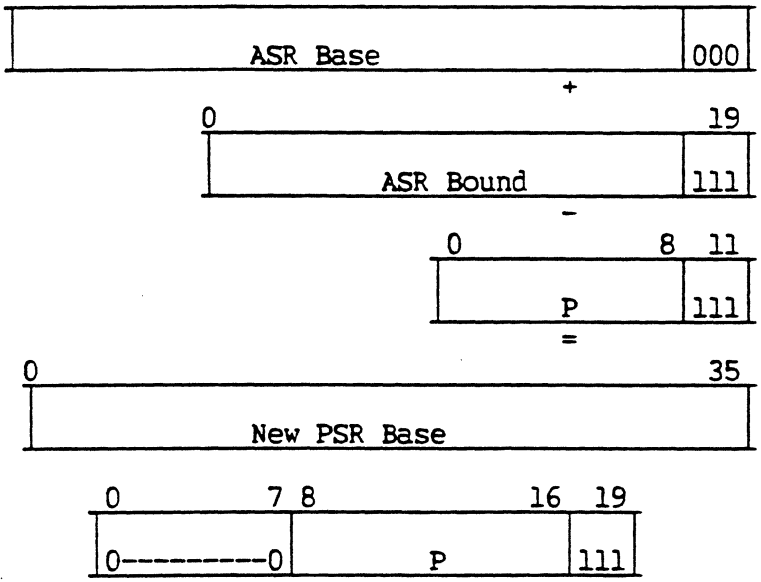
For an interdomain transfer, the 18-bit entry location contained in the entry descriptor is placed in the instruction counter.

d. Argument Stack Register (ASR) and Parameter Segment Register (PSR Generation

When E bit = 0 (pass no parameters) or when E bit = 1 (pass parameters) and DR0 type T = 1

CLIMB

- The new PSR is generated as follows.

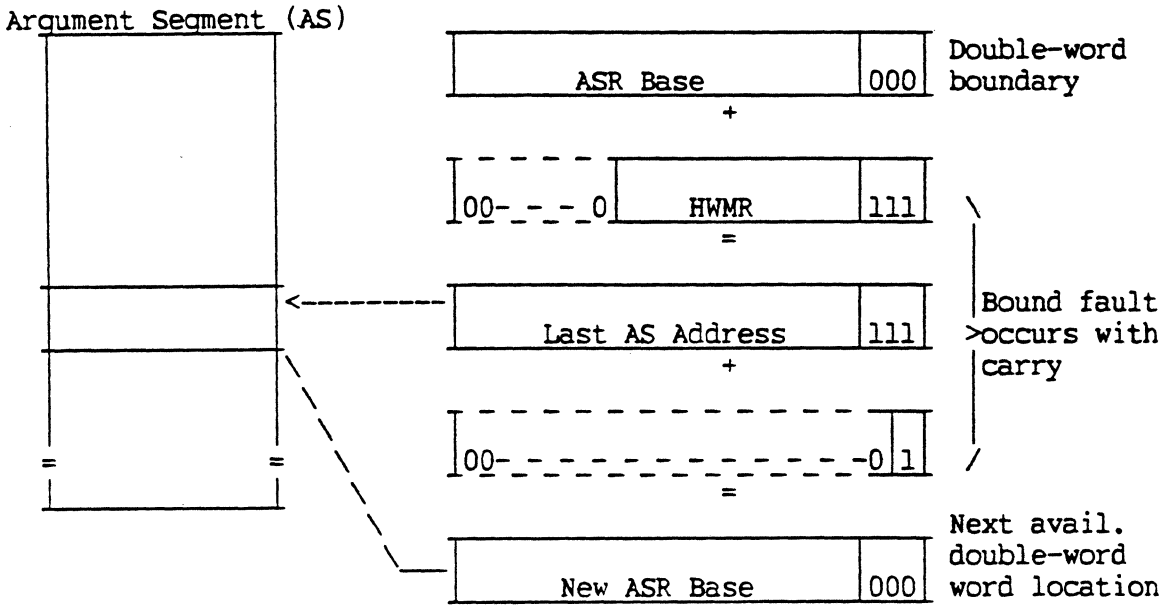


- The new ASR base is generated as follows. (The new ASR generation is independent of ASR flag bit 27.)

- When C(HWMR) = 0

  ASR base --> New ASR base

- When C(HWMR) ≠ 0

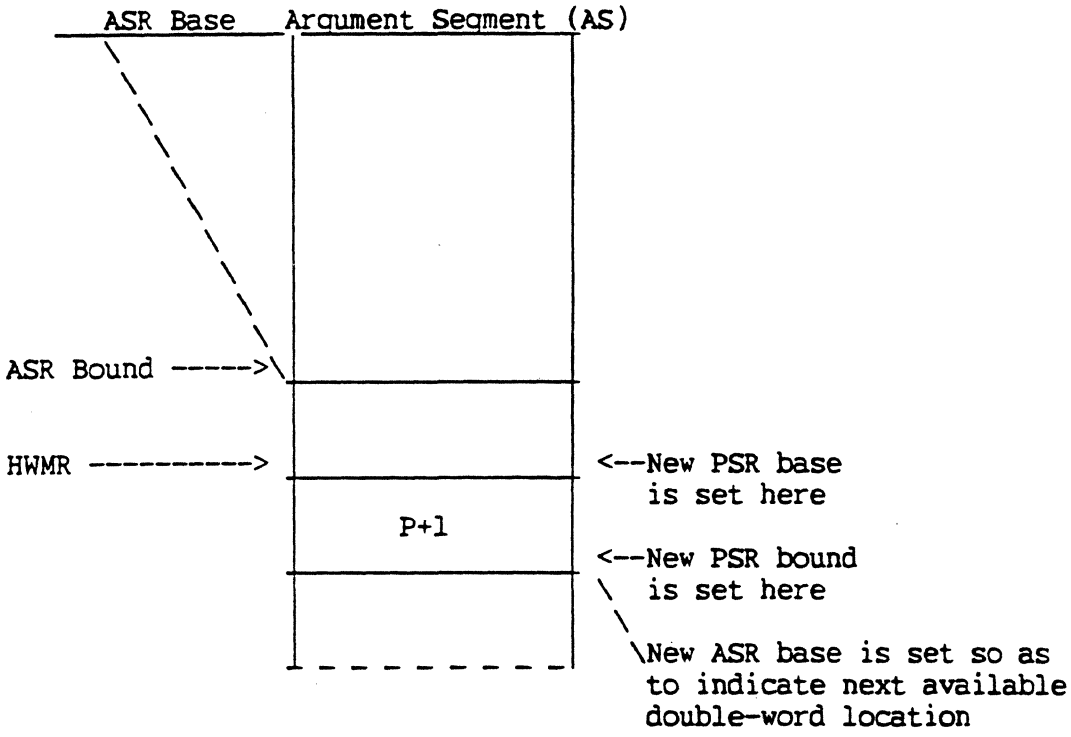


CLIMB

The new ASR bound and flat bit 27 are set to 0.

When E bit = 1 (pass parameters) and DR0 type = 0, 2, 4, 6, 12, or 14

- The segment descriptor described with the PSR is prepared by hardware as with the last P + 1 segment descriptors in the argument segment as follows.



- The descriptors to be framed by the PSR are the last P+1 descriptors in the descriptor segment pointed to by DR0.

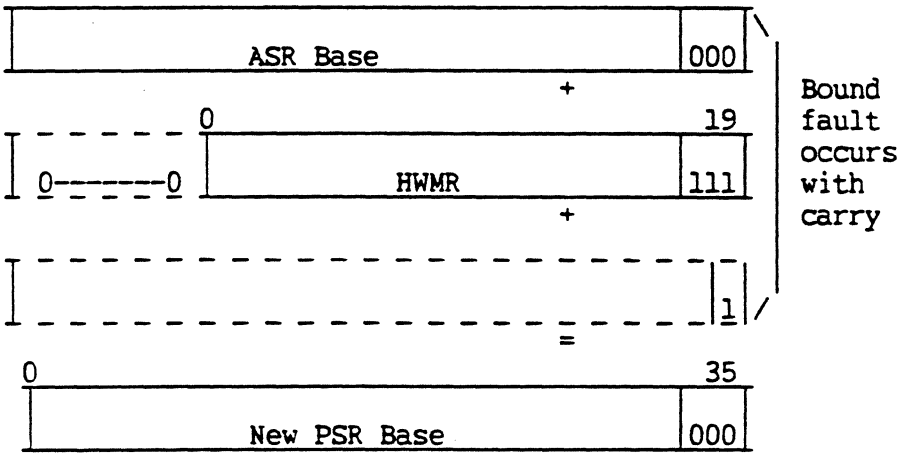
CLIMB

The new PSR base is generated as follows.

- When C(HWMR) = 0

  ASR base --> new PSR base

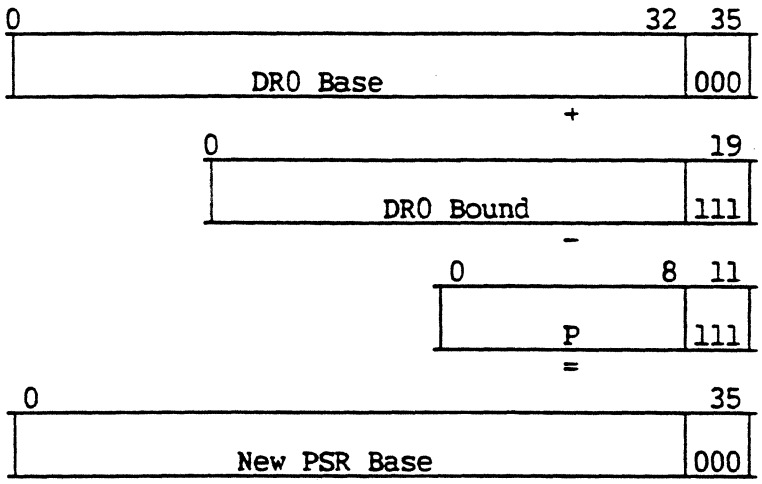
- When C(HWMR) ≠ 0



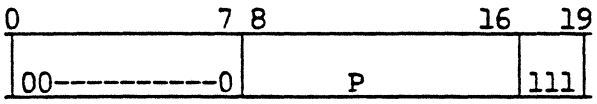
The new PSR base shown above works as the start address in the area where the segment descriptor is prepared as a parameter.

CLIMB

- The new PSR base is set to the value DR0 base + DR0 bound - P as shown below.



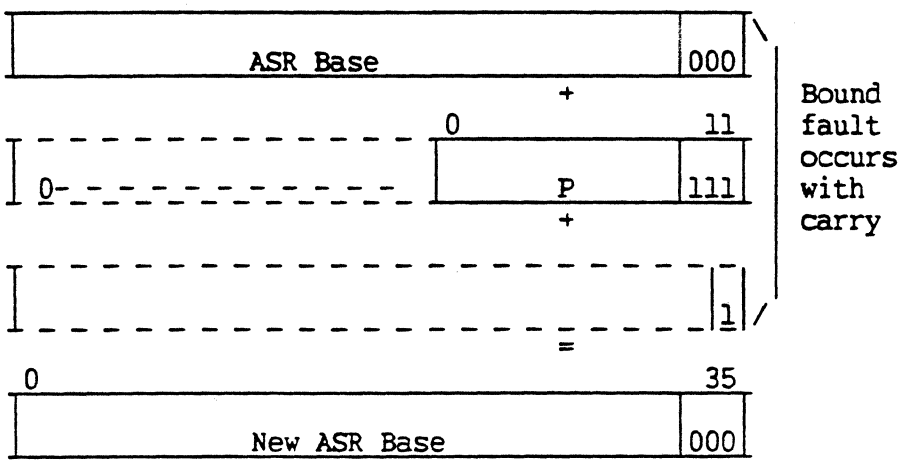
The new PSR bound is generated as follows.



- The new base and bound values formed are loaded into the PSR, framing the last P+1 descriptors of the segment. Bits 20-35 of the first word of DR0 (flags field, WSR or WSN field, and T field) are copied to the corresponding bit positions of the PSR.

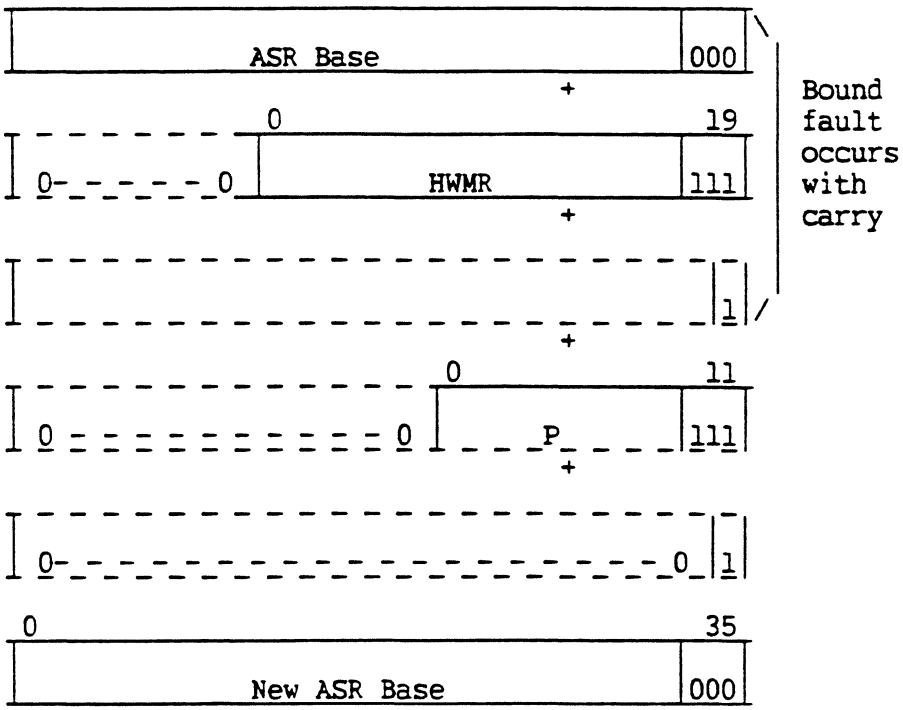
The new ASR base is generated as follows:

- When C(HWMR) = 0



CLIMB

- When C(HWMR) ≠ 0

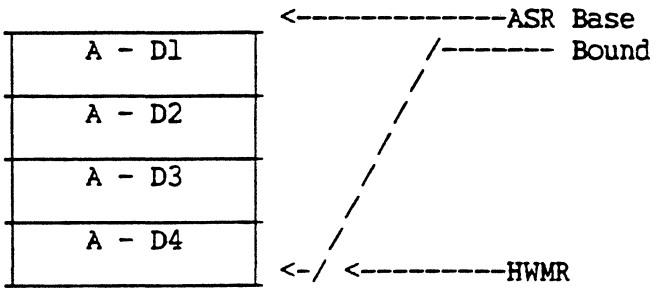


The new ASR bound and flag bit 27 are set to 0.

Independent of the E bit setting and DR0 type, the HWMR is set to 0.

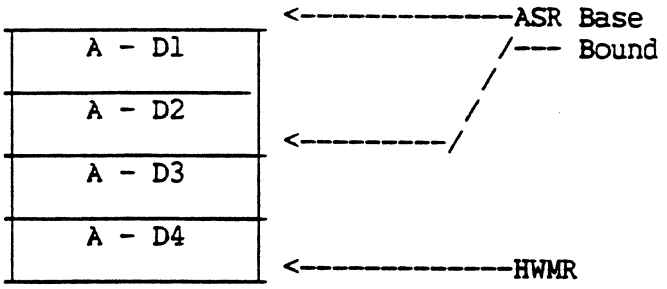
EXAMPLE: This example illustrates how the HWMR protects the program descriptors from one program from being accessed by another program.

1. Program A stores four descriptors on the argument stack (SDRn)

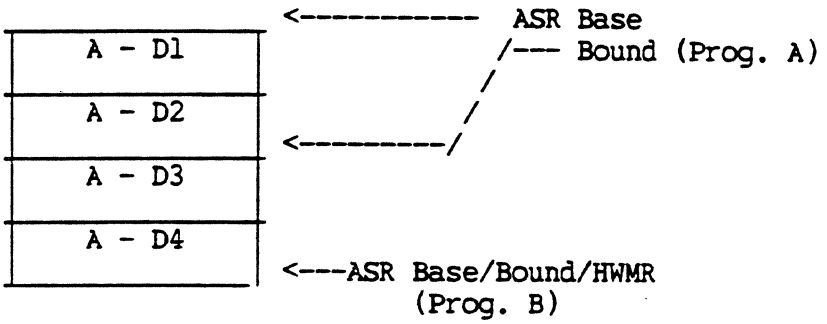


2. Executes an LDDn to copy the ASR to DRn

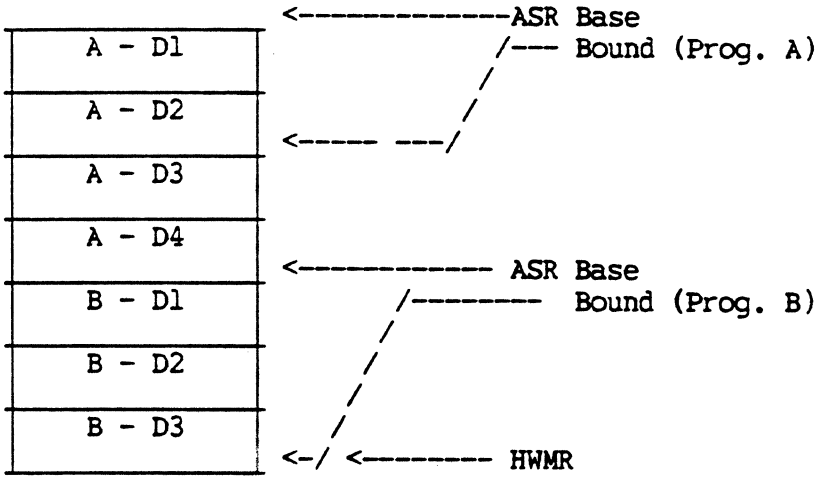
3. Executes a PAS instr. to modify the ASR bound



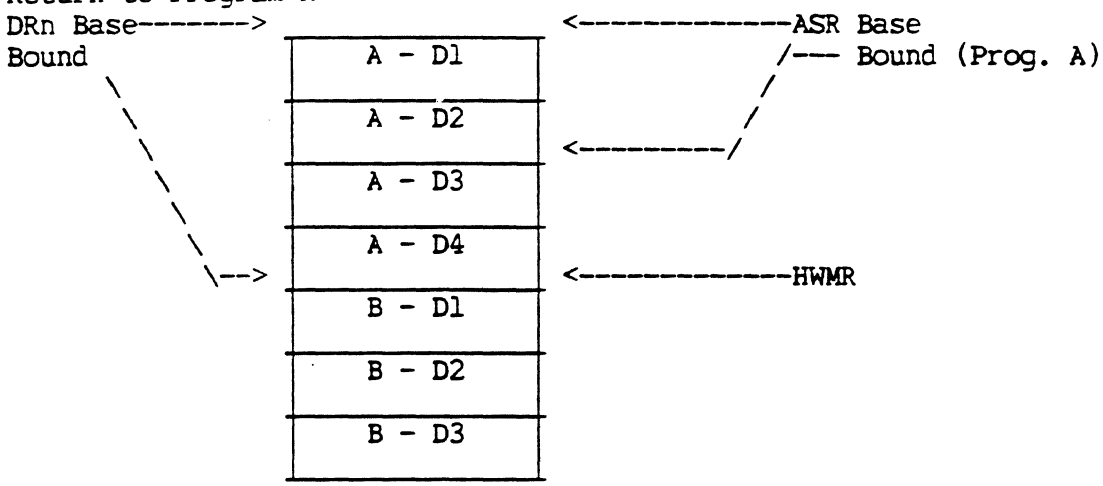
4. Calls program B



5. Program B stores three descriptors on the argument stack



6. Return to Program A



Because the HWMR remembers the highest level reached in the argument stack by an individual program, and uses it to generate the ASR base for a new program, there can be no overlap of descriptors in the argument stack. Security cannot be violated.

e. Loading the Pointer Registers

If type 11 entry descriptor was referenced by the S and D fields of the ICLIMB instruction, all pointer registers are set to the value of the target IS as follows:

ISR --> DR0 through DR7

SEGID (IS) --> SEGID0 through SEGID7

00....0 --> AR0 through AR7

NOTE: When the entry descriptor type is other than T = 11, the pointer register content remains unchanged. However, unless the ISR is copied into the DRn with the ICLIMB instruction altering the ISR bit 24, the content of ARn, and SEGIDn is undefined.

f. Loading X0/GX0

- o If bit 18 of the C field of a CLIMB instruction is 1 and the operation is an interdomain transfer, the load is as follows.

Old ISR Bit 24	New ISR Bit 24	X0	GX0
0	0	*C(X0) <--Y_{0-17}	Undefined (meaningless)
0	1	*Undefined (meaningless)	$C(GX0)_{0-17}$ <-- 0 $C(GX0)_{18-35}$ <-- Y_{0-17}
1	0	C(X0) <--Y_{16-33}	**Undefined (meaningless)
1	1	Undefined (meaningless)	**$C(GX0)_{0-1}$ <-- 0 $C(GX0)_{2-35}$ <-- Y_{0-33}

- o If X0 is to be stored in the safe store stack, the content of X0 at the start of a CLIMB instruction is stored.
- o If GX0 is to be stored in the safe store stack, the content of GX0 at the start of a CLIMB instruction is stored.
- o If bits 18 of the C field of a CLIMB instruction is 0, or the operation is not an interdomain transfer, the load is as shown below.

Old ISR Bit 24	New ISR Bit 24	X0	GX0
0	0	Unchanged	Unchanged (meaningless)
0	1	Unchanged (meaningless)	$C(GX0)_{0-17}$ <- 0 $C(GX0)_{18-35}$ <- C(X0)
1	0	C(X0)<-- $C(GX0)_{18-35}$	Unchanged (meaningless)
1	1	Unchanged (meaningless)	Unchanged

The above table also applies to the fault/interrupt CLIMB.

NOTE: When the CLIMB instruction alters bit 24 of the ISR, the content of X1-X7/GX1-GX7 is undefined.

6. Setting Mode Indicators for System Entry CLIMB

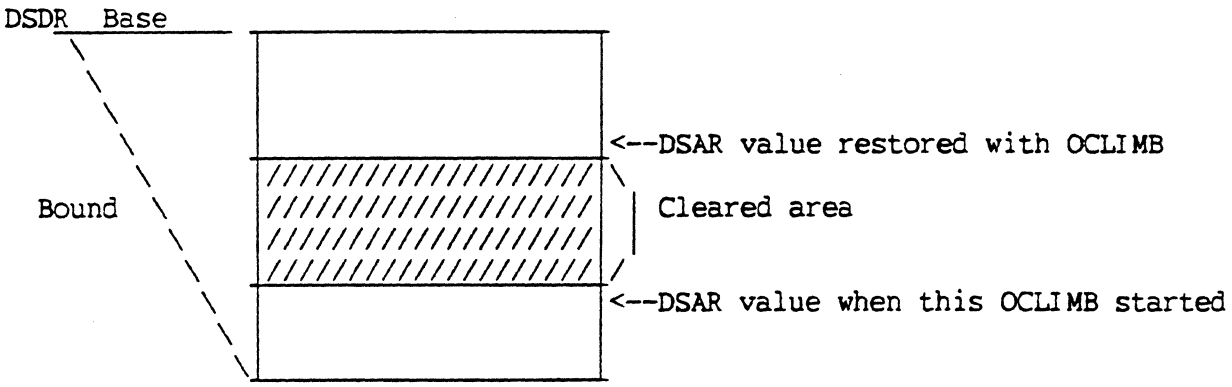
When the CLIMB is a system entry (PMME) where S = 0 and D = 1760 (octal), the Master mode indicator is set to ON. If it is not a system entry and bit 19 of the C field equals 0, the processor is set to Slave mode and the Master mode indicator is set to OFF. If it is neither, the mode remains unchanged. When this CLIMB is executed as a response to a fault or an interrupt, the Master mode indicator is always set to ON.

Outward CLIMB (RET/OCLIMB) C Field Bits 22 and 23 = 01

1. In the OCLIMB version of the CLIMB instruction, a return occurs according to the last frame stored in the safe store stack.

2. The E, P, S, and D fields, and bits 19, 20, and 21 of the C field are ignored.

3. The data stack clear flag (DSCF) of the option register is checked. When DSCF = 1, the data stack area used with the procedure executing the outword CLIMB is cleared. The cleared area is shown by shading in the diagram below.



- In this case, a security fault, class 1 occurs if the DSAR at the start of the CLIMB is less than the restored DSAR.

- If a missing page fault occurs while the data stack is being cleared, the hardware saves the state at the time the fault occurred. When the operating system loads this missing page and returns to the executing procedure, the clearing of the data stack area is re-executed correctly.

CLIMB

4. When an OCLIMB starts, the SCR value determines the number of registers allowed. Registers are restored with the SCR content indicated in the list below.

   An IPR fault occurs if the option register safe store bypass flag (SSBF) is ON at the time.

   When the SCR = 00 (binary), the following registers are restored:

   Instruction Counter (IC)

   Indicator Register (IR)

   Stack Control Register (SCR)

   Instruction Segment Identity Register - SEGID(IS)

   Data Stack Address Register (DSAR)

   Instruction Segment Register (ISR)

   Linkage Segment Register (LSR)

   Argument Stack Register (ASR)

   Parameter Segment Register (PSR)

   When SCR = 01 (binary), all the registers that meet the checks for SCR = 00 (binary) are restored, plus AR 0-7 and SEGID 0-7.

   When SCR = 10 or 11 (binary), the registers for SCR = 01 (binary), the DR0-DR7, X0-X7/GX0-GX7, A, Q, E, and LOR are restored. When word 5, bit 9 of the safe store stack is 1, the pointers and lengths register and the fault recovery information are restored.

   In all cases, the processor number and the timer register are not restored.

CLIMB

5. The base and bound values of the safe store register (SSR) are adjusted according to the new values placed in the SCR from the safe store stack as follows:

SCR(bin.)	Base	SSR Bound
00	-16 words	+16 words
01	-24 words	+24 words
10	-80 words	+80 words
11	-64 words	+64 words

6. Loading DR0-DR7

   When an OCLIMB uses 16 or 24 words for the safe store stack (i.e., the old SCR value = 00 or 01) and then transfers to Slave mode, the new ISR value is loaded into DR0-DR7.

7. Loading X0/GX0

   When the OCLIMB instruction C field bit 18 = 1, the effective address specified with the instruction, in accordance with bit 24 of the ISR restored from the safe store stack, is loaded into X0/GX0. (Refer to table for loading X0/GX0 when bit 18 = 1 under ICLIMB.)

   When the OCLIMB instruction C field bit 18 = 0, with a 64-word or 80-word safe store stack, the safe store stack content is restored into X0/GX0. With other than a 64-word or 80-word safe store stack, the content of X0/GX0 is determined as shown in the table for loading X0/GX0 when bit 18 = 0 under the ICLIMB discussion.

   NOTE: When the contents of X1-X7/GX1-GX7, ARn, and SEGIDn are not restored with the OCLIMB instruction that alters bit 24 of the ISR, those contents are undefined.

8. The IC is restored from the safe store stack as follows:

   From NS or ES to NS or ES mode

   Word 4_{0-17} --> $C(IC)_{0-17}$

   The HWMR is restored from word 0_{0-19}

CLIMB

9. Control is passed to the instruction indicated by the IC and ISR.

10. When the indicator register is restored (with the value stored in the safe store stack), the Master mode bit may be set to ON.

11. Outward CLIMB is interruptible during execution when the following conditions are satisfied.

    - The option register data stack clear flag (DSCF) = 1.
    - The interrupt inhibit bit = 0 (bit 28 of the first word of the instruction).
    - If the interrupt inhibit bit = 1, interrupt is not permitted for this instruction during execution. Interpretation of bit 28 is only valid at the time of outward CLIMB. With the other three CLIMB variations, interrupt is not accepted during execution and the value of bit 28 is not affected by execution of the instruction.
    - The procedure executing this outward CLIMB used the data stack area.

      If there is no area to be cleared (i.e., if the restored DSAR value is equal to the current DSAR value) despite the above two conditions being satisfied, this OCLIMB is not interruptible during execution.

      When the OCLIMB is being executed and the above three conditions are satisfied, the processor samples interrupt at suitable times and responds to any interrupt received, to ensure that a Lockup fault does not occur while the data stack is being cleared. At response to the interrupt, the processor saves the current state in the safe store stack and the interrupted OCLIMB is re-executed normally. The clear operation is restarted correctly from the point at which it was interrupted.

CLIMB

Lateral Transfer (LTRAS/GCLIMB) C Field Bits 22 and 23 = 10

In the GCLIMB version of the CLIMB instruction, the safe store register and the parameter segment register remain unchanged. Also, the base and bound of the argument stack register remain unchanged. The HWMR is not stored in the safe store stack.

1. The bit in the E field is not interpreted and the SCR remains unchanged.

2. The GCLIMB may be an inter- or intradomain transfer that is determined by the descriptor referenced in the S and D fields. This version functions as the ICLIMB, except as indicated. Since the state of the processor is not saved, control cannot return to an instruction executing the GCLIMB.

3. Because the processor state is not saved, the procedure executing the GCLIMB cannot return correctly with an OCLIMB.

If the descriptor referenced by the S and D fields of the GCLIMB instruction is a type 11 descriptor, the pointer registers are set to the state of the target instruction segment. When the type is not 11, the pointer register remains unchanged. If T is not 11 when the GCLIMB instruction is altering bit 24 of the ISR, the pointer registers are undefined.

Lateral Transfer (PCLIMB/LTRAD) C Field Bits 22 and 23 = 11

The execution of the PCLIMB version is identical with that of ICLIMB, except for the following:

1. The CPU state is not saved in the safe store stack.

2. The HWMR is not saved in the safe store stack.

3. The SCR remains unchanged.

If the descriptor referenced by the S and D fields is a type 11 descriptor, the pointer registers are set to the state of the target instruction segment. When the type is not 11, the pointer register remains unchanged. If T is not 11 when the PCLIMB instruction is altering bit 24 of the ISR, the pointer registers are undefined.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: XEC or XED

INDICATORS: Master Mode - See notes below and discussion of "C19, Slave Mode" in earlier pages of the CLIMB explanation.

NOTES: 1. Any of the following conditions cause an IPR fault:

- o If illegal repeats or executes precede modifications
- o If illegal address modification is used
- o If the base and bound fields of the instruction segment descriptor are not modulo 32 bytes, of if flag bit 27 = 1 (bound valid) and the bound is not 31 modulo 32 bytes
- o If the S and D fields are S = 0 and D = 1760 or 1761 (octal), and the descriptor from the system entry location is not an entry descriptor
- o If the descriptor referenced in the S and D fields is not a standard, entry, or dynamic linking descriptor (T = 0, 5, 8, 9, or 11)
- o If the type of the descriptor referenced with the S and D fields is T = 1 or 3, and the segment descriptor obtained from this descriptor is not an entry or dynamic linking descriptor
- o If the S and D fields of the vector or the CLIMB instruction are S = 0 and D = 1761 (octal)
- o If the transfer destination ISD type T is not 0

CLIMB

- o If a normal or extended shrink is specified for a segment descriptor placed in the address segment and the pushed segment descriptor type is illegal (T = 5, 7 to 11, 13, 15)
- o If a Return Climb is specified and the safe store bypass flag in the Option Register = 0
- o If E = 1 and DR0 contains a descriptor of type T = 3, 5, or 7-11, 13, or 15
- o If the S and D fields of the vector are S = 0 and D = 1760 (octal)

2. A Command fault may occur as follows.

- o If the S and D fields of the vector are S = 0 and D = 1763 or 1764 (octal) and the processor is not in Privileged Master mode
- o If WS 0 is specified and the processor is not in the Privileged Master mode
- o If WSR 0 is specified and the processor is in Slave mode (except during the access for the ISD when the system entry (PMME) is specified)

3. A Bound fault may occur as follows:

- o If in the ICLIMB version of the instruction, field E = 1, DR0 type = 1, and (P + 1) is greater than the DR0 bound
- o If the transfer destination ISD flag (bit 27) of the instruction segment descriptor is 0 (empty segment)
- o If a carry occurs in forming a new argument stack register (ASR) or parameter segment register (PSR) base
- o If an access for a vector or a descriptor exceeds the upper or lower bounds of the specified segment, or if the bound is not valid (flag bit 27 = 0), or if there is an attempt to address the argument (for the push) and the temporary ASR bound + 1 byte > 8192 bytes

- o If on an access to memory an associative memory error occurs

4. A Security Fault, Class 1 may occur as follows.

   - o If the ISD flag bit 26 = 1 (Privileged mode) and the processor is in Slave mode and the CLIMB did not result from a fault, interrupt, or system entry (PMME)
   - o If, at the end of the CLIMB, ISR flag bit 26 = 1 (Privileged) and either indicator register bit 28 = 0 (Slave) or a nonhousekeeping page is accessed for the next instruction
   - o If at the end of the CLIMB, indicator register bit 28 = 0 (Slave) and a housekeeping page is accessed for a vector
   - o If the page to be accessed is a nonhousekeeping page (PTW flag bit 32 = 0)

5. A Security Fault, Class 2 may occur as follows;

   - o If flag bit 25 of the instruction segment descriptor is 0 (no execute permission)
   - o If read flag bit 20 of the descriptor = 0 for any access to a segment for a vector or descriptor (but not ASR)
   - o If a working space violation occurs
   - o If the specified page (for the push to ASR) does not have write permission

   NOTE: In the SDR<u>n</u> instruction, the ASR needs neither write nor store permission.

6. A Missing Segment fault occurs if flag bit 28 of the descriptor = 0 for any access to the ASR, or to a segment for a vector or descriptor.

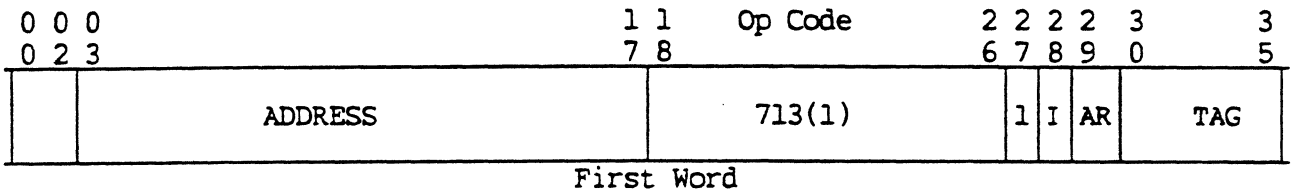
7. A Missing Page fault occurs for any access to the ASR, or to a segment for a vector or descriptor, if flag bit 30 of the PTW for the accessed page = 0.

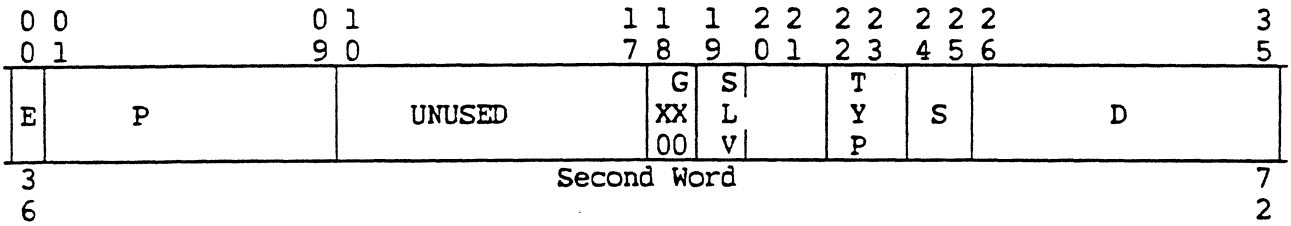
8. A Missing Space fault occurs for any access to the ASR, or to a segment for a vector or descriptor, if bit 20 of the PTDW = 0.

9. A Safe Store Stack fault occurs if the SSR bound < 239 words + 3 bytes as a result of the SSR update adjustment.

10. When the access of the ISD from the LSD formed from the entry descriptor, the same fault checks are made as listed above, except that if the CLIMB resulted from a fault, interrupt, or system entry (PMME), the WS = and WSR 0 Command fault checks are not made. (The entry descriptor does not contain flag bits 20, or 27.)

SUMMARY OF CLIMB INSTRUCTION FORMAT:





The control fields are defined as follows:

E = 0 - No parameters are passed

E = 1 - Pass P+1 parameters (ICLIMB, PCLIMB only)

P = N-1 - Number (minus 1) of descriptions or vectors to pass if E = 1

X0/GX0 = 0 - CLIMB will not affect X0/GX0.

X0/GX0 = 1 - If entry descriptor (T = 8, 9, or 11) is referenced or OCLIMB is executed, X0/GX0 is loaded with the effective address designated by the address tag and AR fields of the CLIMB instruction.

SSLV = 0 - Set Slave mode

SLV = 1 - Do not change Master mode indicator

TYP = 00 - ICLIMB (or PMME)

TYP = 01 - OCLIMB

TYP = 10 - GCLIMB (LTRAS) - Transfer with same ASR and PSR. Do not save processor state.

TYP = 11 - PCLIMB (LTRAD) - Transfer with new ASR and PSR. Do not save processor state.

S,D - Target SEGID

CODING FORMAT:

Coding of a CLIMB varies with the version of the CLIMB instruction being executed.

The following list contains each of the five versions of the CLIMB instruction with their respective fields, which are defined below. The underlined fields are required; all others are optional.

ICLIMB - entry, count, effective address, flags

PCLIMB - entry, count, effective address, flags

GCLIMB - entry, effective address, flags

OCLIMB - effective address

PMME - effective address, count, flags

The fields in the CLIMB instruction are described below:

entry - Name of an entry or a 12-bit number (SEGID) that identifies a descriptor specifying a new linkage segment and instruction segment or the same linkage segment and an instruction segment.

CLIMB

count - Decimal expression representing a value in the range 0 <= count <= 512. This value indicates the number of parameters or descriptors (one for each argument) pointed to by PR0. The first of these is at the location indicated by pointer register zero. A value of zero means that no arguments, and consequently no vectors or descriptors, are present. If no value is given, zero is assumed.

effective address - The effective address may include a tag pointer designation. When this occurs, the field must be enclosed by parentheses; e.g., (address, tag) or (address, tag, pointer). The effective address is used to establish the next instruction location, but only when the entry identifies a descriptor that does not specify a linkage segment. The effective address is a requirement only for the PMME version to designate the Master mode entry.

If the entry identifies a descriptor that specifies a linkage segment (entry descriptor), index register 0 may be loaded with the effective address. If the entry identifies a descriptor that does not specify a linkage segment (standard descriptor), this address is added to the base of the instruction segment (described in the descriptor) to establish the next instruction location and may be loaded in index register 0. If bit 18 of field C is zero or this address is omitted, the content of the effective address field is not loaded in index register 0. Note that an explicit zero is required to load index register 0 with a zero, since a null field prevents register loading.

MASTER - Sets bit 19 of the second word

No flags are used for the OCLIMB version.

NOTE: PMME is synonymous with ICLIMB with 1760_8 coded in the entry field.

flags

EAX0 - Sets bit 18 of the second word
The keyword EAX0 indicates that the effective address field is to be loaded in index register 0 or general index register 0.

NEAX0 - Clears bit 18 of the second word

SLAVE - Clears bit 19 of the second word (for PMME, bit 18 of the second word is forced on, bit 19 is ignored by the hardware)

The keyword SLAVE indicates that the processor will enter Slave mode upon change of domain. If this field is omitted, the mode is not changed, except for the PMME version which is always set to Privileged Master mode.

If both keywords are needed, the field must be enclosed by parentheses with a comma separating the keywords: (e.g., EAX0, SLAVE).

EXAMPLES:

```
1       8       16                  32
*                                   ICLIMB
        INHIB   OFF
ODDF    NULL
NEPR1   LDD     P0,DSTKS            shrink data stack (64 words)
        SDR     P0
        LDD     P1,ODRSH            shrink safe store
        SDR     P1
        LDD     P1,IALPS            ISR,ASR,LSR,PSR
        SDR     P1
        LDD     P1,ISRS             ISR (R,W)
        MLR     (1),(1)             copy safe store frame to data stack
        ADSC9   0,0,256,P.SSR
        ADSC9   0,0,256,P0
        LDP     P0,.ASR,DL          copy ASR to P0
        ICLIMB  .DR+4,3,,SLAVE      climb exception procedure
*       VFD     18/,09/713,1/1,1/0,1/0,6/M.
*       VFD     1/1,9/3-1,8/0,1/.N,1/.O,2/0,2/0,12/.DR+4
        .
        .
```

```
1       8       16                      32
*                                       GCLIMB/ICLIMB
        INHIB   ON
TRVCEL  NULL
        TRA     2,IC
        NOP     ,DL
        EPPR0   1,IC                    .TROPN (system domain only)
        TRA     .CRTRV+12,,P.CR
        EPPR0   1,IC                    .TROPN none (system domain only)
        TRA     2,IC
        EPPR0   1,IC                    .TROPN all (slave domain)
        TRA     .CRTRV+14,,P.CR
TRVC01  LDP7    **,DL                   .TRPUT (system domain)
        TRA     TPUTSY-..DISP,,P7
        NOP     ,DL                     *.TROPN all macros removed
        NOP     ,DL
TRVC03  GCLIMB  **,TOPNG                .TROPN extension
*       VFD     18/TOPNG,09/713,1/1,1/0,1/0,6/M.
*       VFD     1/0,9/0,8/0,1/.N,1/.O,2/0,2/2,12/**
        LDD6    DP.OTE,,P.SSL           .TROPN all for slave domain extension
        ICLIMB  .DR6
*       VFD     18/,09/713,1/1,1/0,1/0,6/M.
*       VFD     1/0,9/0,8/0,1/.N,1/.O,2/0,2/0,12/.DR6
        TRA     0,,P0
```

CMG	Compare Magnitude	405 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: |C(A)| :: |C(Y)|; C(A), C(Y) unchanged

EXPLANATION: This instruction compares the magnitude of signed algebraic numbers. For example, if -1 and +1 are compared, they are considered equal and the zero indicator is set ON.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero	Negative	Relationship
0	0	$\|C(A)\| > \|C(Y)\|$
1	0	$\|C(A)\| = \|C(Y)\|$
0	1	$\|C(A)\| < \|C(Y)\|$

CMK	Compare Masked	211 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35,

$C(Z)_i = \overline{C(Q)_i}$ AND $[C(A)_i$ XOR $C(Y)_i]$

C(A), C(Q), C(Y) unchanged

EXPLANATION: This instruction compares the corresponding bit positions in C(A) and C(Y) to determine whether they are equal or not. Bits for which the corresponding bit of Q is 1 are masked and not compared.

The zero indicator is set ON if the comparison is successful for all bit positions;

if for all i = 0,1,...,35
either $C(A)_i = C(Y)_i$
or
$C(Q)_i = 1$ is established.
Otherwise, the zero indicator is set OFF.

The negative indicator is set ON if the comparison is unsuccessful for bit position 0;

if for $C(A)_0 \neq C(Y)_0$
and
$C(Q)_0 = 0$
Otherwise, the negative indicator is set OFF.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

Negative - If bit 0 of C(Z) = 1, then ON; otherwise, OFF

EXAMPLE: In the following example, the comparison is equal after execution of CMK, and the TZE exit is taken. Only the 2s in NUMBER and DATA are compared; all other bits are masked by 1s in the Q-register.

```
1       8       16
        LDQ     MASK
        LDA     NUMBER
        CMK     DATA
        TZE     OUT

MASK    OCT     777777777707
NUMBER  OCT     300333333326
DATA    OCT     666666666625
```

CMPA	Compare with A-Register	115 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(A) :: C(Y); C(A) and C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Algebraic comparison (Signed Binary Operands)

Zero	Negative	Carry	Relationship	Sign
0	0	0	C(A) > C(Y)	$C(A)_0$=0,$C(Y)_1$=1
0	0	1	C(A) > C(Y)\	
1	0	1	C(A) = C(Y)	>$C(A)_0$=$C(Y)_0$
0	1	0	C(A) < C(Y)/	
0	1	1	C(A) < C(Y)	$C(A)_0$=1,$C(Y)_0$=0

Logical comparison (Unsigned Positive Binary Operands)

Zero	Carry	Relationship
0	1	C(A) > C(Y)
1	1	C(A) = C(Y)
0	0	C(A) < C(Y)

CMPAQ	Compare with AQ-Register	117 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(AQ) :: C(Y-pair); C(AQ) and C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Algebraic comparison (Signed Binary Operands)

Zero	Negative	Carry	Relationship	Sign
0	0	0	C(AQ) > C(Y)	$C(AQ)_0$=0,$C(Y\text{-}pr)_0$=1
0	0	1	C(AQ) > C(Y)\	
1	0	1	C(AQ) = C(Y)	>$C(AQ)_0$=$C(Y\text{-}pr)_0$
0	1	0	C(AQ) < C(Y)/	
0	1	1	C(AQ) < C(Y)	$C(AQ)_0$=1,$C(Y\text{-}pr)_0$=0

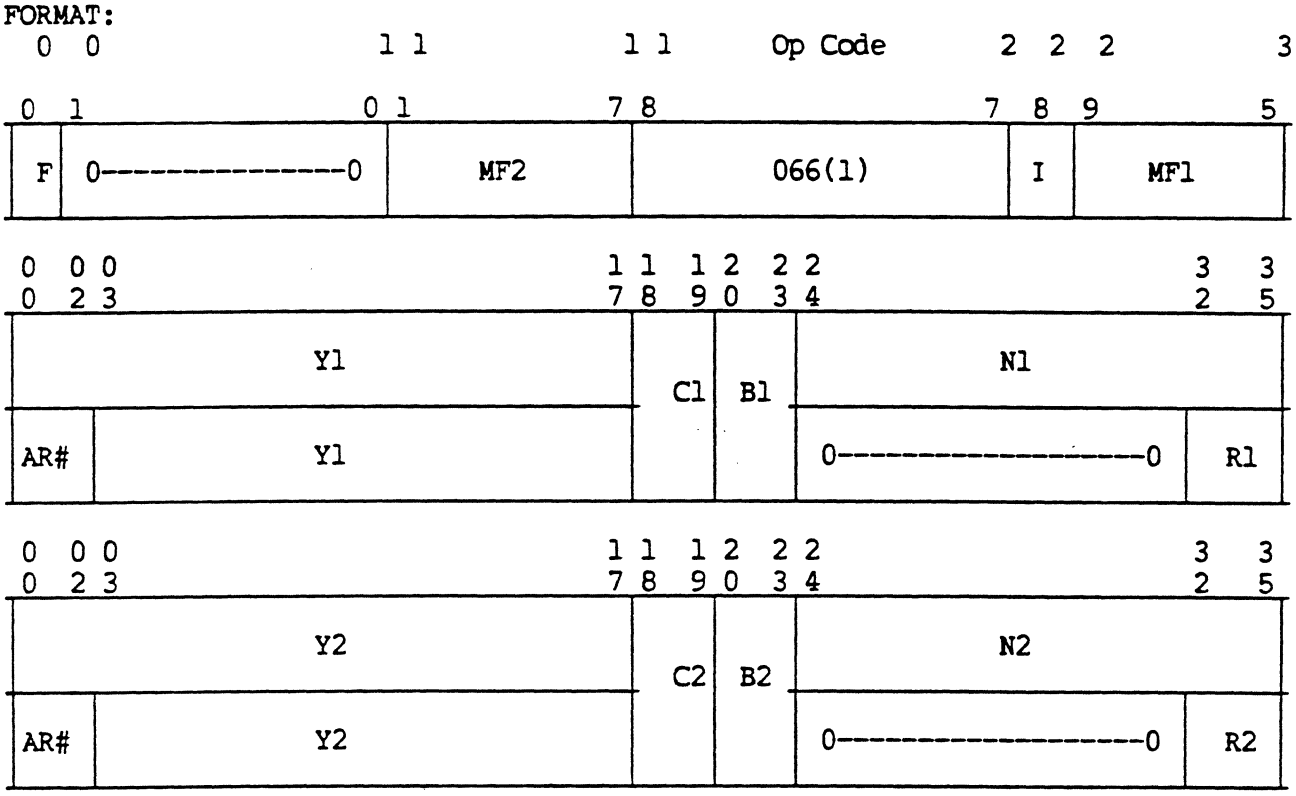
Logical comparison (Unsigned Positive Binary Operands)

Zero	Carry	Relationship
0	1	C(AQ) > C(Y-pr)
1	1	C(AQ) = C(Y-pr)
0	0	C(AQ) < C(Y-pr)

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

CMPB	Compare Bit Strings	066 (1)

FORMAT:



CODING FORMAT: The CMPB instruction is coded as follows:

```
1       8       16
_____________________

        CMPB    (MF1),(MF2),F
        BDSC    LOCSYM,N,C,B,AM
        BDSC    LOCSYM,N,C,B,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) :: C(string 2)

EXPLANATION: The string of bits starting at location YCB_1 is logically compared with the string of bits starting at location YCB_2 until an inequality is found or until the larger tally (L1 or L2) is exhausted. If L1 is not equal to L2, the fill bit (F) is used to pad the least significant bits of the shorter string. The contents of both strings remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero	Carry	Relationship
0	0	C(string 1) < C(string 2)
1	1	C(string 1) = C(string 2)
0	1	C(string 1) > C(string 2)

NOTES:

1. If L1 or L2 = 0, both the Zero and Carry indicators are turned ON, but no Illegal Procedure fault occurs.
2. An Illegal Procedure fault occurs if DU or DL modifications are used for MF1 or MF2 or if an illegal repeat is used.

EXAMPLES:

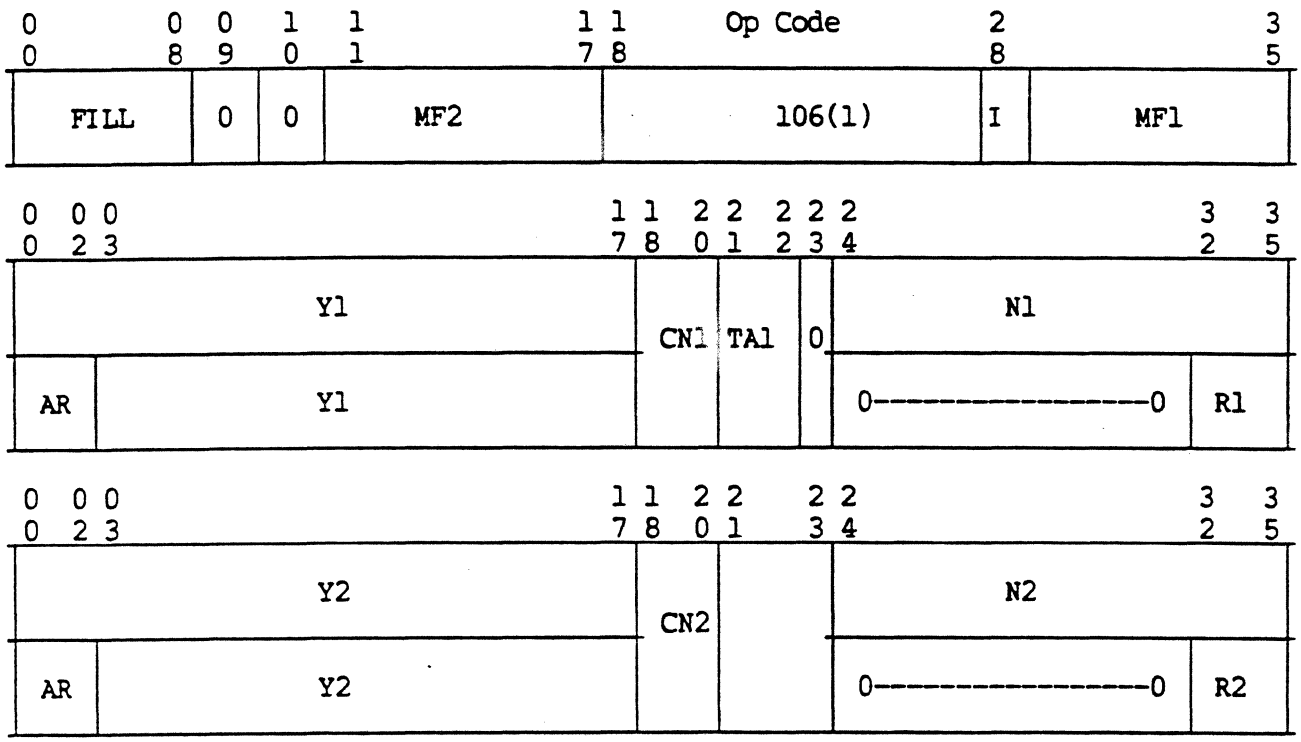
```
1       8       16                32
                                  
        CMPB    ,,1               fill bit 1 option
        BDSC    FLD1,45,0,0       FLD1 operand descriptor
        BDSC    FLD2,48           FLD2 operand descriptor
        TRC     EQU.GR            FLD1 equal/greater than FLD2
        USE     CONST.            bits compared (octal representation)
FLD1    OCT     0,777000000000    0 0 0 0 0 0 0 0 0 0 0 0 7 7 7 7
FLD2    OCT     0,777000000000    0 0 0 0 0 0 0 0 0 0 0 0 7 7 7 0
        USE                       Result - FLD1  >  FLD2

        CMPB                      no options
        BDSC    FLD1,36,0,0       FLD1 operand descriptor
        BDSC    FLD2,19,1,3       FLD2 operand descriptor
        TZE     EQUAL             FLD1 = FLD2
        TRC     FLD1GR            FLD1 > FLD2
        TRA     FLD1LS            FLD1 < FLD2
        USE     CONST.            bits compared (octal representation)
FLD1    VFD     18/-1             7 7 7 7 7 7 0 0 0 0 0 0
FLD2    VFD     12/0,19/-1        7 7 7 7 7 7 4 0 0 0 0 0
        USE                       Result - FLD1  <   FLD2
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16                32
                                  
        EAX2    12                load FLD1's bit modifier into X2
        EAX6    6                 load FLD1's length into X6
        EAX4    FLD1              load FLD1's address into X4
        AWDX    0,4,4             put FLD1's address into AR4
        CMPB    (1,1,,X2),(,,1)   with modification
        BDSC    0,X6,0,0,4        FLD1 operand descriptor
        ARG     INDSCR            pointer to FLD2's indirect descriptor
        TZE     EQUAL             FLD1 = FLD2
        USE     CONST.            bits compared    memory contents
FLD1    VFD     12/0,6/1               7 7 0       000077000000
FLD2    VFD     24/0,6/1               7 7 0       000000007700
INDSCR  BDSC    FLD2,9,2,6        FLD2 indirect operand descriptor
        USE                       Result - FLD1   =   FLD2
```

CMPC	Compare Alphanumeric Character Strings	106 (1)

FORMAT:



CODING FORMAT: The CMPC instruction is coded as follows:

```
1       8       16
        CMPC    (MF1),(MF2),FILL
        ADSCn   LOCSYM,CN,N,AM
        ADSCn   LOCSYM,CN,N,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) :: C(string 2)

EXPLANATION: Starting at location YC1, the string of alphanumeric characters of type TA1 is logically compared with the string of alphanumeric characters of assumed type TA1 that starts at location YC2 until either an inequality is found or until the larger tally (L1 or L2) is exhausted. If L1 is not equal to L2, the FILL character is used to pad the least significant characters of the shorter string. The contents of both strings remain unchanged. Bits 21-23 of descriptor 2 are not interpreted.

Bits 0-8 are compared for the FILL character to be used to pad the least significant characters of the shorter string. If a character string is a 6- or 4-bit character, zeros are inserted at the left of each to produce 9-bit characters for comparison.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero	Carry	Relation
0	0	C(string 1) < C(string 2)
1	1	C(string 1) = C(string 2)
0	1	C(string 1) > C(string 2)

NOTES:

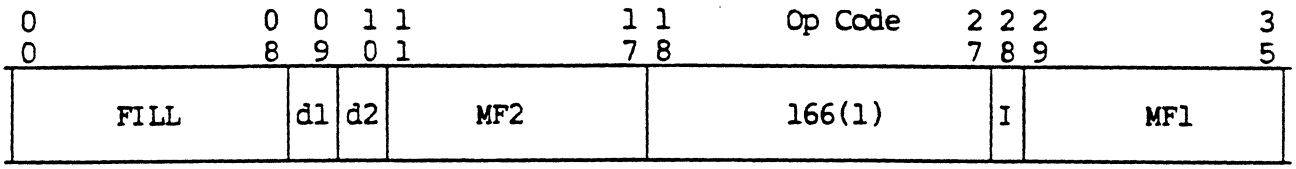
1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2 and if there are illegal repeats.
2. If L_1 or L_2 = 0, the zero and carry indicators are affected as illustrated under Indicators.

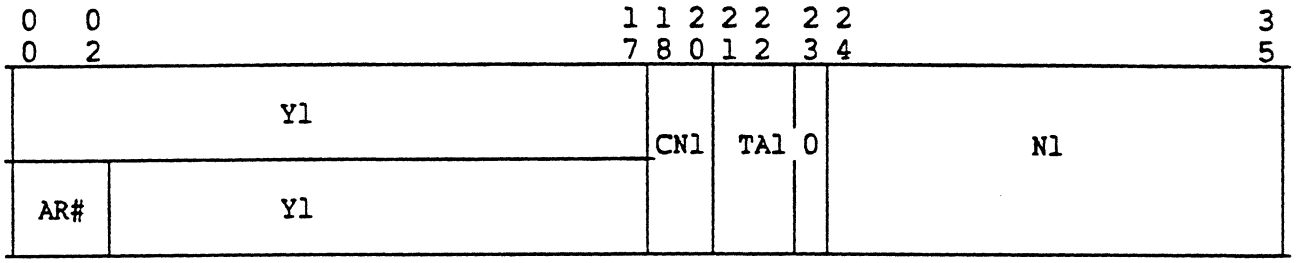
EXAMPLE:

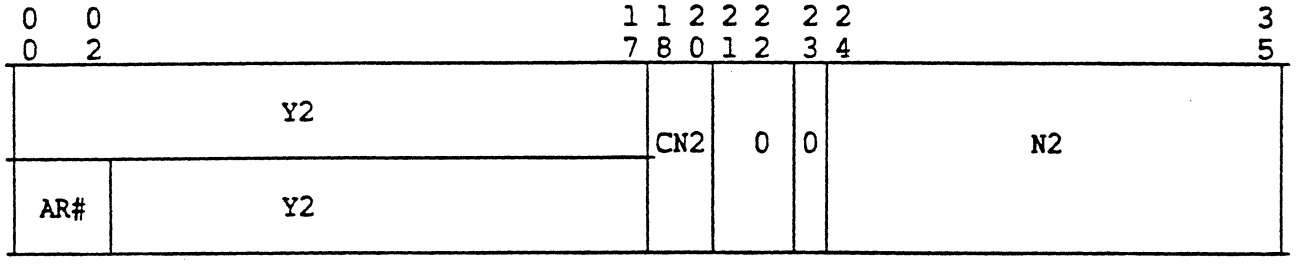
```
1       8       16                  32
        CMPC    ,,020               compare with blank fill
        ADSC6   FLD1,0,6            field 1 operand descriptor
        ADSC6   FLD2,4,4            field 2 operand descriptor
        TZE     EQUAL               both fields equal
        TRC     FLD1GR              field 1 greater
        NULL                        field 1 less
        USE     CONST.              characters compared
FLD1    BCI     1,ABCD              ABCDbb
FLD2    BCI     2,XXXXABCDXXXX      ABCDbb
        USE                         Result - FLD1   =   FLD2
```

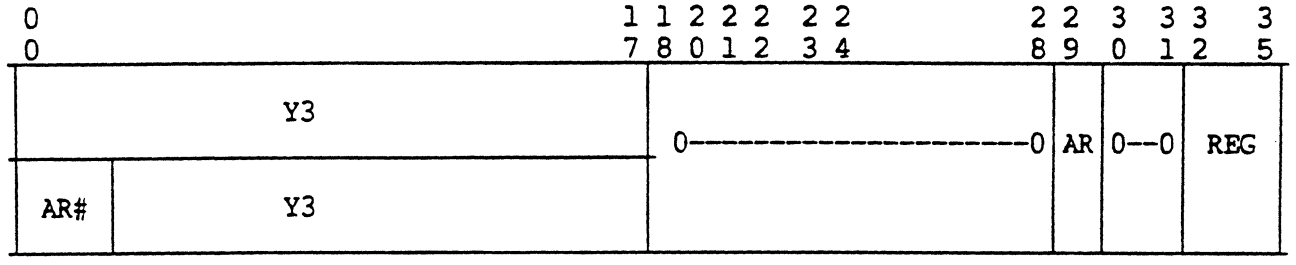
CMPCT	Compare Characters and Translate	166 (1)

FORMAT:









(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

EXPLANATION: Starting at location YC1, the string of alphanumeric characters of type TA1 is logically compared with the string of alphanumeric characters of assumed type TA1 that starts at location YC2, until either an inequality is found or until the larger tally (L1 or L2) is exhausted.

If an inequality is found, the next action depends on d1 and d2. If d1 and d2 = 0, then both characters are transliterated and the resulting characters compared. This is accomplished as follows.

The character from the string starting at YC1 and the character from the string starting at YC2 are each used as an index to a table of 9-bit characters starting at location Y3. The two characters thus taken from the table are compared, the indicators set as indicated below, and the instruction terminates. For the case d1 = d2 = 1, no transliteration takes place; the indicators are set according to the way the two original characters compared. When d1 ≠ d2, one character is translated and the other is not, and then the two characters are compared. For example, if d1 = 1 and d2 = 0, the character from the string starting at YC2 is transliterated (as described above) and compared with the character from the string starting at YC1 and the indicators are set accordingly.

Note that a 9-bit compare is always made. If d1 ≠ d2 and the nontranslated character is a 4- or 6-bit character, then the upper bit positions of the character are zero-filled for the 9-bit compare.

If L1 ≠ L2, fill characters are used to fill the low-order character positions of the shorter string. The contents of both strings remain unchanged.

The transliteration table must begin at a word boundary at character position 0. The index, which is expressed by the number of 9-bit characters, is added to the starting word address of the table. The beginning address of the table is calculated in the same manner as is any normal address modification. However, the computed address is used as word address, with character position ignored, and the index is added to this word address as a 9-bit character number.

Refer to the MVT instruction specifications for details on generating the transliteration table address when address register modification is specified.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Let C1 = C(last char from string 1, translated if d1 = 0)
Let C2 = C(last char from string 2, translated if d2 = 0)

Zero	Carry	Relationship
0	0	C1 < C2
1	1	C1 = C2
0	1	C1 > C2

NOTES:

1. When L1 or L2 = 0, the zero and carry indicators are still affected as indicated in the above table. If L1=L2=0, both the zero and carry indicators are turned ON.

2. A 9-bit character (zero-filled as appropriate) and/or the full 9 bits of the table entry are used in all comparisons.

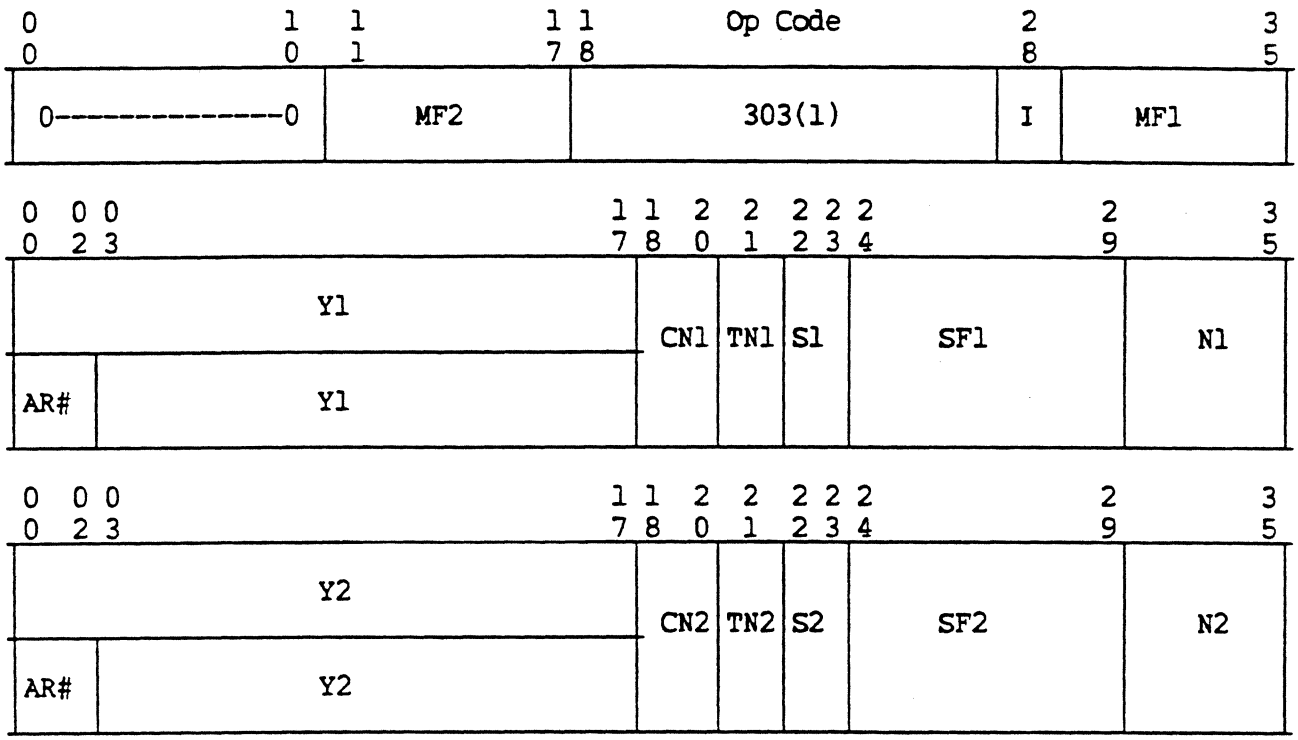
3. The CMPCT instruction is intended for comparisons in situations where the character collating sequence is different from the sequence of character codes.

4. If $L_1 < L_2$, and type TA_1 is 4- or 6-bit, the low-order 4 or 6 bits of the 9-bit FILL character in the instruction are defined as a table index, respectively.

5. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CMPN	Compare Numeric	303 (1)

FORMAT:



CODING The CMPN instruction is coded as follows:

```
1       8       16
        CMPN    (MF1),(MF2)
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) :: C(string 1)

EXPLANATION: Starting at location YC1, the decimal number of data type TN1 and sign and decimal type S1 is algebraically compared with the decimal number of data type TN2 and sign and decimal type S2 that starts at location YC2. The comparison effectively subtracts number 1 from number 2. Zeros (4 bits - 0000) are used to pad the integral and fractional parts of the shorter field. Both numbers remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero	Negative	Relationship
0	1	C(number 1) > C(number 2)
1	0	C(number 1) = C(number 2)
0	0	C(number 1) < C(number 2)

Zero	Carry	Relationship
0	0	|C(number 1)| > |C(number 2)|
1	1	|C(number 1)| = |C(number 2)|
0	1	|C(number 1)| < |C(number 2)|

NOTES:

1. An IPR fault occurs if any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.

2. An IPR fault occurs if the values for the number of characters (Ni) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.

3. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLES:

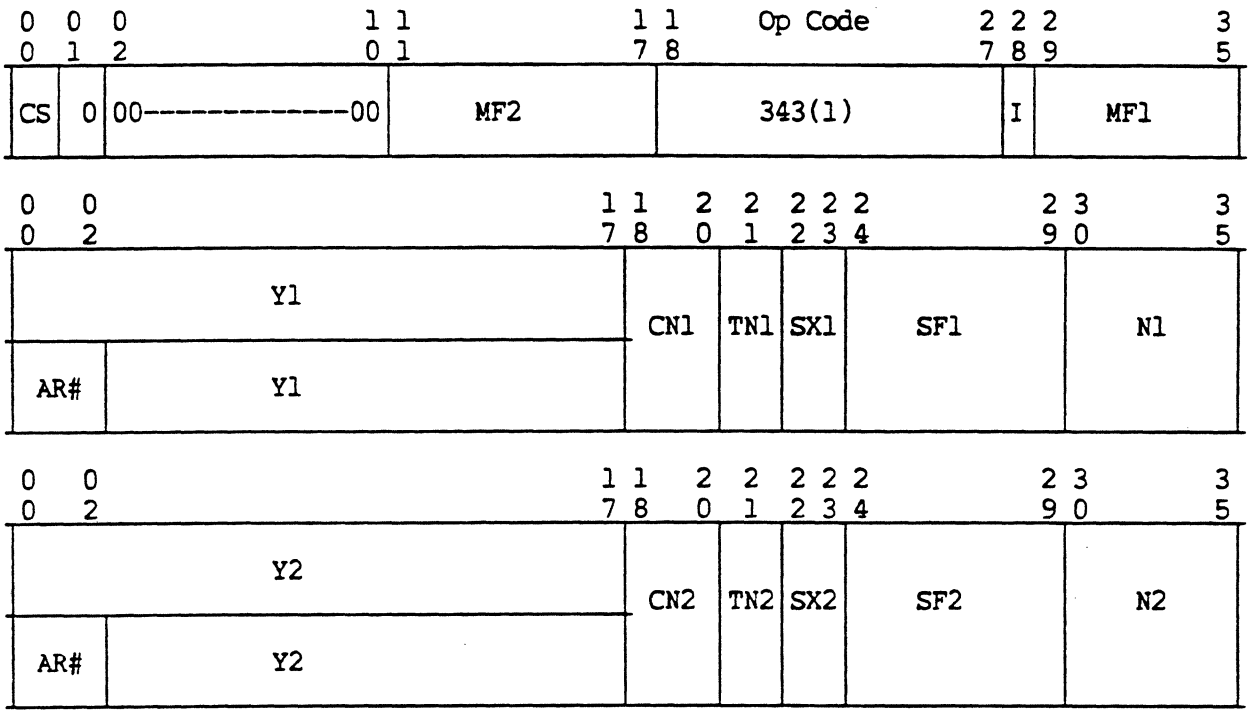
```
1       8       16                  32
        CMPN                        no modification
        NDSC4   FLD1,0,8,1,-2       FLD1 operand descriptor
        NDSC4   FLD2,0,8,0          FLD2 operand descriptor
        TZE     EQUAL               FLD2 = FLD1
        TMI     LESS                FLD2 < FLD1
        TNC     ABS.LT              |FLD2| < |FLD1|
        USE     CONST.              numbers compared
FLD1    EDEC    8P-12345            - 0 0 1 2 3 4 5
FLD2    EDEC    8P-123.45           - 0 0 1 2 3 4 5
        USE                         Result - FLD2 =  FLD1

        CMPN                        no modification
        NDSC9   FLD1,2,2,3          FLD1 operand descriptor
        NDSC4   FLD2,0,8,2,-3       FLD2 operand descriptor
        TZE     EQUAL               FLD2 = FLD1
        TMI     LESS                FLD2 < FLD1
        TRA     GREATER             FLD2 > FLD1
        USE     CONST.              numbers compared
FLD1    EDEC    4A0012              + 0 0 1 2 0 0 0
FLD2    EDEC    8P12000+            + 0 0 1 2 0 0 0
        USE                         Result - FLD2 =  FLD1
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16                  32
        EAX2    2                   load character modifier into X2
        EAX6    6                   load FLD1 length into X6
        EAX4    FLD1                load FLD1 address into X4
        AWDX    0,4,4               put FLD1 address into AR4
        CMPN    (1,1,,X2),(,,1)     with address modification
        NDSC4   0,0,X6,3,-3,4       FLD1 operand descriptor (FLD1,2,6,3,-3)
        ARG     FLD2.I              pointer to FLD2 operand descriptor
        TZE     EQUAL               FLD2 = FLD1
        TPL     MORE                FLD2 > FLD1
        TRA     LESS                FLD2 < FLD1
        USE     CONST.              numbers compared
FLD1    EDEC    8P123456            + 0 0 1 2 3 4 5 6
FLD2    EDEC    8P123456+           + 0 1 2 3 4 5 6 0
FLD2.I  NDSC4   FLD2,0,8,2,-2
        USE                         Result - FLD2 > FLD1
```

CMPNX	Compare Numeric Extended	343 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
CMPNX  (MF1),(MF2),CS
NDSCn  LOCSYM,CN,N,SX,SF,AM
NDSCn  LOCSYM,CN,N,SX,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) :: C(string 2)

EXPLANATION: Starting at location YC1, the decimal number of data type TN1 and sign and decimal type SX1 is algebraically compared with the decimal number of data type TN2 and sign and decimal type SX2 that starts at location YC2. The comparison effectively subtracts number 1 from number 2. Zeros (4 bits - 0000) are used to pad the integral and fractional parts of the shorter field. Both numbers remain unchanged.

The character set is defined by CS (EBCDIC/ASCII).

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero	Negative	Relationship
0	1	C(number 1) > C(number 2)
1	0	C(number 1) = C(number 2)
0	0	C(number 1) < C(number 2)

Carry	Relationship
0	$\lvert C(\text{number 1})\rvert > \lvert C(\text{number 2})\rvert$
1	$\lvert C(\text{number 1})\rvert \leq \lvert C(\text{number 2})\rvert$

NOTES:

1. An IPR fault occurs if any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or if any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.

2. An IPR fault occurs if the values for the number of characters (Ni) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.

3. Refer to the specifications on MVNX for information on coding of overpunched signs.

4. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CMPQ	Compare with Q-Register	116 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Q) :: C(Y); C(Q) and C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Algebraic comparison (Signed Binary Operands)

Zero	Negative	Carry	Relationship	Sign
0	0	0	C(Q) > C(Y)	$C(Q)_0$=0,$C(Y)_0$=1
0	0	1	C(Q) > C(Y)\	
1	0	1	C(Q) = C(Y) >	$C(Q)_0$=$C(Y)_0$
0	1	0	C(Q) < C(Y)/	
0	1	1	C(Q) < C(Y)	$C(Q)_0$=1,$C(Y)_0$=0

Logical comparison (Unsigned Positive Binary Operands)

Zero	Carry	Relationship
0	1	C(Q) > C(Y)
1	1	C(Q) = C(Y)
0	0	C(Q) < C(Y)

CMPXn	Compare with Index Register n		10n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,..., or 7 as determined by op code

C(Xn) :: $C(Y)_{0-17}$; C(Xn) and C(Y) unchanged

ES Mode

For n = 0,1,...,or 7 as determined by op code

C(GXn) :: C(Y); C(GXn) and C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of CMPX0

INDICATORS: Algebraic (signed binary) comparison:

NS Mode

Zero	Negative	Carry	Relationship	Sign
0	0	0	C(Xn) > $C(Y)_{0-17}$	$C(Xn)_0$=0,$C(Y)_0$=1
0	0	1	C(Xn) > $C(Y)_{0-17}$ \	
1	0	1	C(Xn) = $C(Y)_{0-17}$	>$C(Xn)_0$=$C(Y)_0$
0	1	0	C(Xn) < $C(Y)_{0-17}$ /	
0	1	1	C(Xn) < $C(Y)_{0-17}$	$C(Xn)_0$=1,$C(Y)_0$=0

ES Mode

Zero	Negative	Carry	Relationship	Sign
0	0	0	C(GXn) > C(Y)	$C(GXn)_0$=0,$C(Y)_0$=1
0	0	1	C(GXn) > C(Y)	\
1	0	1	C(GXn) = C(Y)	>$C(GXn)_0$=$C(Y)_0$
0	1	0	C(GXn) < C(Y)	/
0	1	1	C(GXn) < C(Y)	$C(GXn)_0$=1,$C(Y)_0$=0

CMPXn

Logical comparison (Unsigned Positive Binary Operands)

NS Mode

Zero	Carry	Relationship
0	1	$C(Xn) > C(Y)_{0-17}$
1	1	$C(Xn) = C(Y)_{0-17}$
0	0	$C(Xn) < C(Y)_{0-17}$

ES Mode

Zero	Carry	Relationship
0	1	C(GXn) > C(Y)
1	1	C(GXn) = C(Y)
0	0	C(GXn) < C(Y)

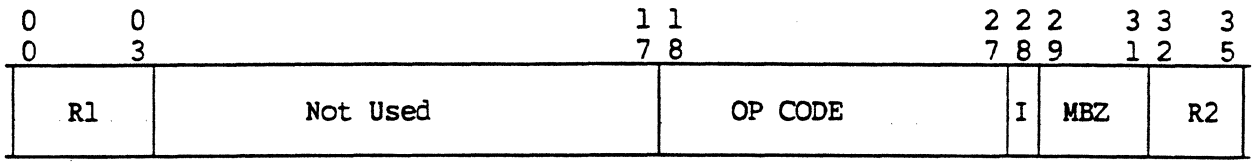
NOTES:

1. When DL modification is specified in the NS Mode, it is executed with all zeros for data.

2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CMRR	Compare Register to Register	534 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
        CMRR    R1,,R2
```

OPERATING MODES: Executes in ES mode only

SUMMARY: R1, R2, = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) : : C(R2)

C(R1), C(R2) unchanged

EXPLANATION: C(R1) is compared with C(R2) and the indicators are set as indicated below.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Algebraic (signed fixed-point) Comparison

Zero	Negative	Carry	Relationship	Sign
0	0	0	C(R1) > C(R2)	$C(R1)_0=0$, $C(R2)_0=1$
0	0	1	C(R1) > C(R2)	\
1	0	1	C(R1) = C(R2)	> $C(R1)_0=C(R2)_0$
0	1	0	C(R1) < C(R2)	/
0	1	1	C(R1) < C(R2)	$C(R1)_0=1$, $C(R2)_0=0$

Logic (unsigned fixed-point) Comparison

Zero	Carry	Relationship
0	0	C(R1) < C(R2)
1	1	C(R1) = C(R2)
0	1	C(R1) > C(R2)

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.

2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

CNAA	Comparative NOT AND with A-Register	215 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35, $C(Z)_i = C(A)_i$ AND $\overline{C(Y)_i}$

C(Q) and C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

Negative - If $C(Z)_0 = 1$, then ON; otherwise, OFF

CNAAQ	Comparative NOT AND with AQ-Register	217 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 71, $C(Z)_i = C(AQ)_i$ AND $\overline{C(Y\text{-pair})}_i$

C(AQ) and C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

Negative - If $C(Z)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

CNAQ	Comparative NOT AND with Q-Register	216 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY For i = 0 to 35, $C(Z)_i = C(Q)_i$ AND $C\overline{(Y)_i}$

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

Negative - If $C(Z)_0 = 1$, then ON; otherwise, OFF

CNAXn	Comparative NOT AND with Index Register n	20n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,...,or 7 as determined by op code

For i = 0 to 17, $C(Z)_i = C(Xn)_i$ AND $\overline{C(Y)_i}$
C(Xn) and C(Y) unchanged

ES Mode

For n = 0,1,...,or 7 as determined by op code

For i = 0 to 35, $C(Z)_i = C(GXn)_i$ AND $\overline{C(Y)_i}$
C(GXn) and C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of CNAX0

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

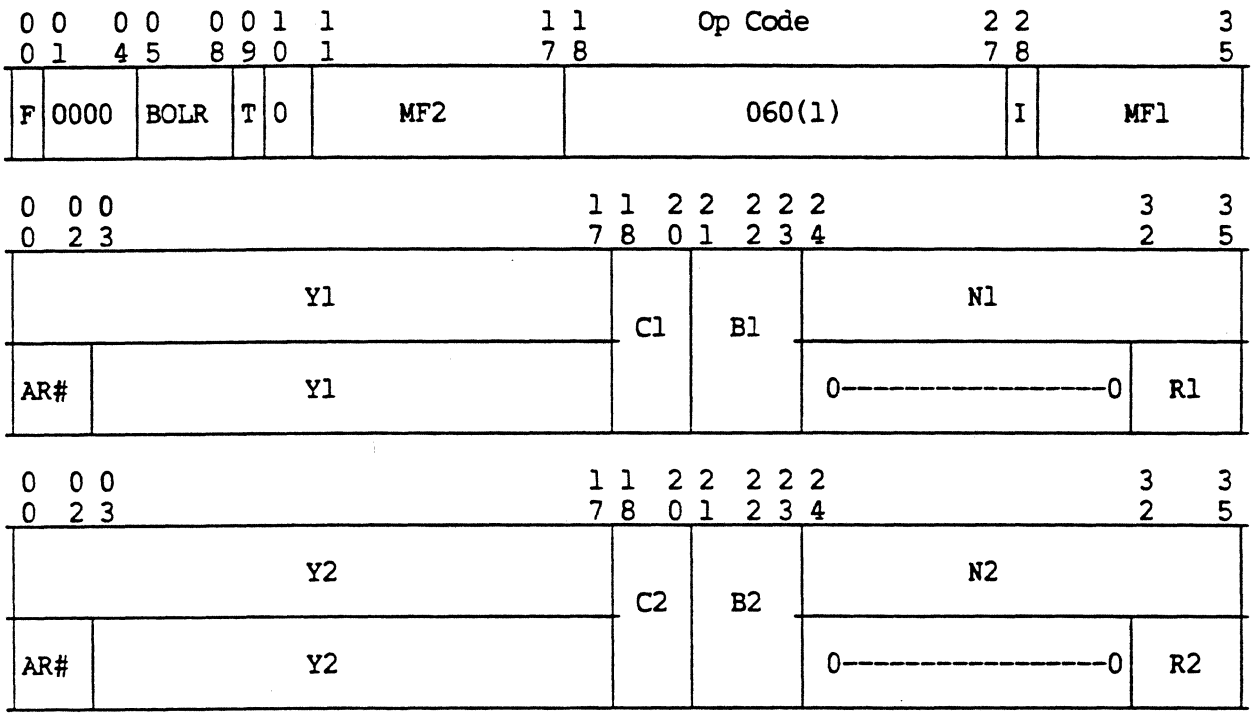
Negative - If $C(Z)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged illegal but executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CSL	Combine Bit Strings Left	060 (1)

FORMAT:



CODING FORMAT: The CSL instruction is coded as follows:

```
1       8       16
        CSL     (MF1),(MF2),BOLR,F,T
        BDSC    LOCSYM,N,C,B,AM
        BDSC    LOCSYM,N,C,B,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) : (BOLR) : C(string 2) --> C(string 2)

EXPLANATION: The string of bits starting at location YCB1 is evaluated, bit by bit, with the string starting at location YCB2 and the appropriate bit from the BOLR control field is placed into each corresponding bit of the string starting at location YCB2. If L1 is greater than L2, the least significant L1-L2 bits of string 1 are truncated and the Truncation indicator is set. If L1 is less than L2, the fill bit (F) is used as the L2-L1 least significant bits of string 1. The contents of string 1 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero - If all the resultant bits generated are zero, then ON if L2=0 and $L_1 \geq 0$; otherwise, OFF

Truncation - If L1 is > L2, then ON; otherwise, OFF
If L1>0 and L2=0, then ON. If L1=L2=0, then OFF.

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification is used or if an illegal repeat is used.

2. An IPR fault does not occur even when $L_1 = 0$ or $L_2 = 0$. In this case, the zero and truncation indicators are affected.

EXAMPLES:

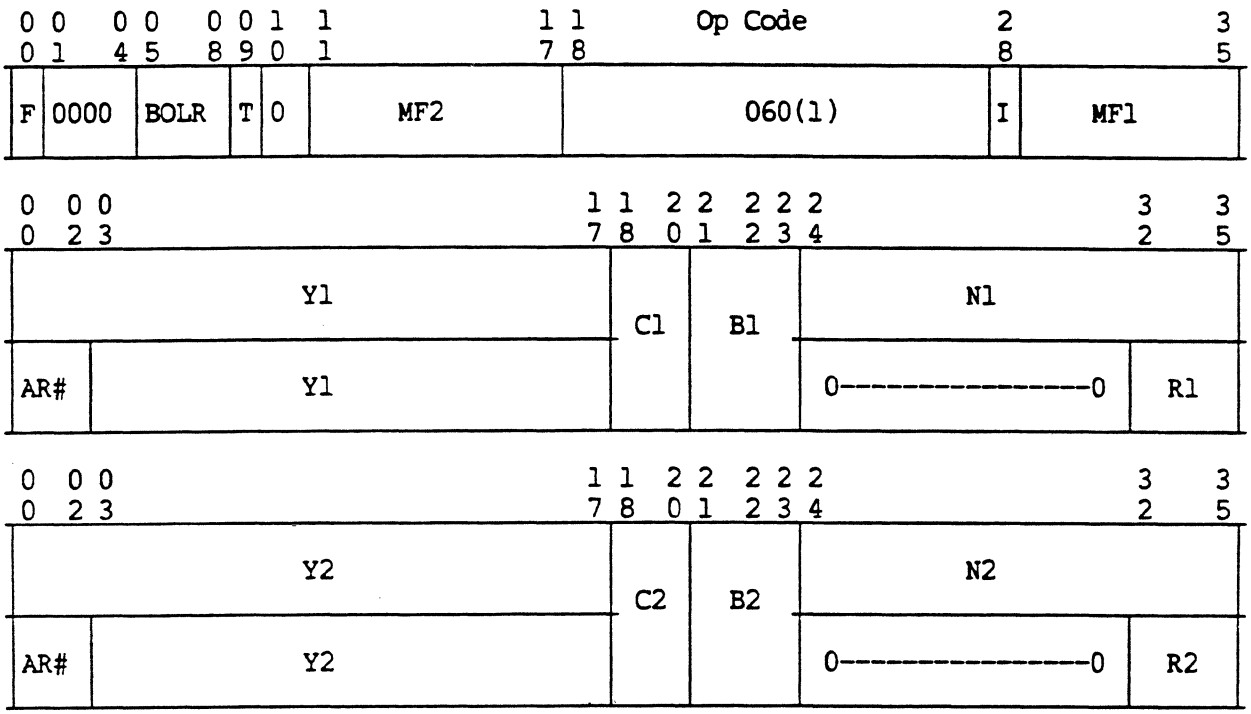
```
1       8       16              32
        REM     BITS 0-17 OF FLD2 FORCED ON
        CSL     ,,07,,1         "ORING" with truncation enable option
        BDSC    FLD1,24,1,3     FLD1 operand descriptor
        BDSC    FLD2,18,0,0     FLD2 operand descriptor
        USE     CONST           memory contents in octal
FLD1    VFD     12/0,18/-1,6/0
        0 0 0 0 7 7 7 7 7 7 0 0
FLD2    LDA     0,2             0 0 0 0 0 0 2 3 5 0 1 2
        USE                     7 7 7 7 7 7 2 3 5 0 1 2  (Result)

        REM     BITS 18-35 OF FLD2 INVERTED
        CSL     ,,06,1 .        exclusive OR with fill bit 1 option
        BDSC    ,0              FLD1 operand descriptor
        BDSC    FLD2,18,2,0     FLD2 operand descriptor
        USE     CONST.          memory contents in octal
FLD2    DEC     0               0 0 0 0 0 0 0 0 0 0 0 0
        USE                     0 0 0 0 0 0 7 7 7 7 7 7  (Result)
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16              32
        EAX6    12              load char and bit address modifier into X6
        EAX7    54              load FLD2 length into X7
        EAX4    FLD2            load FLD2 address into X4
        AWDX    0,4,4           put FLD2 address into AR4
        CSL     (,,1),00,(1,1,,6),00 clear operation with address
        modification ARG        2,4
                pointer to FLD1 indirect operand descriptor
        BDSC    0,X7,,,4        FLD2 operand descriptor (FLD2,54,1,3)
        USE     CONST.          memory contents in octal
FLD2    VFD     36/-1,36/-1     777777777777 777777777777
        BDSC    ,0              FLD1 operand descriptor (control field zeros)
        USE                     777700000000 000000000077  (Result)
```

CSR	Combine Bit Strings Right	061 (1)

FORMAT:



CODING FORMAT: The CSR instruction is coded as follows:

```
1       8       16
_______________________

        CSR     (MF1),(MF2),BOLR,F,T
        BDSC    LOCSYM,N,C,B,AM
        BDSC    LOCSYM,N,C,B,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) : (BOLR) : C(string 2) --> C(string 2)

EXPLANATION: This instruction operates the same as CSL except that the starting locations are YCB1 + (L1-1) and YCB2 + (L2-1) and the evaluation is from right to left (least to most significant bits). Any truncation or fill is of most significant bits.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Same as for CSL

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification is used or if an illegal repeat is used.
2. An IPR fault does not occur even when $L_1 = 0$ or $L_2 = 0$. In this case, the zero and trunctaion indicators are affected.

EXAMPLES:

```
1       8       16              32
        CSR     ,,14,,1         invert with truncation fault enable option
        BDSC    FLD1,18,2,0     FLD1 operand descriptor
        BDSC    FLD2,12,0,0     FLD2 operand descriptor
        USE     CONST.          memory contents in octal
FLD1    OCT     444444          000000444444
FLD2    DEC     0               333300000000  (Result)
        USE                     truncation

        CSR     ,,17            force ones operation
        BDSC    ,0              FLD1 operand descriptor
        BDSC    FLD2,36,0,0     FLD2 operand descriptor
        USE     CONST.          memory contents in octal
FLD2    BSS     1               7 7 7 7 7 7 7 7 7 7 7 7    (Result)
        USE                     none
```

CWL	Compare with Limits	111 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Y) :: closed (algebraic) interval [C(A), C(Q)] and (algebraic comparison) C(Y) :: C(Q)

C(Y), C(A), C(Q) are unchanged

EXPLANATION: This instruction tests the algebraic value of C(Y) to determine if it is within the range of algebraic values bounded by C(A) and C(Q). The indicators are then set to reflect the result. This instruction is not recommended for logical (unsigned) comparisons.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Y) is contained in the closed interval [C(A), C(Q)] i.e., either $C(A) \leq C(Y) \leq C(Q)$ or $C(A) \geq C(Y) \geq C(Q)$, then ON; otherwise, OFF

Negative	Carry	Relationship	Sign
0	0	C(Q) > C(Y)	$C(Q)_0 = 0$, $C(Y)_0 = 1$
0	1	C(Q) ≥ C(Y)	$C(Q)_0 = C(Y)_0$
1	0	C(Q) < C(Y)	$C(Q)_0 = C(Y)_0$
1	1	C(Q) < C(Y)	$C(Q)_0 = 1$, $C(Y)_0 = 0$

DFAD	Double-Precision Floating Add	477 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) + C(Y-pair)] normalized --> C(EAQ);

C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTES:

1. The definition of normalization is located under the description of the FNO instruction.
2. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is CPU mode register, bit 33.
3. An Illegal Procedure fault occurs if illegal address modification is used.

DFCMG	Double-Precision Floating Compare Magnitude	427 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: $|C(E,AQ_{0-63})|$ :: $|C(Y\text{-pair})|$; magnitude comparison

C(EAQ), C(Y-pair) unchanged

EXPLANATION: This comparison is executed as follows:

1. Compare C(E) :: $C(Y)_{0-7}$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.

2. Compare the absolute values of the mantissas and set the indicators accordingly.

The DFCMG instruction is identical to the DFCMP instruction except that the magnitudes of the mantissas are compared instead of the algebraic values.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	Negative	Relationship
0	0	$\|C(E,AQ_{0-63})\| > \|C(Y\text{-pair})\|$
1	0	$\|C(E,AQ_{0-63})\| = \|C(Y\text{-pair})\|$
0	1	$\|C(E,AQ_{0-63})\| < \|C(Y\text{-pair})\|$

NOTES:

1. When indicator bit 32 = 1 and hex permission flag = 1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary. The hex permission flag is CPU mode register, bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

DFCMP	Double-Precision Floating Compare	517 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(E,AQ_{0-63}) :: C(Y-pair); C(EAQ), C(Y-pair) unchanged

EXPLANATION: This comparison is executed as follows:

a. Compare C(E) :: $C(Y)_{0-7}$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.

b. Compare the mantissas and set the indicators accordingly.

The DFCMP instruction is identical to the FCMP instruction except for the precision of the mantissas actually compared.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	Negative	Relationship
0	0	$\|C(E,AQ_{0-63})\| > \|C(Y\text{-pair})\|$
1	0	$\|C(E,AQ_{0-63})\| = \|C(Y\text{-pair})\|$
0	1	$\|C(E,AQ_{0-63})\| < \|C(Y\text{-pair})\|$

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary. The hex permission flag is Mode register, bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

DFDI	Double-Precision Floating Divide Inverted	527 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Y-pair) / C(EAQ) --> C(EAQ); C(Y-pair) unchanged

EXPLANATION: If AQ_{64-71} is not = 0 and A_0 = 0, a 1 is added to AQ_{63}. Zero is moved to AQ_{64-71}, unconditionally. AQ_{0-63} is then used as the divisor mantissa. The 8-bit dividend exponent and 72-bit mantissa are placed in working registers. The dividend mantissa is shifted right, and the dividend exponent is increased accordingly until: |Dividend mantissa| < $|C(AQ)_{0-63}|$. When such a shift occurs, significant bits from the dividend may be lost.

$C(AQ)_{0-63}$ is used as the divisor mantissa. 64 bits of quotient mantissa are placed in AQ_{0-63}. Zeros are placed in AQ_{64-71}.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

	When Division Occurs	When No Division Occurs
Zero	If C(A) = 0, then ON; otherwise, OFF	If divisor mantissa =0, then ON; otherwise, OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise, OFF	If dividend < 0, then ON; otherwise, OFF
Exponent Overflow	If quotient exponent is > +127, then ON	
Exponent Underflow	If exponent of floating point result < - 128, then ON	

DFDI

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is Mode register bit 33.

2. If the divisor mantissa C(AQ) is zero, the division does not take place. Instead, a Divide Check fault occurs and all registers remain unchanged. The dividend and divisor are not normalized by the hardware prior to division.

3. An Illegal Procedure fault occurs if illegal address modification is used.

DFDV	Double-Precision Floating Divide	567 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(EAQ) / C(Y-pair) --> C(EAQ); C(Y-pair) unchanged

EXPLANATION: $C(AQ)_{0-71}$ are used by this instruction. If the divisor mantissa $C(Y\text{-pair})_{8-71}$ is zero, then the division does not take place. Instead, a Divide Check fault occurs. The divisor C(Y) remains unchanged, C(AQ) contains the dividend magnitude in absolute, and the Negative indicator reflects the dividend sign. Dividend and divisor are not normalized by the hardware prior to division.

The dividend mantissa C(AQ) is shifted right and the dividend exponent is increased accordingly until

$$|C(AQ)_{0-63}| < |C(Y\text{-pair})_{8-71}| \text{ with zero fill}|.$$
$$C(E) - C(Y\text{-pair})_{0-7} \quad \text{--> } C(E)$$

When such a shift occurs, significant bits from the dividend may be lost. 64 bits of the quotient mantissa are placed in AQ_{0-63}. Zeros are placed in AQ_{64-71}.

When the divisor mantissa is 0, division is not executed and a Divide Check fault occurs. The absolute value of the dividend is loaded into AQ, and the Negative indicator is set in accordance with the sign of the dividend.

Refer to the FDV instruction for details of the method of shifting the dividend.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

	When Division Occurs	When No Division Occurs
Zero	If C(A) = 0, then ON; otherwise, OFF	If divisor mantissa =0, then ON; otherwise, OFF
Negative	If $C(AQ)_0$ = 1, then ON; otherwise, OFF	If dividend < 0, then ON; otherwise, OFF
Exponent Overflow	If quotient exponent is > +127, then ON	
Exponent Underflow	If exponent of floating point result < - 128, then ON	

NOTES:

1. When indicator bit 32=1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

DFLD	Double-Precision Floating Load	433 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Y-pair), 00...0 --> C(EAQ); C(Y-pair) unchanged

$C(Y)_{0-7}$ --> C(E)

$C(Y\text{-pair})_{8-71}$ --> $C(AQ)_{0-63}$

00...0 --> $C(AQ)_{64-71}$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

DFLP	Double-Precision Floating Load Positive	532 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: |C(Y-pair)|, normalized --> Z

Z_{0-7} --> C(E)

Z_{8-71} --> $C(AQ)_{0-63}$

00...0 --> $C(AQ)_{64-71}$

EXPLANATION: The memory operand C(Y) is processed as double-precision floating-point data. The absolute value of this data is normalized and its exponent, mantissa (bits 8-71), and 0 are loaded into C(E), $C(AQ)_{0-63}$, and $C(AQ)_{64-71}$, respectively.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent > +127, then ON.

Exponent Underflow - If exponent of floating point result < - 128, then ON

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

DFMP	Double-Precision Floating Multiply	463 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) * C(Y-pair)] normalized --> C(EAQ); C(Y-pair) unchanged

EXPLANATION: This multiplication is executed as follows:

$C(E) + C(Y\text{-pair})_{0-7} \rightarrow C(E)$.

$C(AQ) * C(Y\text{-pair})_{8-71}$ results in a 134-bit product plus sign. This sign plus the leading 71 bits are loaded into the AQ. C(EAQ) normalized --> C(EAQ).

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	-	If C(AQ) = 0, then ON; otherwise, OFF
Negative	-	If $C(AQ)_0$ = 1, then ON; otherwise, OFF
Exponent Overflow	-	If exponent > +127, then ON
Exponent Underflow	-	If exponent of floating point result < - 128, then ON

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

DFRD	Double-Precision Floating Round	473 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(EAQ) rounded to 64 bits and normalized --> C(EAQ)

EXPLANATION: A true round is performed on C(EAQ) to reduce the mantissa of the floating-point number to 64 bits. The exponent is set to -128 if the rounded mantissa = 0.

This instruction is identical with FRD except that the rounding constant is added to bits 65-71 and the results are rounded to 64 bits of precision. Bits 64-71 of C(AQ) are replaced by zeros.

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

DFSB	Double-Precision Floating Subtract	577 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) - C(Y-pair)] normalized --> C(EAQ); C(Y-pair) unchanged

EXPLANATION: The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

DFSBI	Double-Precision Floating Subtract Inverted	467 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(Y-pair) - C(EAQ)] normalized --> C(EAQ);
C(Y-pair) unchanged

EXPLANATION: The two's complement of the subtrahend is first taken and the smaller value is then right shifted to equalize it. The shifted portion is truncated and the addition is executed. After addition, the sum is normalized and the 72 bits of the mantissa are loaded into AQ.

The order of execution of the operation conforms to that of the DFSB instruction. Normalization is defined under FNO.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

DFST	Double-Precision Floating Store	457 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(E) --> $C(Y\text{-pair})_{0-7}$

$C(AQ)_{0-63}$ --> $C(Y\text{-pair})_{8-71}$

C(EAQ) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

DFSTR	Double-Precision Floating Store Rounded	472 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: $C(EAQ)_{0-71}$ rounded, normalized --> C(Y-pair);

C(EAQ) unchanged

EXPLANATION: This instruction performs a true round on C(EAQ) to 64 bits of precision in C(AQ). The result is normalized and stored in the Y-pair. C(EAQ) is unchanged. The exponent is stored as -128 if the rounded mantissa = 0. (See the FRD instruction for the definition of true round.)

Except for precision, this instruction is identical with the FSTR instruction.

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS:

Zero	-	If C(Y-pair) = floating-point zero, then ON; otherwise, OFF
Negative	-	If $C(Y\text{-pair})_8$ = 1, then ON; otherwise, OFF
Exponent Overflow	-	If exponent > +127, then ON
Exponent Underflow	-	If exponent of floating point result < - 128, then ON

DFSTR

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

DIS	Delay Until Interrupt Signal	616 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Executes in NS mode only with Privileged Master mode

SUMMARY: No operation takes place other if enabled, PATROL is invoked. The processor does not continue with the next instruction, but waits for a program interrupt signal. When an interrupt occurs, PATROL is stopped.

ILLEGAL ADDRESS MODIFICATIONS: None. Modification is performed, including modification of any indirect words specified. However, the effective address has no effect on the operation, including the final value of the instruction counter.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The inhibit bit in this instruction only affects the recognition of a Timer Runout (TRO) fault as follows:

   - o Inhibit ON delays the recognition of a TRO until the processor enters Slave mode.
   - o Inhibit OFF allows the TRO to interrupt the DIS state

   For all other faults and interrupts, the inhibit bit is ignored.

2. A Command fault occurs if execution is attempted in Slave or Master mode.

3. An IPR fault occurs if this instruction is used in the ES mode.

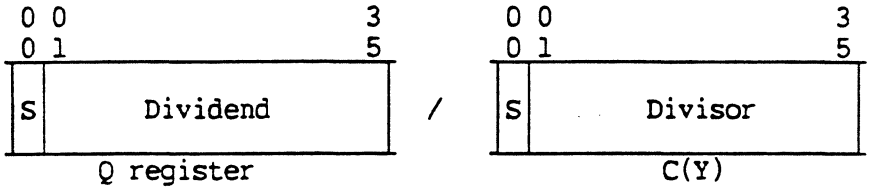
DIV	Divide Integer	506 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

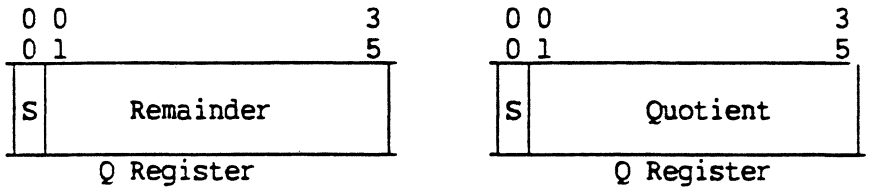
OPERATING MODES: Any

SUMMARY:
C(Q) / C(Y)
integral quotient --> C(Q), right-adjusted
integral remainder --> C(A), right-adjusted
C(Y) unchanged

EXPLANATION: C(Q) and C(Y) are considered as 36-bit integers (including sign). The integer quotient of C(Q) divided by C(Y) is loaded into the Q register and the integer remainder is loaded into the A register. The remainder sign is the same as that of the dividend unless the remainder is zero.



yielding:



ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

	If division takes place	If no division takes place
Zero	If C(Q) = 0, ON; otherwise, OFF	If divisor = 0, ON; otherwise, OFF
Negative	If bit 0 of C(Q) = 1,ON; otherwise, OFF	If dividend < 0, ON; otherwise, OFF

NOTE: If the dividend = -2**35 and the divisor = +/-1, or if the divisor is 0 under any condition, division does not take place. Instead, a Divide Check fault occurs, C(Y) remains unchanged, C(Q) contains the dividend magnitude, and the Negative indicator reflects the dividend sign, and C(A) is set to zero.

DRL	Derail Fault	002 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: DRL generates a Derail fault, which causes the processor to switch to Privileged Master mode and execute an Inward CLIMB instruction using the entry descriptor obtained from the word pair in memory locations 32 and 33 octal.

If the safestore bypass flag in the option register = 1, a safestore frame is generated. The size of this safestore frame is determined by the type of the entry descriptor. The occurrence of the DRL fault is indicated in the safestore frame by a code of 00110 in bits 12-16 of word 5.

The wired-in CLIMB instruction functions as though the second word of the CLIMB instruction had the following characteristics:

E = 0	No parameters
C_{18}	Do not load X0
C_{19}	No effect. Turn Master Mode indicator ON.
C_{22-23}	= 0 Inward CLIMB
S, D	No effect

The entry descriptor specifies a descriptor to be obtained from the linkage segment for loading into the instruction segment register (ISR). The entry descriptor also specifies the value to be loaded into the instruction counter (ID).

The processor is placed in Privileged Master mode for the execution of the wired-in CLIMB. Upon completion of the CLIMB, the processor remains in Privileged Master mode if flag bit 26 of the new ISR = 1 (privileged); otherwise, the processor changes to Master Mode.

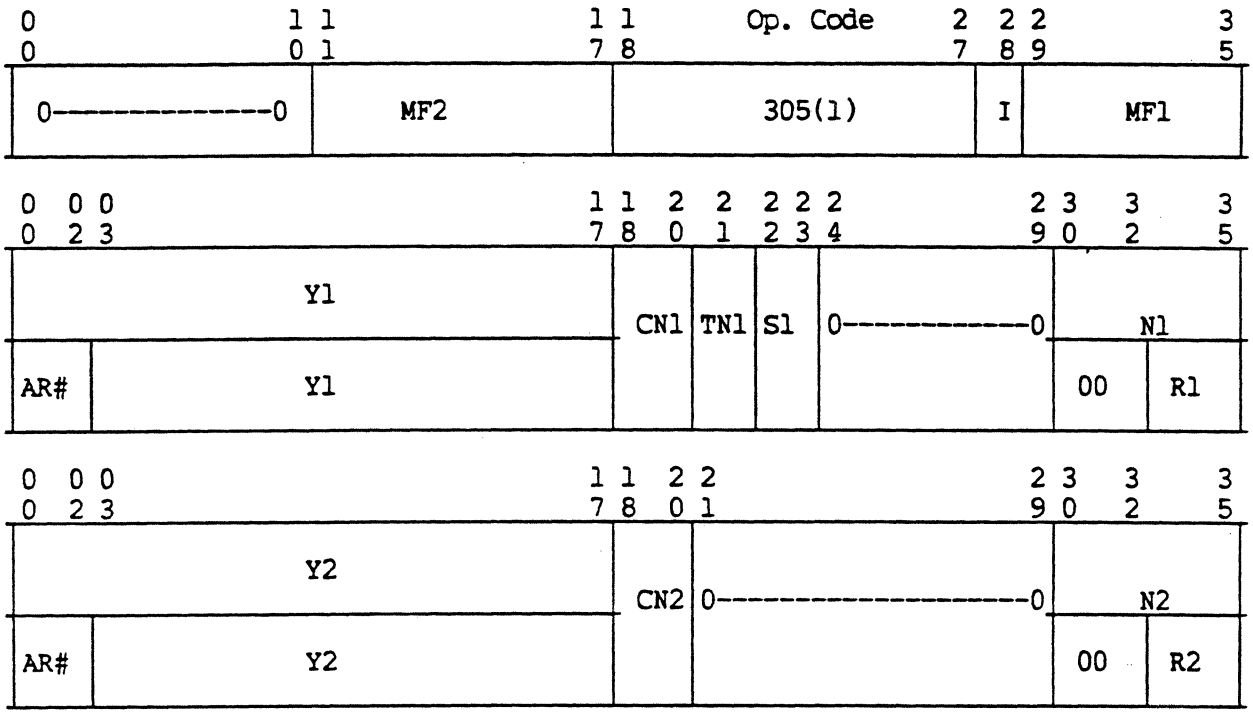
ILLEGAL ADDRESS MODIFICATIONS: Not executed

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Master Mode - ON

DTB	Decimal-to-Binary Convert	305 (1)

FORMAT:



CODING FORMAT: The DTB instruction is coded as follows:

```
1       8       16
_____________________

        DTB     (MF1),(MF2)
        NDSCn   LOCSYM,CN,N,S,,AM
        NDSC9   LOCSYM,CN,N,,,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY:

converted
C(string 1) -------> C(string 2)

The string of decimal characters of data type TN1, sign and decimal type S1 (S1 = 00 is illegal), and scale factor 0 that starts at YC1 is converted into a two's complement binary integer and stored, right-justified, as a character string of length L2, starting at location YC2. If the string generated is longer than L2, the high-order excess is truncated and the overflow indicator is set. CN2 is given in the 9-bit character format with legal codes of 000, 010, 100, and 110.

If string 1 contains more than 32, when the generated binary string is longer than L_2, the upper bits are truncated and the overflow indicator is set.

CN_2 specifies the value for the 9-bit character format, the correct codes being 000, 010, 100, or 110. L2 specifies the length of the stored binary value in 9-bit units, and must be equal to or less than 8. The length of the stored binary value is 9, 18, 27, 36, 45, 54, 63, or 72 bits.

Provided that string 1 and string 2 are not overlapped, the contents of string 1 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero - If all the resultant bits generated are zero, then ON; otherwise, OFF

Negative - If the resultant sign is negative, then ON; otherwise, OFF

Overflow - If L2 is less than the number of 9-bit segments generated, then ON; otherwise, unchanged

NOTES:

1. An Illegal Procedure fault occurs for the following reasons:

   - o If DU or DL modifications are used for MF1 or MF2
   - o If L2 is less than 1 or > 8
   - o If CN2 does not contain a legal code
   - o If S1 = 00
   - o If illegal digit or sign is detected in string 1
   - o If N1 is not large enough to specify the number of characters required for the specified sign and/or exponent, plus at least one digit

2. An IPR fault occurs if illegal address modification is specified or if an illegal repeat is used.

3. If string 1 has the value -2**(9*L2-1), the result is zero and the overflow indicator is turned ON.

4. If string 1 contains more than 22 significant digits, an incorrect result is produced and the overflow indicator is turned ON.

5. If the binary result is longer than L2 9-bit characters, the most significant nontruncated bit is forced to agree with the result sign.

EXAMPLES:

```
1       8       16                  32
        DTB
        NDSC4   FLD1,3,5,2          decimal operand descriptor
        NDSC9   FLD2,0,4            binary operand descriptor
        USE     CONST.              memory contents in octal
FLD1    EDEC    8P1234-             0 0 0 0 0 1 0 4 3 1 1 5
FLD2    BSS     1                   7 7 7 7 7 7 7 7 5 4 5 6  (Result)
        USE                         any indicators set?   negative

        DTB
        NDSC9   FLD1,0,22,3         decimal operand descriptor
        NDSC9   FLD2,0,8            binary operand descriptor
        USE     CONST.              memory contents
FLD1    EDEC    22A2361183241434822606847 (maximum decimal value)
FLD2    BSS     2                   377777777777777777777777 (Result)
        USE                         any indicators set?   none

        DTB
        NDSC4   FLD1,3,3,3          decimal operand descriptor
        NDSC9   FLD2,2,2            binary operand descriptor
        USE     CONST.              memory contents in octal
FDL1    EDEC    8P51200             0 0 0 0 0 5 0 2 2 0 0 0
FLD2    DEC     -1                  7 7 7 7 7 7 0 0 1 0 0 0
        USE                         any indicators set?   none

        DTB
        NDSC9   FLD1,0,4,3          decimal operand descriptor
        NDSC9   FLD2,3,1            binary operand descriptor
        USE     CONST.              memory contents in octal
FLD1    EDEC    4A1023              0 6 1 0 6 0 0 6 2 0 6 3
FLD2    DEC     0                   0 0 0 0 0 0 0 0 0 7 7 7
        USE                         any indicators set?   overflow
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16              32
        EAX0    0               load FLD character modifier into X0
        EAX2    2               load FLD2 length into X4
        EAX7    FLD2            load FLD2 address modifier into X7
        AWDX    0,7,4           put FLD2 address modifier into AR4
        DTB     (,,1),(1,1,,0)  with modification
        ARG     1,,4            pointer to FLD1 indirect descriptor
        NDSC9   0,,X2,,,4       binary FLD2 descriptor (FLD2,0,2)
        TZE     *+3             zeros was the result
        TMI     *+2             negative result
        TOV     *+1             high-order bit truncated
        USE     CONST.          memory contents in octal
FLD1    EDEC    4PL-512         3 2 5 0 2 2 0 0 0 0 0 0
FLD2    OCT     111111          7 7 7 0 0 0 1 1 1 1 1 1
        NDSC4   FLD1,0,4,1      decimal operand descriptor
        USE                     any indicators set?   negative
```

DUFA	Double-Precision Unnormalized Floating Add	437 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) + C(Y-pair)] not normalized --> C(EAQ)
C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary. The hex permission flag is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

DUFM	Double-Precision Unnormalized Floating Multiply	423 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) * C(Y-pair)] not normalized --> C(EAQ)
C(Y-pair) unchanged

EXPLANATION: This multiplication is executed like the DFMP instruction, except that the final normalization is performed only if both factor mantissas are = -1.00...0.

Except for the precision of the mantissa of the operand from main memory, the DUFM instruction is identical to the UFM instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow- If exponent of floating point result < - 128, then ON

NOTES:

1. When indicator bit 32 = 1 the the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

DUFS	Double-Precision Unnormalized Floating Subtract	537 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) - C(Y-pair)] not normalized --> C(EAQ)
C(Y-pair) unchanged

EXPLANATION: The two's complement of the subtrahend is first taken and the smaller value is then right-shifted to equalize it. The portion shifted out is truncated and addition is executed.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

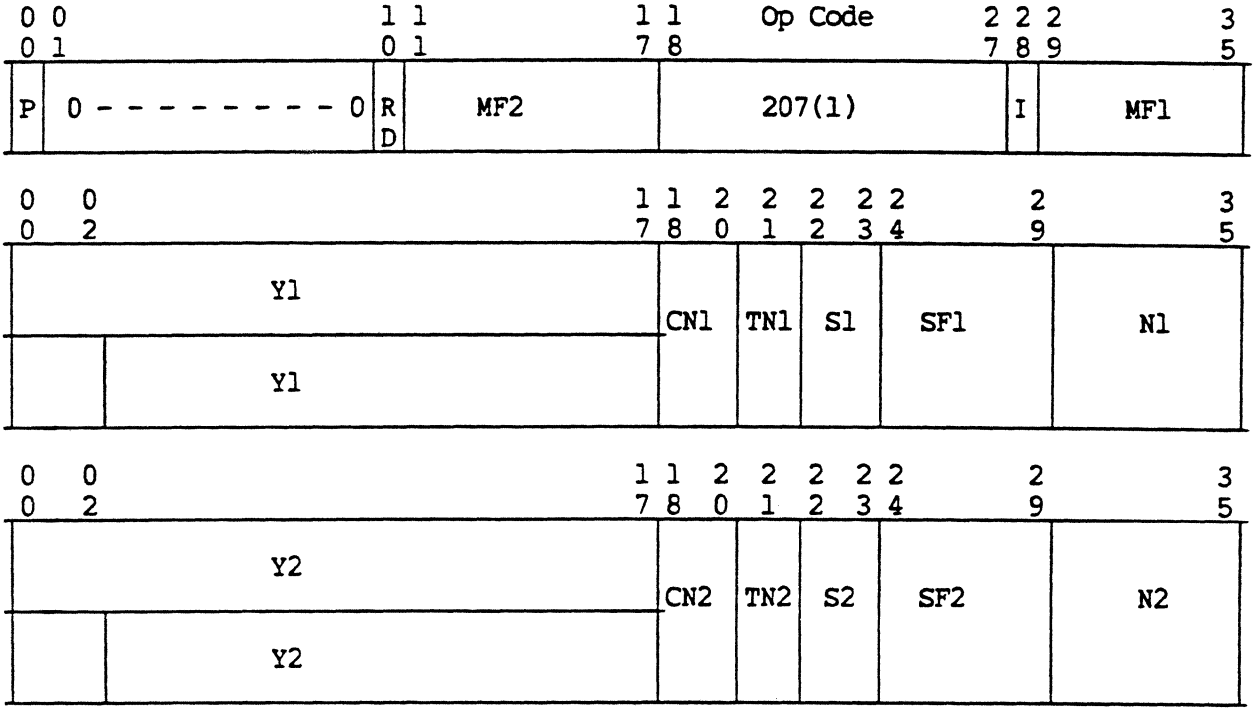
NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary. The hex permission flag is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

FORMAT:

DV2D	Divide Using Two Decimal Operands	207 (1)



CODING FORMAT: The DV2D instruction is coded as follows:

```
1       8       16
-------------------------------------

        DV2D    (MF1),(MF2),RD,P
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) / C(string 1) --> C(string 2)

EXPLANATION: Same as for DV3D except that the quotient is stored using YC2, TN2, S2 and, if S2 indicates a scaled format, SF2.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Same as for DV3D

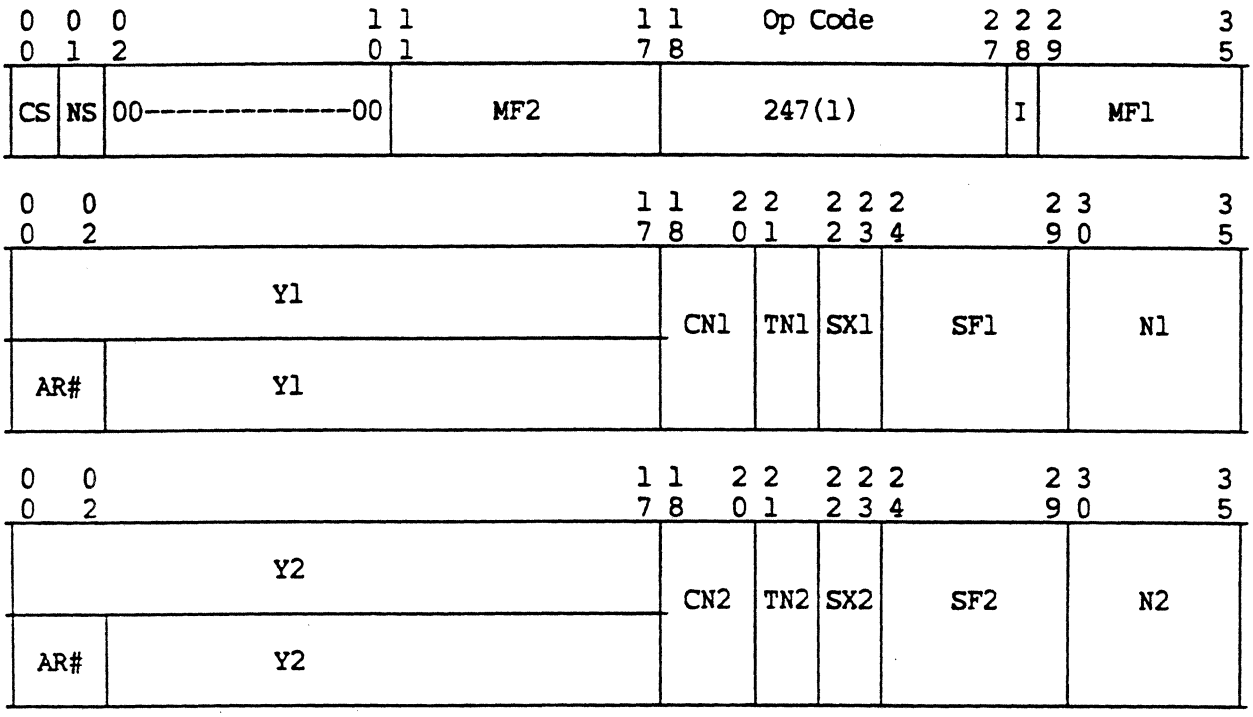
NOTE: The notes of DV3D apply.

EXAMPLES:

```
1       8       16              32
        DV2D
        NDSC4   FLD1,4,4,2,-4   divisor operand descriptor
        NDSC4   FLD2,0,8,0      dividend operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    8P2+             0002+
FLD2    EDEC    8P+8642E0       +08642 +0
        USE                     +43210 +3   (Quotient)

        DV2D    ,,1             with rounding option
        NDSC9   FLD1,0,4,1,-3   divisor operand descriptor
        NDSC4   FLD2,0,8,1,-2   dividend operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    4A+5            + 005
FLD2    EDEC    8P+1234         +0001234
        USE                     +0246800     (Quotient)
 *                              indicators on?   none
```

DV2DX	Divide Using Two Decimal Operands Extended	247 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
DV2DX   (MF1),(MF2),RD,CS,NS
NDSCn   LOCSYM,CN,N,SX,SF,AM
NDSCn   LOCSYM,CN,N,SX,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) / C(string 1) --> C(string 2)

EXPLANATION: Same as for DV3DX except that the quotient is stored using YC2, TN2, SX2 and, if SX2 indicates a scaled format, SF2.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or MF2

ILLEGAL REPEATS: RPT, RPD, RPL

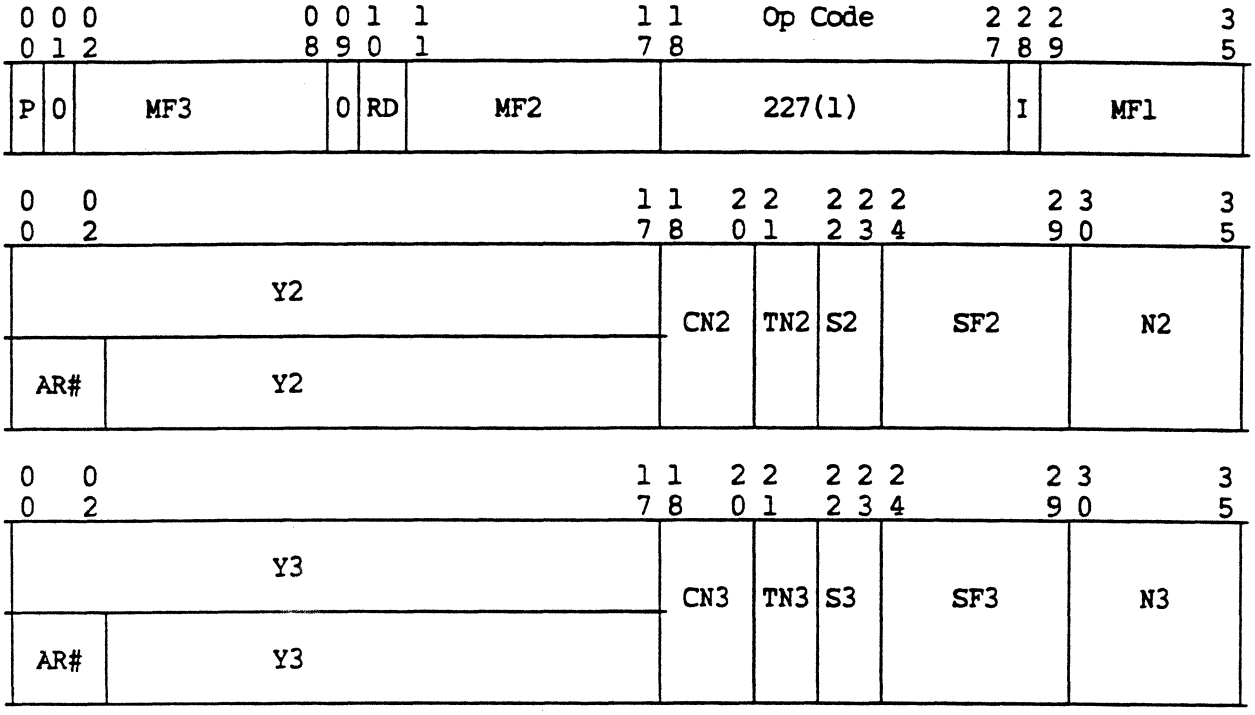
INDICATORS: Same as for DV3D

NOTES:

1. Notes of DV3D apply.
2. See MVNX for information about coding of overpunched signs.

DV3D	Divide Using Three Decimal Operands	227 (1)

FORMAT:



CODING FORMAT: The DV3D instruction is coded as follows:

```
1       8       16
_______________________

        DV3D    (MF1),(MF2),(MF3),RD,P
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) / C(string 1) --> C(string 3)

EXPLANATION: The decimal number of data type TN1, sign and decimal type S1, and starting location YC1, is divided into the decimal number of data type TN2, sign and decimal type S2, and starting location YC2. The quotient is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type S3.

If S3 indicates a fixed-point format, the quotient is stored using scale factor SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most-significant-digit overflow or least-significant-digit truncation to occur.

If S3 indicates a floating-point format, the quotient is right-justified to preserve the most significant nonzero digits; this may cause least-significant-digit truncation.

If P=1, positive signed 4-bit results are stored using octal 13 as the plus sign. If P=0, positive signed 4-bit results are stored with octal 14 as the plus sign.

If RD is a 1, the quotient is rounded prior to storage.

Provided that strings 1, 2, and 3 are not overlapped, the contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

The divide operation stops when the number of required digits have been formed or, in the case where rounding is specified (RD = 1), when the required number of quotient digits plus 1 have been formed. In fixed-point operations or floating-point operations where the quotient is stored in fixed-point format, the required number of quotient digits is determined as follows:

When the quotient descriptor specifies that the quotient is to be stored in fixed-point format, the necessary number of quotient digits to form is calculated as follows:

#QD = (LD-#LZD+1)-(LDR-#LZR)+(ED-EDR-EQ)

DV3D

where:

#LZD = number of leading zeros in dividend

#QD = number of quotient digits to form

LD = length of dividend

LDR = length of divisor

#LZR = number of leading zeros in divisor

ED = exponent of dividend

EDR = exponent of divisor

EQ = scale factor for quotient

The hardware performs this calculation prior to beginning the divide operation and, if #QD > 63, the divide operation does not take place; a Divide Check fault occurs. If #QD<=0, then zero is stored.

In a floating-point divide operation, the required number of quotient digits is determined as follows. With the divisor greater than the dividend, a leading zero is generated in the quotient. The leading zero counts as one of the generated output digits. For example, if 4-digit output accuracy is specified and the above relationship exists between the divisor and the dividend, only 3-digit accuracy will be attained. Under this condition, it would be necessary to specify a 5-digit output to achieve 4-digit accuracy.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS: RPT, RPD, RPL

DV3D

INDICATORS:

Zero	-	If result equals zero, then ON; otherwise, OFF
Negative	-	If result is negative, then ON; otherwise, OFF
Exponent Overflow	-	If exponent of floating-point result is > 127, then ON
Exponent Underflow	-	If exponent of floating point result < - 128, then ON
Overflow	-	If fixed-point integer overflow, then ON; otherwise, unchanged
Truncation	-	If the least significant digits are truncated without rounding, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if:
   - o DU or DL modification is specified for MF1 or MF2.
   - o Any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.
   - o The values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.
2. A Divide Check fault occurs under either of the following two conditions.
   - o If the divisor equals zero. The divisor is the number starting at YC1.
   - o If S3 specifies that the quotient be stored in scaled format and the calculated length required for the quotient is greater than 63 (refer to length requirements above).
3. If an illegal digit or sign is detected, the receive field is not changed before the IPR fault occurs.

EXAMPLE:

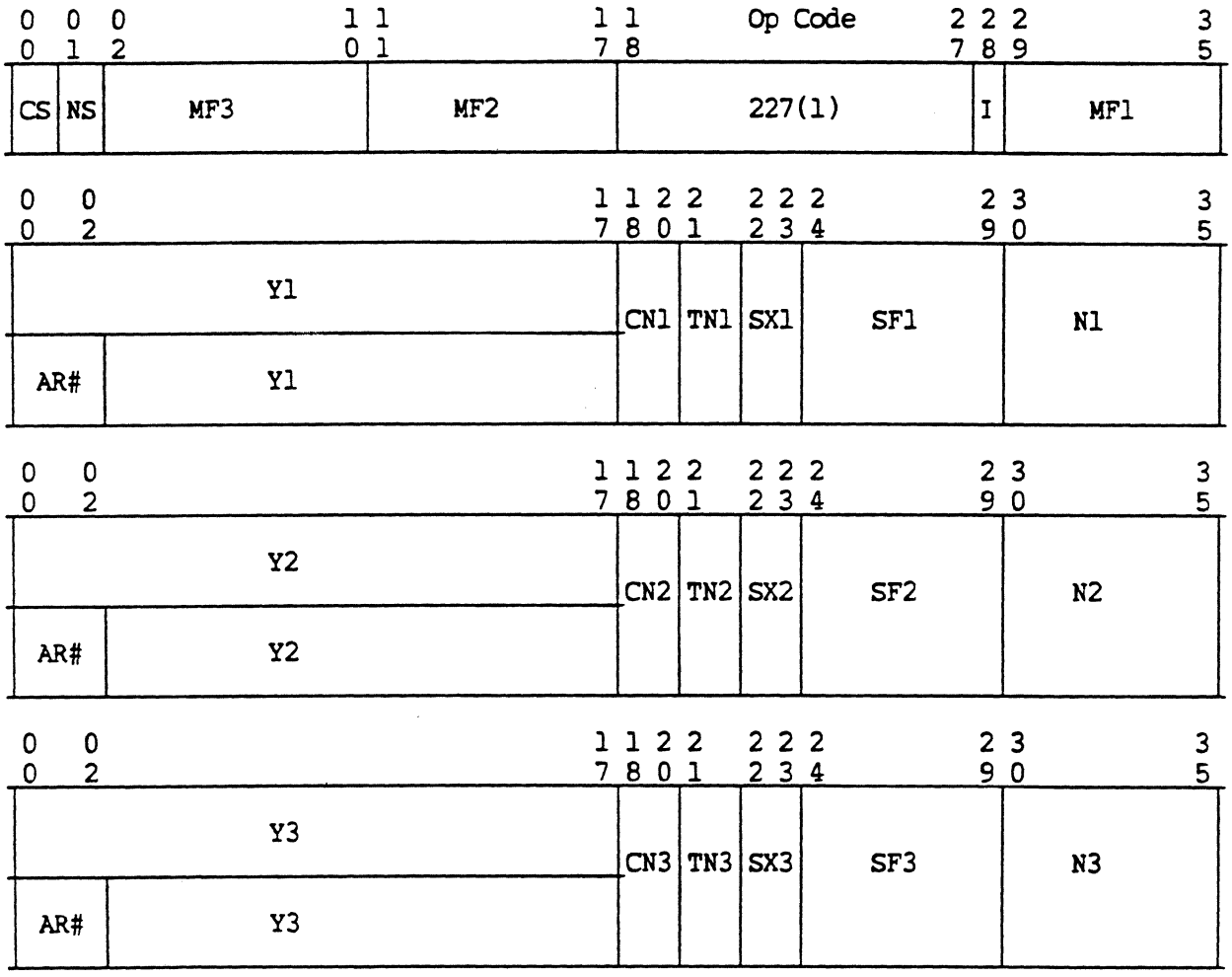
```
          1         8         16                32
                    DV3D      ,,,1,1            with rounding and plus sign options
                    NDSC9     FLD1,1,3,2,-2     divisor operand descriptor
                    NDSC4     FLD2,0,9,0        dividend operand descriptor
                    NDSC4     FLD3,2,6,1,-1     quotient operand descriptor
                    USE       CONST.            memory contents
          FLD1      EDEC      4A2-              002-
          FLD2      EDEC      9P-876543E-3      -876543-3
          FLD3      BSS       1                 xx+38272   (Quotient)
                    USE                         instruction fault?   overflow
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
          1         8         16                32
                    EAX2      2                 load character modifier into X2
                    EAX7      8                 load FLD2 length into X7
                    EAX4      FLD1              load FLD1 address into X4
                    AWDX      0,4,4             put FLD1 address into AR4
                    DV3D      (1,,,2),(,1),(,,1),1,1  with address modification
options
                    NDSC9     0,0,2,3,-2,4      divisor operand descriptor
(FLD1,2,2,3,-2)
                    NDSC9     FLD2,0,X7,0       dividend operand descriptor
(FLD2,0,8,0)
                    ARG       2,2,4             pointer to quotient operand descriptor

                    USE       CONST.            memory contents
          FLD1      EDEC      4A2               0002
          FLD2      EDEC      8A+876543E-3      +876543-3
          FLD3      BSS       1                 x+438272
                    NDSC4     FLD3,1,7,1,-1     quotient operand descriptor
                    USE                         instruction fault?     none
```

DV3DX	Divide Using Three Decimal Operands Extended	267 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
DV3DX  (MF1),(MF2),(MF3),RD,CS,NS
NDSCn  LOCSYM,CN,N,SX,SF,AM
NDSCn  LOCSYM,CN,N,SX,SF,AM
NDSCn  LOCSYM,CN,N,SX,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) / C(string 1) --> C(string 3)

EXPLANATION: The decimal number of data type TN1, sign and decimal type SX1, and starting location YC1, is divided into the decimal number of data type TN2, sign and decimal type SX2, and starting location YC2. The quotient is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type SX3.

If SX3 indicates a fixed-point format, the quotient is stored using scale factor SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied, most-significant-digit overflow, or least-significant-digit truncation.

If SX3 indicates a floating-point format, the quotient is right-justified to preserve the most-significant nonzero digits; this may cause least-significant-digit truncation.

The character set is defined by CS (EBCDIC/ASCII). Placement of overpunched sign in the output is controlled by NS. (Refer to the introductory pages of this section for definition of the NS field.) If RD is 1, the quotient is rounded prior to storage. The contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, or MF3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Same as for DV3D.

NOTES:

1. Explanation of the divide operation in the DV3D description apply.

2. A divide check fault occurs under either of the following two conditions:

   - o If the divisor (the number starting at YC1) equals zero.

   - o If SX3 specifies that the quotient be stored in fixed-point format and the calculated length required for the quotient is greater than 63 (see Note 2 of DV3D).

3. Refer to specifications on MVNX for information about coding of overpunched signs.

4. IPR fault conditions are the same as for DV3D.

DVF	Divide Fraction	507 (0)

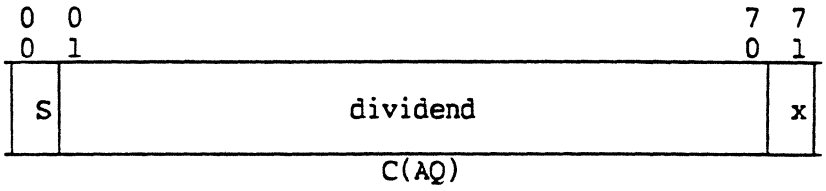
FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

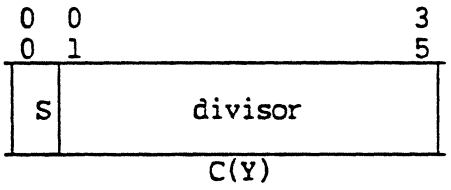
SUMMARY: C(AQ) / C(Y)

fractional quotient --> C(A), left-adjusted
fractional remainder --> C(Q), left-adjusted
C(Y) unchanged

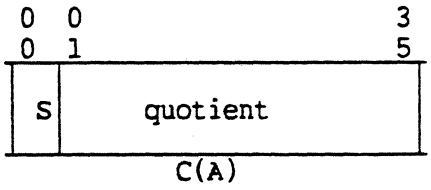
EXPLANATION: This instruction divides a 71-bit fractional dividend (including sign) by a 36-bit fractional divisor (including sign) to form a 36-bit fractional quotient (including sign) and a 36-bit fractional remainder (including sign). Bit 35 of the remainder corresponds to bit 70 of the dividend. The remainder sign is equal to the dividend sign unless the remainder is zero. Bit 71 of C(AQ) is not used.

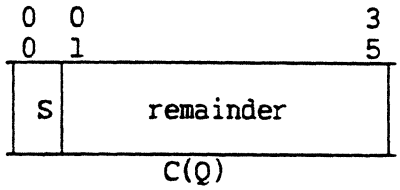


/



yielding:





If |dividend| >= |divisor| or if the divisor = 0, division does not take place. Instead, a Divide Check fault occurs, C(Y) remains unchanged, C(AQ) contains the dividend magnitude as an absolute value, and the negative indicator reflects the dividend sign.

ILLEGAL ADDRESS MODIFICATIONS: None

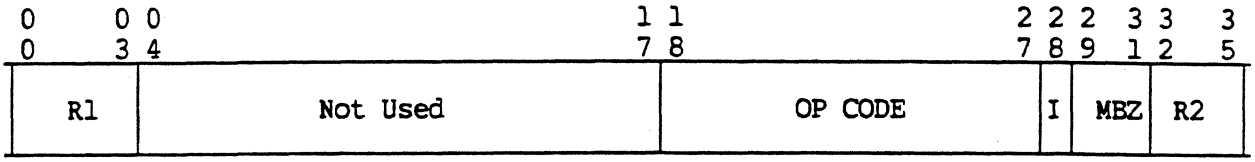
ILLEGAL REPEATS: None

INDICATORS:

	If division takes place:	If no division takes place:
Zero	If C(A) = 0, then ON; otherwise, OFF	If divisor = 0, then ON; otherwise, OFF
Negative	If $C(A)_0$ = 1, then ON; otherwise, OFF	If dividend < 0, then ON; otherwise, OFF

DVRR	Divide Register by Register	533 (1)

FORMAT:



CODING FORMAT:

```
1       8       16
        DVRR    R1,,R2
```

OPERATING MODES: Executes in ES mode only

SUMMARY: When "register pair" is implied

R1, R2 = 0, 2, 4, 6, AQ

otherwise

R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

Quotient of C(R1-odd) / C(R2) --> C(R1-odd)

Remainder of C(R1-odd) / C(R2) --> C(R1-even)

C(R2) unchanged

EXPLANATION: A register pair is specified in R1. The content of the odd-numbered register, or Q if AQ is specified, is divided by C(R2). The resulting quotient is loaded into R1-odd and the remainder into R1-even.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS:

	If division takes place:	If no division takes place:
Zero	If C(R1-odd) = 0, then ON; otherwise, OFF	If divisor = 0, then ON; otherwise, OFF
Negative	If $C(R1\text{-odd})_0$ = 1, then ON; otherwise, OFF	If dividend < 0, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.

2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

3. Both the dividend and divisor are regarded as a 36-bit signed integer. The sign of the remainder is the same as that of the dividend unless the remainder is 0.

4. A Divide Check fault occurs in the following cases:

   - Dividend = -2^{35} and divisor + -1
   - Divisor = 0

   In these cases, the instruction is not executed. C(R2) remains unchanged, C(R1-odd) takes the absolute value of the dividend, and C(R1-even) is 0. If the dividend is -2^{35}, then -2^{35} is loaded into R1-odd.

EAA	Effective Address to A-Register	635 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS mode

$Y \rightarrow C(A)_{0-17}$

$0...0 \rightarrow C(A)_{18-35}$; C(Y) unchanged

ES mode

$00 \rightarrow C(A)_{0-1}$

$Y_{0-33} \rightarrow C(A)_{2-35}$; C(Y) unchanged

EXPLANATION: This instruction permits inter-register data movement. The data source is specified by the address modification and the data destination by the operation code of the instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. In the ES mode, the negative indicator is always set to OFF.

EAQ	Effective Address to Q-Register	636 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

Y --> $C(Q)_{0-17}$;

00...0 --> $C(Q)_{18-35}$; C(Y) unchanged

ES Mode

00...0 --> $C(Q)_{0-1}$

Y_{0-33} -->$C(Q)_{2-35}$)

EXPLANATION: This instruction permits inter-register data movement. The data source is specified by the address modification and the data destination by the operation code of the instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. In the ES mode, the negative indicator is always set to OFF.

EAXn	Effective Address to Index Register n	62n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,...,7 as determined by opcode

Y_{0-33} --> (Xn); C(Y) unchanged

ES Mode

For n = 0,1,...,7 as determined by opcode

00 --> $C(GXn)_{0-1}$

Y_{0-33} --> $C(GXn)_{2-35}$

EXPLANATION: This instruction permits inter-register data movement. The data source is specified by the address modification and the data destination by the operation code of the instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL

ILLEGAL REPEATS: RPT, RPD, or RPL of EAX0

INDICATORS: Zero - If C(Xn/GXn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn/GXn)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. In the ES mode, the negative indicator is always set to OFF.

EPAT	Effective Pointer and Address to Test	412 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: This instruction tests the virtual address to real memory address mapping function of the hardware. Addresses are generated in the normal sequence and stored in four special test registers instead of accessing memory.

Real memory address$_{1-27}$ --> C(Test Reg 0)$_{0-26}$

Effective WSN --> C(Test Reg 0)$_{27-35}$

Relative Virtual address --> C(Test Reg 1)$_{0-35}$

C(DR)effective --> C(Test Reg 2,3)

The high-order real address bit is not placed in the test register.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Illegal address modifications and illegal repeats cause an IPR fault.
2. This instruction is only intended for use with Test and Diagnosis (T&D) programs.

EPPRn	Effective Pointer to Pointer Register n	63n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: This set of eight instructions generates an effective address (EA) and loads it into the pointer register (ARn, SEGIDn, DRn).

NS Mode

If instruction bit 29 = 0 then

SEGID(IS)	--> SEGIDn
C(ISR)	--> C(DRn)

If instruction bit 29 = 1 and indirection is not used in forming EA, then

Effective address (EA)	-->$C(AR)_{0-23}$
Effective SEGID	-->C(SEGIDn)
Effective DR	-->C(DRn)

If instruction bit 29 = 1 and indirection is used in forming EA, then

EA_{0-17}	-->$C(ARn)_{0-17}$
0...0	-->$C(ARn)_{18-23}$
SEGIDm	-->SEGIDn
C(DRm)	-->DRn

ES Mode

If instruction bit 29 = 0, then

EA_{4-33}	-->$C(AR)_{0-29}$
EA_{34-39}	-->C(AR)30-35
Effective SEGID	-->C(SEGIDn)
Effective DR	-->C(DRn)

If instruction bit 29 = 1 and indirection is not used in forming EA, then

EA_{4-39}	-->$C(AR)_{0-35}$
SEGIDm	-->SEGIDn
C(DRm)	-->C(DRn)

If instruction bit 29 = 1 and indirection is used in forming EA, then

EA_{4-33}	-->C(ARn)0-29
$EA_{34-39}=0$	-->$C(ARn_{30-35}$
SEGIDm	-->SEGIDn
C(DRm)	-->C(DRn)

EXPLANATION: If the instruction bit 29 = 0, AR is not used for generation of the effective address and the ARn byte and bit portions are set to zero.

When the instruction bit 29 = 0, the generated operand address is in the instruction segment. The ISR and SEGID(IS) content are loaded into DRn and SEGIDn, respectively.

If the instruction bit 29 = 1, the Address Register ARm specified with bits 0, 1, and 2 of the instruction word are used to generate the effective address. Provided that indirect modification is not specified, the ARn byte and bit portions are preserved during computation of the effective address and loaded into the byte and bit portions of the corresponding ARn. If indirect modification is specified, zero is loaded into the ARn byte and bit portions.

ILLEGAL ADDRESS
MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An IPR fault occurs if illegal address modification or illegal repeats are used.

EXAMPLE:

```
1       8       16              32
        ADQ     =3HOBI,DC       file codefile
        ORQ     =0400000,DL     read permissions
        EPPR0   ALCPRF          allocate file command block
        PPME    ALPRMF,2        allocate file

ALEPRF  VEC     .ISR,NAME,NAMEX,(R,W,S)
        VEC     .ISR,CBUFF,CBUFFX,(R,W,S)
          .
NAME    BCI     4
          .
          .
          .
NAMEX   EQU     *-NAME
CBUFF   BSS     355
CBUFFX  EQU     *-CBUFF
```

ERA	EXCLUSIVE OR to A-Register	675 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35, $C(A)_i$ XOR $C(Y)_i$ --> $C(A)_i$; C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0$ = 1, then ON; otherwise, OFF

ERAQ	EXCLUSIVE OR to AQ-Register	677 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 71, $C(AQ)_i$ XOR $C(Y\text{-pair})_i$ --> $C(AQ)_i$; C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

ERQ	EXCLUSIVE OR to Q-Register	676 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35, $C(Q)_i$ XOR $C(Y)_i$ --> $C(Q)_i$;
C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

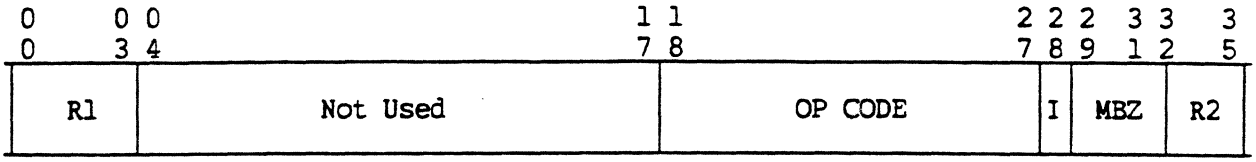
ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0$ = 1, then ON; otherwise, OFF

ERRR	EXCLUSIVE OR Register to Register	537 (1)

FORMAT:



CODING FORMAT:

```
1       8       16
        . ERRR  R1,,R2
```

OPERATING MODES: Executes in ES mode only.

SUMMARY: R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

$C(R1)_i$ XOR $C(R2)_i$ --> $C(R1)_I$ i = 0, 1, 2,...,35

C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

ERSA	EXCLUSIVE OR to Storage with A-Register	655 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35, $C(A)_i$ XOR $C(Y)_i$ --> $C(Y)_i$;
C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

NOTES:
1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. See Examples under ERA.

ERSQ	EXCLUSIVE OR to Storage with Q-Register	656 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35, $C(Q)_i$ XOR $C(Y)_i$ --> $C(Y)_i$;
C(Q) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLE:

```
1       8       16
        LDQ     =1,DL
        ERSQ    FLAG

 * If bit 35 of FLAG is ON, then set to zero
```

ERSXn	EXCLUSIVE OR to Storage with Index Register n	64n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,...,7 as determined by op code

For i = 0 to 17, $C(Xn)_i$ XOR $C(Y)_i$ --> $C(Y)_i$;

C(Xn) and $C(Y)_{18-35}$ unchanged

ES Mode

For n = 0,1,...,7 as determined by op code

For i = 0 to 35, $C(GXn)_i$ XOR $C(Y)_i$ --> $C(Y)_i$;

C(GXn) is unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, or RPL of ERSX0

INDICATORS: NS Mode

Zero - If $C(Y)_{0-17}$ = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

ES Mode

Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ERXn	EXCLUSIVE OR to Index Register n	66n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,...,or 7 as determined by op code

For i = 0 to 17, $C(Xn)_i$ XOR $C(Y)_i$ --> $C(Xn)_i$;

C(Y) unchanged

ES Mode

For n = 0,1,...,or 7 as determined by op code

For i = 0 to 35, $C(GXn)_i$ XOR $C(Y)_i$ --> $C(GXn)_i$;

C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of ERX0

INDICATORS: NS Mode

Zero - If C(Xn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)_0$ = 1, then ON; otherwise, OFF

ES Mode

Zero - If C(GXn) = 0, then ON; otherwise, OFF

Negative - If $C(GXn)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged illegal but executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

FAD	Floating Add	475 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) + C(Y)] normalized --> C(EAQ); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is Mode register bit 33.

2. See the FNO instruction for a definition of normalization.

3. An Illegal Procedure fault occurs if illegal address modification is used.

FCMG	Floating Compare Magnitude	425 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: $|C(E,AQ_{0-27})|$ :: $|C(Y)|$; magnitude comparison;

C(EAQ), C(Y) unchanged.

EXPLANATION: This comparison is executed as follows:

1. Compare C(E) :: $C(Y)_{0-7}$, select the number with the lower exponent, and shift its mantissa right by the number of places (binary or hex) determined by the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.

2. Compare the absolute values of the mantissas and set the indicators accordingly.

The FCMG instruction is identical to the FCMP instruction except that the magnitudes of the mantissas are compared instead of the algebraic values.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	Negative	Relationship
0	0	$\|C(E,AQ_{0-27}\| > \|C(Y)\|$
1	0	$\|C(E,AQ_{0-27}\| = \|C(Y)\|$
0	1	$\|C(E,AQ_{0-27}\| < \|C(Y)\|$

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary. The hex permission flag is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

FCMP	Floating Compare	515 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(E,AQ_{0-27}):: C(Y); algebraic comparison

EXPLANATION: This comparison is executed as follows:

1. Compare C(E) :: $C(Y)_{0-7}$, select the number with the lower exponent, and shift its mantissa right by the number of places (binary or hex) determined by the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.

2. Compare the mantissas and set the indicators accordingly.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	Negative	Relationship
0	0	C(E,AQ_{0-27} > C(Y)
1	0	C(E,AQ_{0-27} = C(Y)
0	1	C(E,AQ_{0-27} < C(Y)

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary. The hex permission flag is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

FDI	Floating Divide Inverted	525 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Y) / C(EAQ) --> C(EA); 00...0 --> C(Q);
contents of C(Y) unchanged

EXPLANATION: The dividend mantissa is shifted right and the dividend exponent is increased accordingly until:

$$|\text{Dividend mantissa}| < |C(AQ_{0-27})|$$

When such a shift occurs, only zeros from the dividend will be lost.

$C(AQ)_{0-27}$ is used as the divisor mantissa.

36 bits of quotient mantissa are placed in A.

If AQ_{28-71} is not equal to 0 and $A_0 = 0$, then 1 is added to AQ_{27}. 0 --> AQ_{28-71} unconditionally. AQ_{0-27} is then used as the divisor mantissa. The 8-bit dividend exponent and 72-bit mantissa are placed in working registers.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

	If division occurs:	If no division occurs:
Zero -	If C(A) = 0, then ON; otherwise, OFF	If divisor mantissa = 0, then ON; otherwise, OFF
Negative -	If $C(A)_0 = 1$, then ON; otherwise, OFF	If dividend < 0, then ON; otherwise, OFF
Exponent Overflow -	If exponent is > +127, then ON	
Exponent Underflow-	If exponent of floating point result < - 128, then ON	

FDI

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is Mode register bit 33.

2. If the divisor mantissa C(AQ) is zero, division does not take place. Instead, a Divide Check fault occurs and all registers remain unchanged. Dividend and divisor are not normalized by the hardware prior to division.

3. An Illegal Procedure fault occurs if illegal address modification is used.

FDV	Floating Divide	565 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(EAQ) / C(Y) --> C(EA); 00...0 --> C(Q); C(Y) unchanged

EXPLANATION: This division is executed as follows:

The dividend mantissa C(AQ) is shifted right and the dividend exponent C(E) is increased accordingly until

$|C(AQ)_{0-27}| < |C(Y)_{8-35}$ with zero fill$|$

When such a shift occurs, significant bits from the dividend may be lost.

Dividend and divisor are not normalized by the hardware prior to division.

36 bits of quotient mantissa are placed in A.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

	If division occurs:	If no division occurs:
Zero -	If C(A) = 0, then ON; otherwise, OFF	If divisor mantissa = 0, then ON; otherwise, OFF
Negative -	If $C(A)_0$ = 1, then ON; otherwise, OFF	If dividend < 0, then ON; otherwise, OFF
Exponent Overflow -	If exponent is > +127, then ON	
Exponent Underflow-	If exponent of floating point result < - 128, then ON	

FDV

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is Mode register bit 33.

2. If the divisor mantissa (bits 8-35 of C(Y)) is zero, division does not take place. Instead, a Divide Check fault occurs. The divisor C(Y) remains unchanged, C(AQ) contains the dividend's magnitude as an absolute value, and the negative indicator reflects the dividend's sign.

3. An Illegal Procedure fault occurs if illegal address modification is used.

FLD	Floating Load	431 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: $C(Y)_{0-7}$ --> C(E)

$C(Y)_{8-35}$ --> $C(AQ)_{0-27}$

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

FLP	Floating Load Positive	530 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: |C(Y)|, normalized --> Z

Z_{0-7} --> C(E)

Z_{8-35} --> $C(AQ)_{0-27}$

00...0 --> $C(AQ)_{28-71}$

EXPLANATION: The memory operand C(Y) is processed as single-precision floating-point data. The absolute value of this data is normalized and its exponent, mantissa (bits 8-35) and 0 are loaded into C(E), $C(AQ)_{0-27}$ and $C(AQ)_{28-71}$, respectively.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent > +127, then ON

Exponent Underflow If exponent of floating point result < - 128, then ON

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

FMP	Floating Multiply	461 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) * C(Y)] normalized --> C(EAQ); C(Y) unchanged

EXPLANATION: This multiplication is executed as follows:

$C(E) + C(Y)_{0-7}$ --> C(E)

$C(AQ) * C(Y)_{8-35}$ results in a 98-bit product plus sign, the leading 71 bits plus sign of which --> C(AQ).

C(EAQ) normalized --> C(EAQ).

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

FNEG	Floating Negate	513 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: -C (EAQ) normalized --> C(EAQ)

EXPLANATION: This instruction changes the number in C(EAQ) to its normalized negative (if C(AQ) ≠ 0). The operation is executed by first forming the two's complement of C(AQ), and then normalizing C(EAQ).

Even if C(EAQ) is already normalized, an exponent overflow can still occur, namely when C(E) = +127 and C(AQ) = -100...0 (the two's complement representation for the decimal value -1.0).

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flag is Mode register bit 33.

2. An Illegal Procedure fault occurs if an illegal repeat is used.

FNO	Floating Normalize	573 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(EAQ) normalized --> C(EAQ)

EXPLANATION: The instruction normalizes the number in C(EAQ). If the overflow indicator is ON, the number in EAQ is normalized one place to the right; the sign bit 0 of C(AQ) is then inverted to reconstitute the actual sign. The Overflow indicator is set OFF.

A normalized floating binary number is defined as one whose mantissa lies in the interval (0.5, 1.0) such that

$$0.5 \leq |C(AQ)| < 1.0$$

which, in turn, requires that $C(AQ)_0 \neq C(AQ)_1$

A normalized floating hexadecimal number is defined as one whose mantissa lies in the interval (0.0625,1.0) such that

$$0.0625 \leq |C(AQ)| < 1.0$$

which, in turn, requires that

if $C(AQ)_0 = 0$, then $C(AQ)_{1-4} \neq 0000$, and
if $C(AQ)_0 = 1$, then $C(AQ)_{1-4} \neq 1111$

Normalization is performed by shifting $C(AQ)_{1-71}$ to the left (one place if binary, four places if hex) and reducing C(E) by 1, repeatedly, until the conditions for $C(AQ)_0$ and $C(AQ)_1$ or $C(AQ)_{1-4}$ are met. Bits shifted out of AQ_1 are lost.

If C(AQ)=0, then C(E) is set to -128 and the zero indicator is set ON.

This instruction can be used to correct overflows that occur with fixed-point numbers:

```
1       8       16
        TOV     1,IC
        LDAQ    M
        ADAQ    N
        LDE     =71B25,DU
        FNO
```

will normalize C(M-pair) + C(N-pair) correctly, whether or not the addition caused an overflow (assuming overflow masked or successful recovery from Overflow fault).

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero	-	If C(AQ) = 0, then ON; otherwise, OFF
Negative	-	If $C(AQ)_0$ = 1, then ON; otherwise, OFF
Exponent Overflow	-	If exponent is > +127, then ON
Exponent Underflow	-	If exponent of floating point result < - 128, then ON
Overflow	-	Set OFF

NOTE: When indicator bit 32 = 1 and the hex permission flat = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flat is Mode register bit 33.

FRD	Floating Round	471 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(EAQ) rounded to 28 mantissa bits and normalized --> C(EAQ)

EXPLANATION: This instruction performs a true round of C(EAQ) to a precision of 28 bits in C(AQ). The result is then normalized and restored to the EAQ registers. A true round means that the same rounding operation applied to a number of the same magnitude and with an opposite sign would result in a sum of the two rounded numbers of exactly zero.

The rounding operation is performed as follows:

a. A constant (all 1s) is added to bits 29-71 of the mantissa.

b. If the number being rounded is positive, a carry is inserted into the least significant bit position of the adder.

c. If the number being rounded is negative, the carry is not inserted.

d. Bits 28-71 of C(AQ) are replaced by zeros.

If the mantissa overflows upon rounding, it is shifted right one place and a corresponding correction is made to the exponent.

If the mantissa does not overflow and is nonzero upon rounding, normalization is performed.

If the resultant mantissa is all zeros, the exponent is forced to -128 and the zero indicator is set.

If the exponent resulting from the operation is greater than +127, the exponent overflow indicator is set.

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPL

INDICATORS:

Zero	-	If C(AQ) = zero, then ON; otherwise, OFF
Negative	-	If $C(AQ)_0$ = 1, then ON; otherwise, OFF
Exponent Overflow	-	If exponent is > +127, then ON

NOTES:

1. When indicator bit 32 = 1 and the hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flat is Mode register bit 33.

2. An Illegal Proceduree fault occurs if an illegal repeat is used.

FSB	Floating Subtract	575 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) - C(Y)] normalized --> C(EAQ); C(Y) unchanged

EXPLANATION: The two's complement of the subtrahend is first taken and the smaller value is then right-shifted to equalize it. The shifted portion is truncated and the addition is executed. The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32 = 1 and the hex permission flat = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flat is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification is used.

FSBI	Floating Subtract Inverted	465 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(Y) - C(EAQ)] normalized --> C(EAQ); C(Y) unchanged

EXPLANATION: The two's complement of the subtrahend is first taken and the smaller value is then right-shifted to equalize it. The shifted portion is truncated and the addition is executed. After addition, the sum is normalized and the 72 bits of the mantissa are loaded into AQ.

The order of execution of the operation conforms to that of the FSB instruction. Normalization is defined under FNO.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent of floating point result < - 128, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

FST	Floating Store	455 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: $C(E) \rightarrow C(Y)_{0-7}$

$C(A)_{0-27} \rightarrow C(Y)_{8-35}$

C(E), C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

FSTR	Floating Store Rounded	470 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(EAQ) rounded and normalized --> C(Y); C(EAQ) unchanged

EXPLANATION: This instruction performs a true round of C(EAQ) to a precision of 28 bits in C(AQ). The result is then normalized and stored in Y. A true round means that the same rounding operation applied to a number of the same magnitude and opposite sign would result in a sum of the two rounded numbers of exactly zero.

Upon completion of the rounding and normalization, the exponent and truncated mantissa are stored as follows:

a. Exponent in bits 0-7 of C(Y)
   Bits 0-27 of mantissa in bits 8-35 of C(Y)

b. If the resultant mantissa bits 0-27 are all zero, the exponent is forced to -128 and the zero indicator is set (floating-point zero).

The rounding and normalization operation of this instruction is identical with FRD.

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(Y) = floating-point zero, then ON; otherwise, OFF

Negative - If $C(Y)_8$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent
Underflow - If exponent of floating point result < - 128, then ON

NOTES:

1. When indicator bit 32 = 1 and hex permission flag = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The hex permission flat is Mode register bit 33.

2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

FSZN	Floating Set Zero and Negative Indicators from Storage	430 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: Test C(Y); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	Negative	Relationship
0	0	Mantissa $C(Y)_{8-35} > 0$
1	0	Mantissa $C(Y)_{8-35} = 0$
0	1	Mantissa $C(Y)_{8-35} < 0$ (bit 8 of C(Y) = 1)

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

FTR	Floating Truncate Fraction	474 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(EAQ) fraction-truncated and normalized --> C(EAQ)

EXPLANATION: This instruction truncates the fraction part of the floating-point data of C(EAQ) to obtain an integer. The result is normalized and stored into C(EAQ). A proper truncation to an integer is such that truncating the fractional parts of two numbers with the same absolute and different sign and adding the results produces 0.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification does not affect instruction operations, but the modification is executed.

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if an illegal repeat is used.

GLDD	Load Double to GXn	32n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        GLDD    n,Y,R,AM
```

OPERATING MODES: Only ES mode.

SUMMARY: C(Y-pair) --> C(GXn-pair)

EXPLANATION: C(Y-pair) is loaded into the GXn-pair specified by bits 24-26 of the op code. The contents of bits 24-26(n) of the op code determines the load destination of the GXn-pair as follows:

n (octal)	GXn-pair
0	GX0, GX1
2	GX2, GX3
4	GX4, GX5
6	GX6, GX7

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: The same GXn used as an address modification register in an RPL.

ILLEGAL EXECUTES: Execution in NS mode.

INDICATORS: Zero - If C(GXn-pair) = 0, then ON; otherwise, OFF

Negative - If $C(GXn\text{-pair})_0 = 1$, then ON; otherwise, OFF

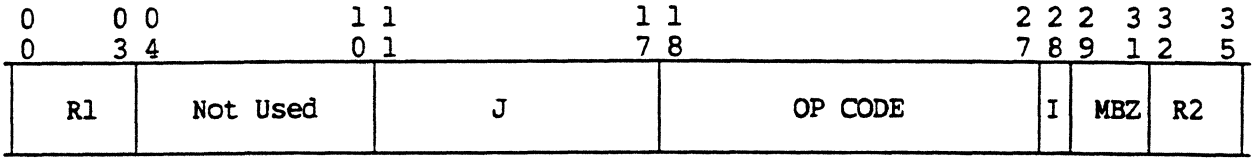
NOTES:

1. An IPR fault occurs if illegal address modifications or repeats are used or if this instruction is executed in the NS mode.

2. An IPR fault occurs if N = 1, 3, 5, or, 7.

GLLS	GXn Long Left Shift	466 (1)

FORMAT:



CODING FORMAT:

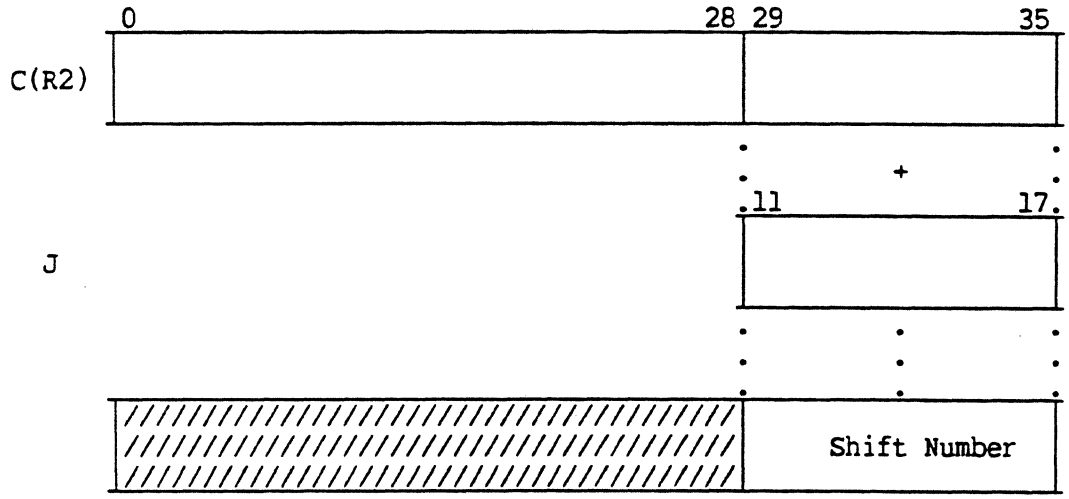
```
1       8       16
        GLLS    R1,J,R2
```

OPERATING MODES: Only ES mode

SUMMARY: R1 = 0, 2, 4, 6, AQ

C(R1-pair) is shifted left. Vacated positions in C(R1-pair) are filled with zeros.

EXPLANATION: The number of bits to be shifted is given by the following:



J is added to $C(R2)_{29-35}$ and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS:

Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0 = 1$, then ON; otherwise, OFF

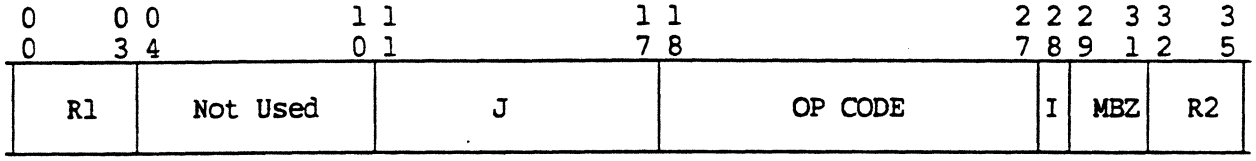
Carry - If a carry out of bit 0 of C(R1) is generated, then ON; otherwise, OFF.

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GLRL	GXn Long Right Logic	465 (1)

FORMAT:



CODING FORMAT:

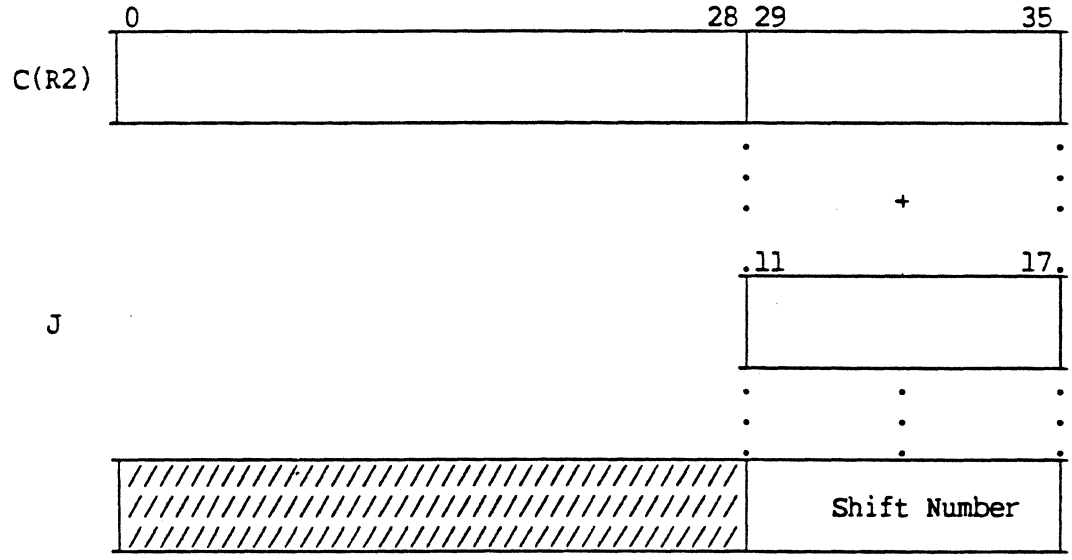
```
1       8       16
        GLRL    R1,J,R2
```

OPERATING MODES: Only ES mode

SUMMARY: R1 = 0, 2, 4, 6, AQ

C(R1-pair) is shifted right. Vacated positions in C(R1-pair) are filled with zeros.

EXPLANATION: The number of bits to be shifted is given by the following:



J is added to $C(R2)_{29-35}$ and the low-order 7 bits of the sum specify the shift number.

GLRL

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1) = 0, then ON; otherwise, OFF

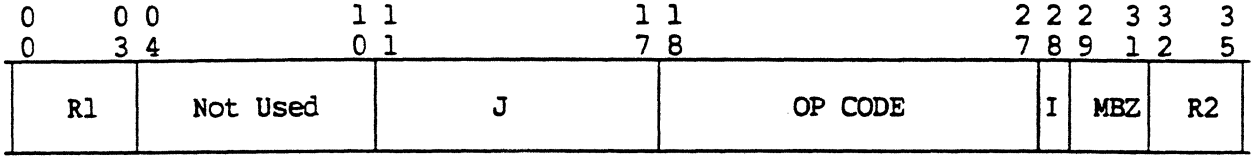
Negative - If $C(R1)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GLRS	GXn Long Right Shift	464 (1)

FORMAT:



CODING FORMAT: 1 8 16

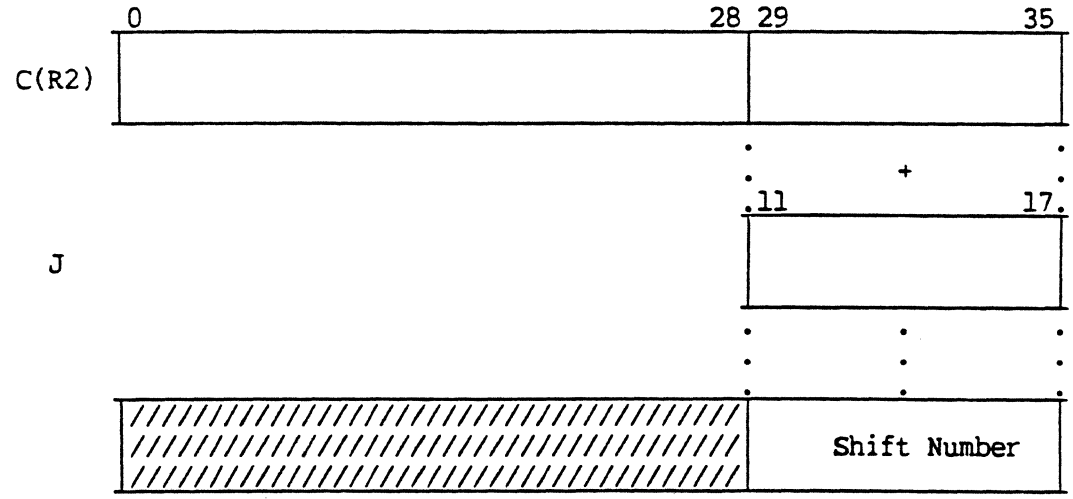
```
        GLRS    R1,J,R2
```

OPERATING MODES: Only ES mode.

SUMMARY: R1 = 0, 2, 4, 6, AQ

C(R1-pair) is shifted right. Vacated positions in C(R1-pair) are filled with bits equal to bit 0 of C(R1-pair).

EXPLANATION: The number of bits to be shifted is given by the following:



J is added to $C(R2)_{29-35}$ and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1) = 0, then ON; otherwise, OFF

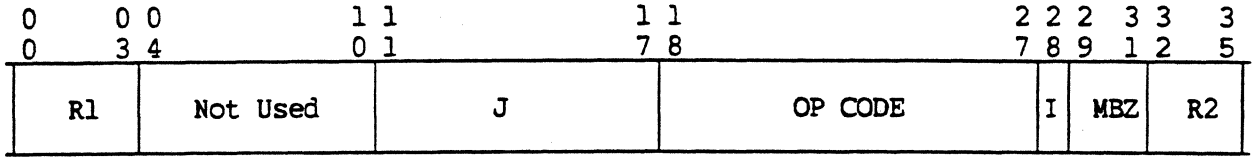
Negative - If $C(R1)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GLS	GXn Left Shift	462 (1)

FORMAT:



CODING FORMAT:

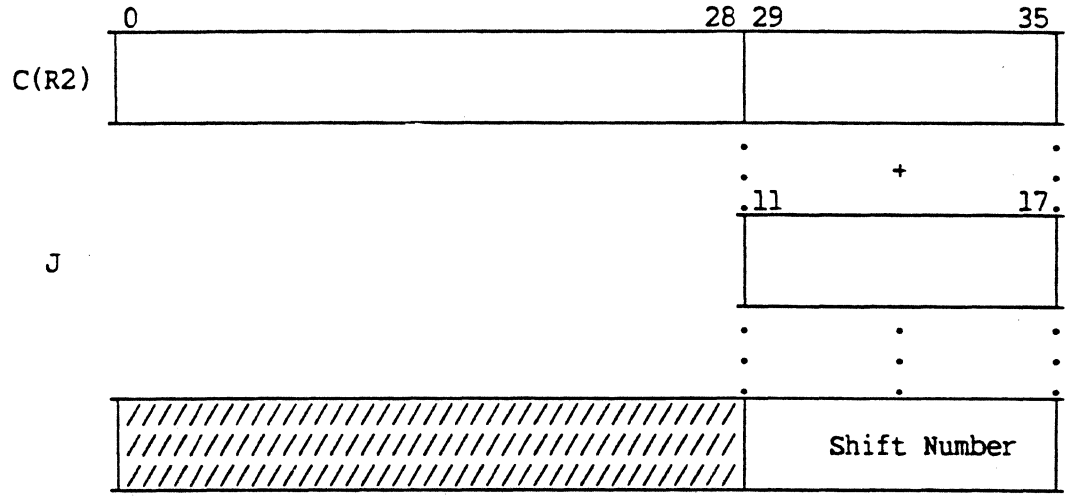
```
1       8       16
        GLS     R1,J,R2
```

OPERATING MODES: Only ES mode.

SUMMARY: R1 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) is shifted left. Vacated positions in C(R1) are filled with zeros.

EXPLANATION: The number of bits to be shifted is given by the following:



J is added to $C(R2)_{29-35}$ and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS:

Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0 = 1$, then ON; otherwise, OFF

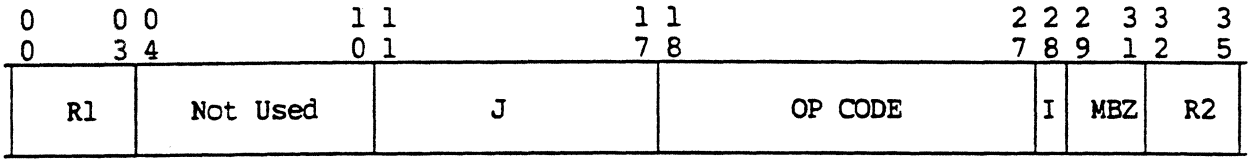
Carry - If a carry out of bit 0 of C(R1) is generated, then ON; otherwise, OFF.

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GRL	GXn Right Logic	461 (1)1)

FORMAT:



CODING FORMAT: 1 8 16

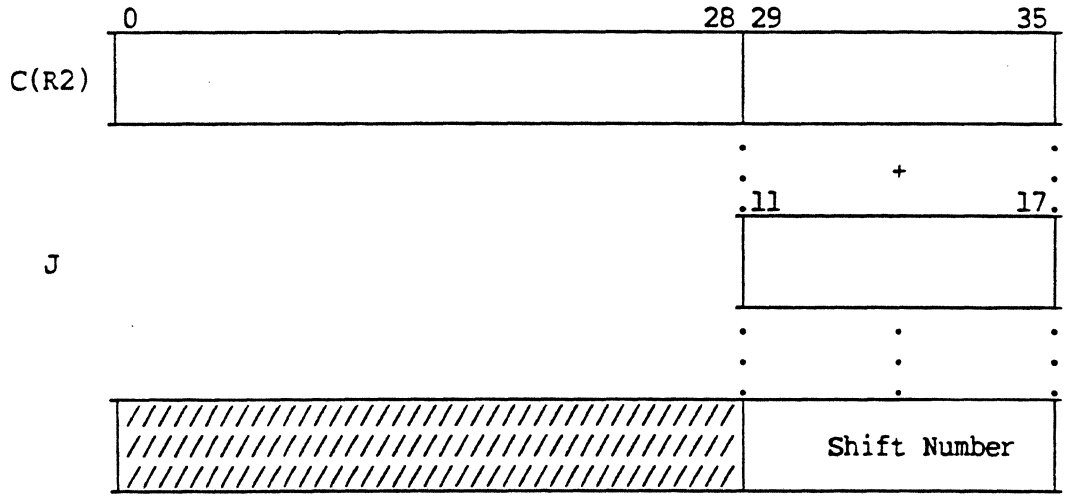
```
        GRL    R1,J,R2
```

OPERATING MODES: Only ES mode.

SUMMARY: R1 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) is shifted right. Vacated positions in C(R1) are filled with zeros.

EXPLANATION: The number of bits to be shifted is given by the following:



J is added to $C(R2)_{29-35}$ and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1) = 0, then ON; otherwise, OFF

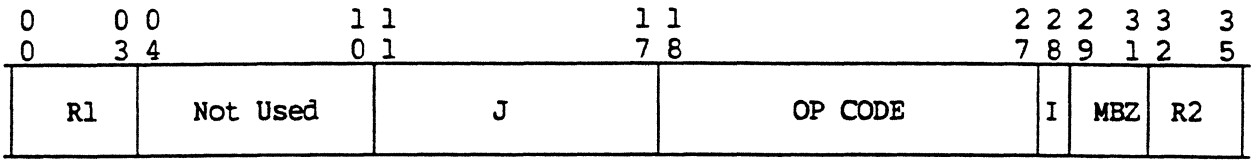
Negative - If $C(R1)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GRS	GXn Right Shift	460 (1)

FORMAT:



CODING FORMAT:

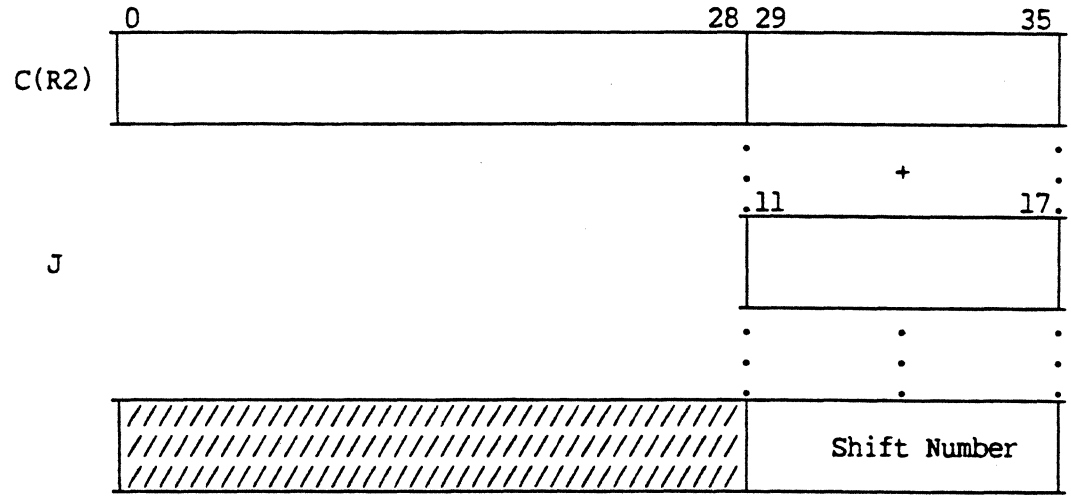
```
1       8       16
        GRS     R1,J,R2
```

OPERATING MODES: Only ES mode.

SUMMARY: R1 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) is shifted right. Vacated positions in C(R1) are filled with bits equal to bit 0 of C(R1).

EXPLANATION: The number of bits to be shifted is given by the following:



J is added to $C(R2)_{29-35}$ and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

GSTD	Store Double from GXn	14n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        GSTD    n,Y,R,AM
```

OPERATING MODES: Only ES mode.

SUMMARY: C(GXn-pair) --> C(Y-pair)

EXPLANATION: The content of the GXn-pair specified by bits 24-26 of the op code is stored in the memory location of Y-pair. The GXn-pair whose contents are to be stored is specified as follows:

n (octal)	GXn-pair
0	GX0, GX1
2	GX2, GX3
4	GX4, GX5
6	GX6, GX7

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

ILLEGAL EXECUTES: Execution in NS mode.

INDICATORS: None affected.

NOTE: An IPR fault occurs if illegal address modifications or repeats are used or if this instruction is executed in the NS mode.

GTB	Gray-to-Binary Convert	774 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: NS mode only

SUMMARY: C(A) is converted from Gray code to a 36-bit binary number

EXPLANATION: This conversion is defined by the following algorithm in which R and S denote the contents of bit position i of the A register before and after the conversion:

$$S_0 = R_0$$

$$S_1 = (R_0 \text{ AND } \overline{S_{i-1}}) \text{ OR } \overline{(R_i \text{ AND } S_{i-1})}$$

where: $i = 1,\ldots,35$.

Gray code is a method of transmitting numeric code cyclically, one bit at a time, to eliminate transmission errors and is defined as follows:

a. A positional binary notation for numbers in which any two sequential numbers whose difference is 1 are represented by expressions that are the same except in one place or column, and in that place or column differ by only one unit.

b. A type of cyclic unit-distance binary code evolved from the 4-word, 2-bit unit distance code (00, 01, 11, 10) according to the following rule:

   To construct an (n+1)-bit reflected binary code from an n-bit reflected binary code, write the n-bit code twice in sequence, first in forward and then in reverse sequence of code words. Prefix an extra bit to each word, assigning the value 0 to the forward version and the value 1 to the backward version of the n-bit code.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if an illegal repeat is used.

LARn	Load Address Register n	76n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        LARn    LOCSYM,RM,AR
```

OPERATING MODES: Any

SUMMARY: NS Mode

For n=0,1,...,7 as determined by op code

$C(Y)_{0-23}$ --> C(ARn); C(Y) unchanged

ES Mode

For n=0,1,...,7 as determined by op code

C(Y) --> C(ARn); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

```
1       8       16              32
        LAR7    ADDR            load bits 0-23 of address into AR7
                .
                .
                .
ADDR    BDSC    512,,8,8        0 0 1 0 0 0 7 0 0 0 0 0 memory contents
*CONTENTS OF AR7 AFTER:         0 0 1 0 0 0 7 0
```

LAREG	Load Address Registers	463 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        LAREG   LOCSYM,R,AR
```

OPERATING MODES: Any

SUMMARY: NS Mode

$C(Y,Y+1,...,Y+7)_{0-23}$ --> C(AR0,AR1,...,AR7)

ES Mode

C(Y,Y+1,...,Y+7) --> C(AR0,AR1,...,AR7)

EXPLANATION: The hardware assumes that the lower 3 bits of address Y = 000 and the 8 words beginning from the 8-word boundary are accessed. No check is performed to determine whether the lower 3 bits of Y = 000. Location Y must be forced to a multiple of 8 by entering an 8 in column 7 of the statement that defines Y, or by using the EIGHT pseudo-operation.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

```
1       8       16                  32
        LAREG   REGW                load AR0...AR7 from REGW...REGW+7
          .
          .
          .
        EIGHT
REGW    DEC     0,0,0,0,0,0,0,0
*
* Result is that all address Registers are
* cleared.
```

LCA	Load Complement into A-Register	335 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: -C(Y) --> C(A); C(Y) unchanged

EXPLANATION: This instruction changes the number to its negative (if ≠ 0) while moving it from Y to A. The operation is executed by forming the two's complement of the string of 36 bits. An overflow condition exists if C(Y) = 2**35.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of A is exceeded, then ON

LCAQ	Load Complement into AQ-Register	337 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: -C(Y-pair) --> (AQ); C(Y-pair) unchanged

EXPLANATION: This instruction changes the number to its negative (if ≠ 0) while moving it from Y-pair to AQ. The operation is executed by forming the two's complement of the string of 72 bits. An overflow condition exists if C(Y)-pair) = -2**71.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of AQ is exceeded, then ON

NOTE: An Illegal Procedure fault occurs if illegal address modifications are used.

LCON	Load Connect Table	016 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(Y, Y+1, Y+2, Y+3 -->C(Connect Table)

C(Y+4, Y+5, Y+5, Y+7) -->(Secondary Connect Table)

EXPLANATION: The connect table is located in the CPU scratch pad memory at locations 74-77. The secondary connect table is at locations 0-3. (Refer to the description of CIOC in this section.)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, and SCR

ILLEGAL REPEATS: RPD, RPL, and RPT

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if this instruction is executed in Slave or Master mode.
2. An IPR fault occurs if illegal address modification or an illegal repeat is used.
3. The SCPR tag 07 instruction stores the connect table.

LCPR	Load Central Processor Register	674 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master Mode

SUMMARY: The operation has several forms depending upon the tag field:

C(Y) -->C(CPU Register)

Operand -->C(CPU Register)

C(A) -->C(PTWAM)

Tag	C(Y) Bits	CPU Register
02	18, 21, 23-25	-->Cache Mode Register
	34-35	-->Lockup Fault Register
04	0-35	-->CPU Mode Register
11	0-17	-->Port Configuration Register
12	5-35	-->Real Address Trap Register
13	33-35	-->CPU Number Register
14	0-35	-->Virtual Address Trap Register

Tag	Operand	CPU Register
03	0-35 = 0...0} 59-99= 0...0}	-->History Registers
07	0-35 = 1...1} 59-99= 1...1}	-->History Registers

(Refer to Section 4 for register format.)

The following tag loads the contents of the PTWAM directory from the A-register. The entry location is specified by the Y address field in the instruction.

Tag	Column	Row	C(A) Bits	Entry
17	Y_{11-16}	Y_{17}	28,29	PTWAM Directory

EXPLANATION: This instruction provides the capability to load the Central Processor registers. The registers are selected by the instruction tag field. The operation has several forms as indicated under summary.

For LCPR Tag 02, cache is flushed when bit 18 is set to the enable state and when a cache mode changes from disable to enable. If an enable condition corresponding to bits 21, 24, and 25 requires a cache flush, software must manipulate bit 18 to cause a cache flush.

For LCPR tag 17, if bit 29 is ON, C(AR) is added to the Y field and the sum forms the entry select. The full virtual address development is not used.

The real and virtual address trap values are also loaded into processor scratch pad at locations 66,67.

ILLEGAL ADDRESS MODIFICATIONS: None. Tag field defines function.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None

NOTES:

1. Attempted execution of LCPR in the Slave or Master mode results in a Command fault.
2. An Illegal Procedure fault occurs if an illegal tag field or an illegal repeat is used.
3. See the SCPR instruction for selecting the central processor registers to be set.

LCQ	Load Complement into Q-Register	336 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: -C(Y) --> C(Q); C(Y) unchanged

EXPLANATION: This instruction changes the number to its negative value (if ≠ 0) while moving it from Y to Q. The operation is executed by forming the two's complement of the string of 36 bits. An overflow condition exists if C(Y) = -2**35.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero	-	If C(Q) = 0, then ON; otherwise, OFF
Negative	-	If $C(Q)_0$ = 1, then ON; otherwise, OFF
Overflow	-	If range of Q is exceeded, then ON

EXAMPLE:

```
1       8       16              32
        LCQ     =5,DL           Loads -5 into the Q-register
```

LCXn	Load Complement into Index Register n	32n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n=0,1....or 7 as determined by opcode

$-C(Y)_{0-17}$ --> (Xn); C(Y) unchanged

ES Mode

For n=0,1....or 7 as determined by opcode

-C(Y) --> (GXn); C(Y) unchanged

EXPLANATION: This instruction changes the number to its negative value (if ≠ 0) while moving it from bits 0-17 of Y to Xn or from Y to GXn. The operation is executed by forming the two's complement of the string of 18 bits.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, or RPL of LCX0

INDICATORS:

Zero - If C(Xn/GXn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn/GXn)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of Xn/GXn is exceeded, then ON

NOTES:

1. In the NS mode, if DL modificatiion is used, the hardware executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification or illegal repeats are used.

LDA	Load A-Register	235 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Y) --> C(A); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

LDAC	Load A-Register and Clear	034 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Y) --> C(A); 0...0 --> C(Y)

EXPLANATION: This instruction is used for a gating operation in multiple CPU systems. Execution of the next instruction is delayed until the cache-flush request applied to all CPUs has completed.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

LDAQ

LDAQ	Load AQ-Register	237 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Y-pair) --> C(AQ); C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

LDAS	Load Argument Stack Register	770 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(Y-pair) --> C(ASR); C(Y-pair) unchanged

EXPLANATION: A descriptor is fetched from even/odd memory locations Y and Y+1 and the following checks are performed on the descriptor:

a. Type field T = 1.

b. Base and bound are modulo 2 words (the three least significant bits of base must be zeros; the three least significant bits of bound must be ones if flag bit 27 is 1).

If these conditions are met, the descriptor is loaded into the argument stack register (ASR) and, in addition, the bound is loaded into the High Water Mark Register (HWMR). During ASR loading, bits 0-6 of the ASR bound field are forced to zero by the processor instead of being loaded from the memory operand. If flag bit 27 of the operand descriptor is zero, the entire bound field is forced to zero, regardless of any value the operand descriptor bound field may contain, and the bound check is bypassed.

(Refer to the description of the PAS instruction for further information concerning the HWMR.)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

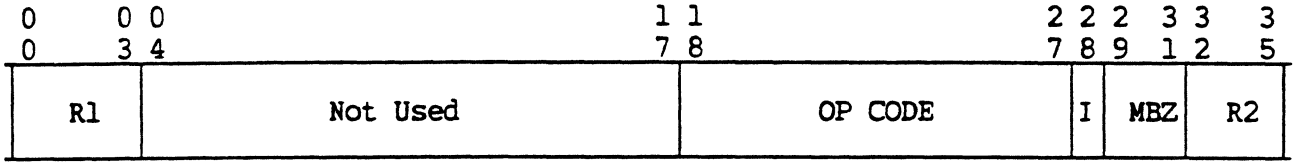
1. Any of the following conditions cause an IPR fault:
   - o Illegal address modifications
   - o Illegal repeats
   - o Segment descriptor type field T is not 1
   - o If the base and bound limits of the operand descriptor are not modulo 2 words.
   - o If flag bit 27 = 1 (bound valid) and the bound is not modulo two words
2. If the processor is in Slave or Master mode, the execution of this instruction causes an Command fault.

EXAMPLE:

```
1       8       16              32
-------------------------------------------------------------

*       ROUTINE TO LOAD REGISTERS - ASR, PSR, DSAR
*       CALLING TSX Z,RDSPRG
        POST    LOST  P0,Z
RDSPRG  EQU     *
        LDP     P0,.SSR,DL      *safe store frame access
        LDP     P0,.CTYP,DL     *change type
        LDDSA   .WDSAR,,P0      *DSAR
        LDAS    .WASR,,P0       *ASR
        LDPS    .WPSR,,P0       *PSR
        TRA     ,Z              *OK
```

LDCR	Load Complement Register from Register	431 (1)

FORMAT:



CODING FORMAT:

```
1       8       16
        LDCR    R1,,R2
```

OPERATING MODES: Executes in ES mode only.

SUMMARY: R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

-C(R2) --> C(R1)

C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0 = 1$, then ON; otherwise, OFF

Overflow - If the range of R1 is exceeded, ON.

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

LDDn	Load Descriptor Register n	67n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: This set of eight instructions provides the capability of loading a descriptor register (DRn) with a new descriptor or modifying the descriptor currently contained in DRn. The segment type referenced by the generated address determines the function to be executed.

In this discussion, DRn represents the specified descriptor, whereas, DRm represents the descriptor register indicated by the y field that is used to load a new segment descriptor.

When the instruction word bit 29 = 1 and the descriptor register specified by bits 0, 1, and 2 of the y field includes a type T = 1 or 3 segment descriptor, the segment descriptor is loaded into the DRn from the segment descriptor segment specified by DRm.

When the instruction word bit 29 = 1 and the type for the segment descriptor in DRm is T = 0, 2, 4, 6, 12, or 14, or when the instruction word bit 29 = 0, a vector operation is performed.

Descriptions of the two types of operations follow. An IPR fault occurs when DRm includes a type T = 7 - 11, 13, or 15 segment descriptor.

Instruction Word Bit 29 = 1; DRm Type T = 1 or 3

The segment descriptor from the segment descriptor segment indicated by DRm is loaded into DRn. When the effective address is generated, only R type modification and DU/DL modification are permitted. The effective address is the offset from the segment descriptor segment indicated by DRm. The segment descriptor from the even/odd location indicated by this address is loaded into DRn and the same checks are performed as for any normal memory reference.

- o A check is made to determine whether a segment is present and whether read is permitted.

o A bound check is made.

The housekeeping bit for that page must be ON because the segment descriptor segment is referenced. If it is OFF, the instruction execution is terminated and a Security Fault, Class 1 occurs. The housekeeping page access for access of the segment descriptor is not dependent upon the CPU mode; it may also be executed in Slave mode.

The ARn and SEGIDn which correspond to the DRn are affected as follows:

o ARn is set to zero.

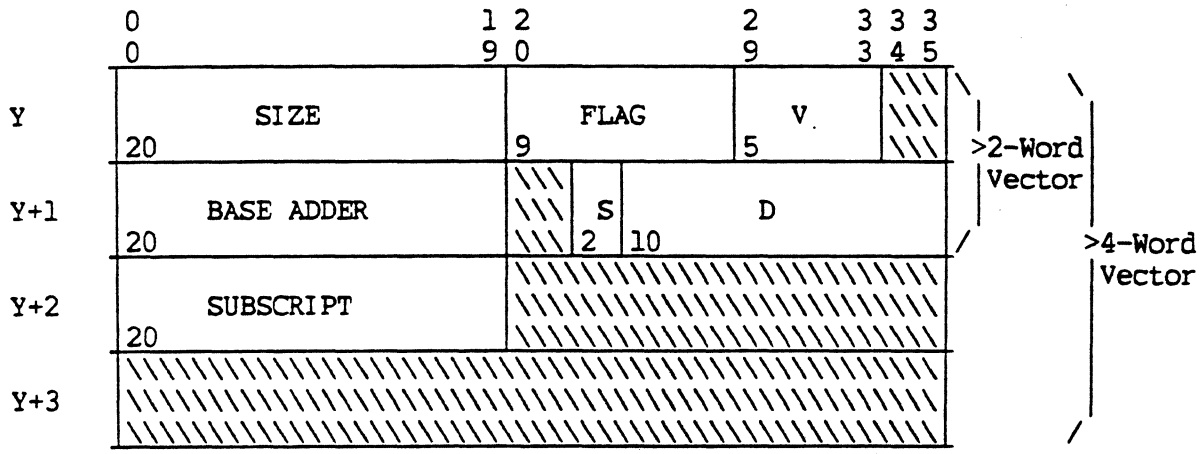
o SEGIDn is set to be self-identifying, i.e., S = 0, D = 177n.

Instruction Word Bit 29 = 0; DRm Type T = 0, 2, 4, 6, 12, or 14

The memory operand vector, consisting of one or two double-words determines the operation to be performed by the instruction. When this vector is obtained from memory, all address modification is permitted except for DU, DL, SC, SCR, and CI.

1. VECTOR FORMAT

a. Vector for Standard Segment Descriptor, Super Segment Descriptor



LDDn

The contents of bits 29-33 (the V field) determine the function to be performed as follows. (XXX for bits indicates that these bits are ignored.)

V = 00XXX Copy: 2-word vector

Copy (load) the selected segment descriptor into DRn. SEGIDn is set to indicate the location from which the segment descriptor was obtained; ARn is set to zero.

V = 01XXX Normal Shrink: 2-word vector

Shrink the selected segment descriptor and load it into DRn. SEDIDn is set to indicate DRn; ARn is set to zero.

V = 10000 Extended Shrink: 4-word vector

V = 10001 Special Extended Shrink: 4-word vector

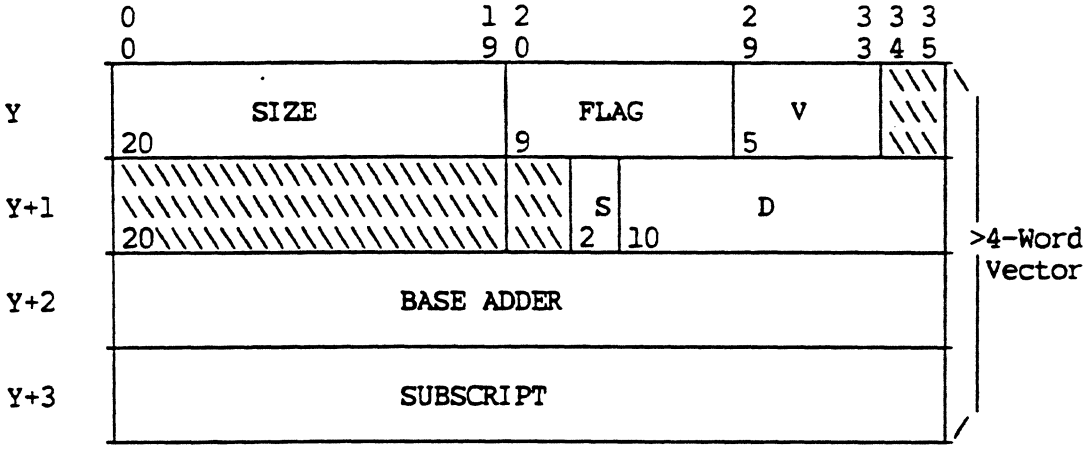
Shrink the selected segment descriptor with the 4-word vector and load it into DRn. SEGIDn is set to indicate DRn; ARn is set to zero. (Refer to details below for difference between Extended Shrink and Special Extended Shrink.)

V = 11XXX Data Stack Shrink: 2-word vector

Use DSDR and DSAR to generate the data stack segment descriptor; load this segment descriptor into DRn. DSAR is updated and ARn is set to zero. SEGID is set to indicate DRn.

LDDn

b. Vector for Extended Segment Descriptor



The contents of bits 29 - 33 determine the function to be performed with the format illustrated above as follows:

V = 10100 Normal Shrink with Type Change

Shrink the selected segment descriptor (T = 12 or 14) and change to a Standard Segment Descriptor. SEGIDn is set to indicate DRn; ARn is set to zero.

V = 10101 Normal Shrink with No Type Change

Shrink the selected segment descriptor (T = 12 or 14). SEGIDn is set to indicate DRn; ARn is set to zero.

V = 10110 Extended Shrink with Type Change

Shrink the selected segment descriptor (T = 12 or 14), by using a subscript, and change to a Standard Segment Descriptor. SEGIDn is set to indicate DRn; ARn is set to zero.

V = 10111 Extended Shrink with No Type Change

Shrink the selected segment descriptor (T = 12 or 14) by using a subscript. SEGIDn is set to indicate DRn; ARn is set to zero.

LDDn

V = 10010 Normal Base Shrink with No Type Change

Shrink the base of a selected segment descriptor (T = 0, 2, 12, 14) and reduce the bound by as much as the base shrinkage. The type remains unchanged, SEGID is set to indicate DRn; ARn is set to zero.

V = 10011 Extended Base Shrink with No Type Change

The same as the normal base shrink, except that the subscript is used. SEGIDn is set to indicate DRn and ARn is set to zero.

2. SHRINK FOR STANDARD AND SUPER SEGMENT DESCRIPTORS

a. V = 00XXX Copy (bits indicated by X ignored)

The S and D fields of the vector indicate the location of the segment descriptor to be loaded into DRn. Definition of these two fields follows.

When S = 0:

For D = 0000 through 1757 (octal), the descriptor is loaded from the parameter segment and D is used as an index to the desired descriptor. The value in D is the number of the descriptor to be loaded and can be treated as a modulo 8 byte index; that is, D can be converted to a byte address by appending three zeros as the three least-significant bits.

D is bound checked against the PSR (parameter Segment Register) bound field. If D > PSR bound, a Bound fault occurs. IF D <= PSR bound, D is added to the PSR base and is used as the segment descriptor address. This address is used to obtain the segment descriptor which is then loaded into DRn.

LDDn

For D = 1760 through 1777 (octal), the descriptors referenced by S, D are contained in selected registers and copied to the DRn.

D = 1760 Undefined, IPR fault
D = 1761 Change Descriptor Type Field in DRn
D = 1762 Instruction Segment Register (ISR)
D = 1763 Data Stack Descriptor Register (DSDR)
D = 1764 Safe Store Register (SSR)
D = 1765 Linkage Segment Register (LSR)
D = 1766 Argument Stack Register (ASR)
D = 1767 Parameter Segment Register (PSR)
D = 1770 DR0, Descriptor Register 0
D = 1771 DR1, Descriptor Register 1
D = 1772 DR2, Descriptor Register 2
D = 1773 DR3, Descriptor Register 3
D = 1774 DR4, Descriptor Register 4
D = 1775 DR5, Descriptor Register 5
D = 1776 DR6, Descriptor Register 6
D = 1777 DR7, Descriptor Register 7

NOTE: When S = 0 with D = 1761 (octal) and the processor is in the Privileged Master mode, if the descriptor contained in DRn is type 1 or 3, the type is changed to 0 or 2, respectively. SEGIDn is set to be self-identifying. However, if the descriptor is not type 1 or 3, no fault occurs and no operation is performed.

When S = 0 with D = 1761, 1763, or 1764 (octal), a command fault occurs unless the CPU is in the Privileged Master mode.

When S = 2

The Dth descriptor of the current argument segment is selected. A relative byte offset is formed by extending the D field by 3 zeros. D is bound-checked against the ASR bound field. If D > the ASR bound, a bound fault occurs. If D <= the bound, D is added to the ASR base, and the segment descriptor is obtained with this address and then loaded into DRn.

LDDn

When S = 1 or 3

The Dn descriptor of the current linkage segment is selected. A relative byte offset is formed by extending the D field by three zeros. D is bound-checked against the LSR bound field. If D > bound, a Bound fault occurs. If D <= the bound , D is added to the LSR base, and the segment descriptor is obtained with this address and then loaded into DRn.

For all values of S, the loading of DRn affects the nth address register (ARn) and the nth segment identity register (SEGIDn) as follows:

- o ARn is set to zero.
- o If DRn was loaded from another DR or the instruction segment register (ISR), the associated segment identity content is transferred to SEGIDn; otherwise, SEGIDn is set to the S and D value contained in the vector. When S = 0 and D = 1761 (octal), SEGIDn is set to be self-identifying.
- o If an IPR or an Bound fault occurs, DRn, ARn, and SEGIDn are not changed.

b. V = 01XXX Normal Shrink

When bits 29 and 30 of the first word in the vector are 01, the specified segment descriptor is obtained, the shrink operation is performed, and the descriptor is then loaded into DRn as with copy. When S = 0 and D = 1761 (octal) in the Privileged Master mode, the segment descriptors for type T = 1 or 3 are changed to T = 0 or 2, respectively. The shrink operation is then performed.

In order to perform the shrink operation, the segment descriptors indicated by S and D must be Standard or Super Segment descriptors. An IPR fault occurs if T = 5 or 7 - 15. If a fault, such as a Bound fault, occurs during the shrink operation, DRn, SEGIDn, and ARn are not changed.

LDDn

Standard Segment Descriptors

With standard segment descriptors, the shrink operation is performed as follows.

- The vector BASE ADDER and SIZE fields are the relative values for the selected segment descriptor base and bound fields. The following check is performed for these values.

  BASE ADDER + SIZE <= bound
  |___Bound fault occurs with carry.

  A Bound fault occurs when the sum of the BASE ADDER and SIZE exceeds the bound or when carry occurs with this addition. Flag bit 27 is not checked.

- When the check is terminated, a new base and bound are generated.

  New Base = old base + BASE ADDER
  |___Bound fault occurs with carry.

  New Bound = size

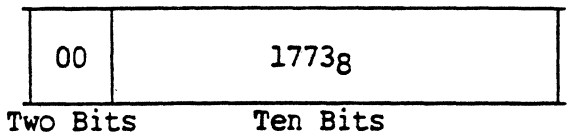
  The new base and bound are loaded into DRn.

- The vector flag field indicates the attributes given to the segment. It is combined with the flag field of the selected segment descriptor to generate a new flag field. The permission conditions for these new flags are such that they are not increased from the previous conditions (i.e., a bit-by-bit logical AND operation of two flag fields takes place). A fault does not occur even if the vector permission conditions are greater than those for the segment descriptors. The result produced by the combination of these two flag fields is loaded into the DRn flag field. As the type T = 2 or 3 segment descriptor flag field are three bits in length, the AND operation is performed for these three bits and the corresponding three bits from the vector.

  The corresponding ARn is set to zero.

LDDn

SEGIDn is set to be self-identifying (DRn); for example, when this instruction references DR3 (LDD#), SEGID3 is set as follows:

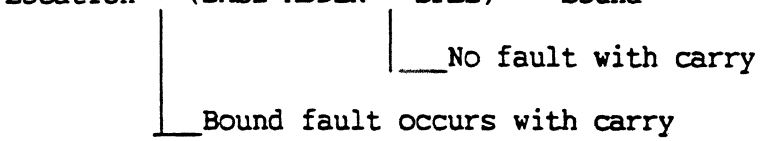


Super Segment Descriptors

When shrink operation is performed for a super segment descriptor, a standard segment descriptor is generated. Type T = 4 super segment descriptor becomes type T = 0 standard segment descriptor, and type T = 6 super segment descriptor becomes type T = 2 standard segment descriptor.

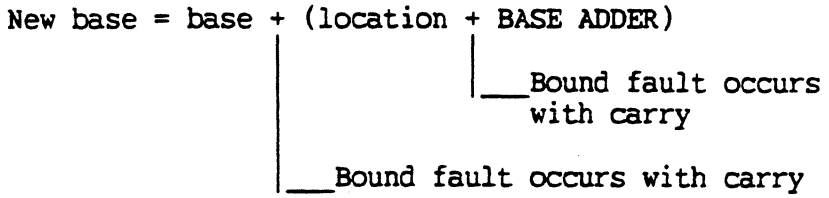
The shrink operation is performed as follows:

o A check is performed to determine whether the following expression is satisfied.



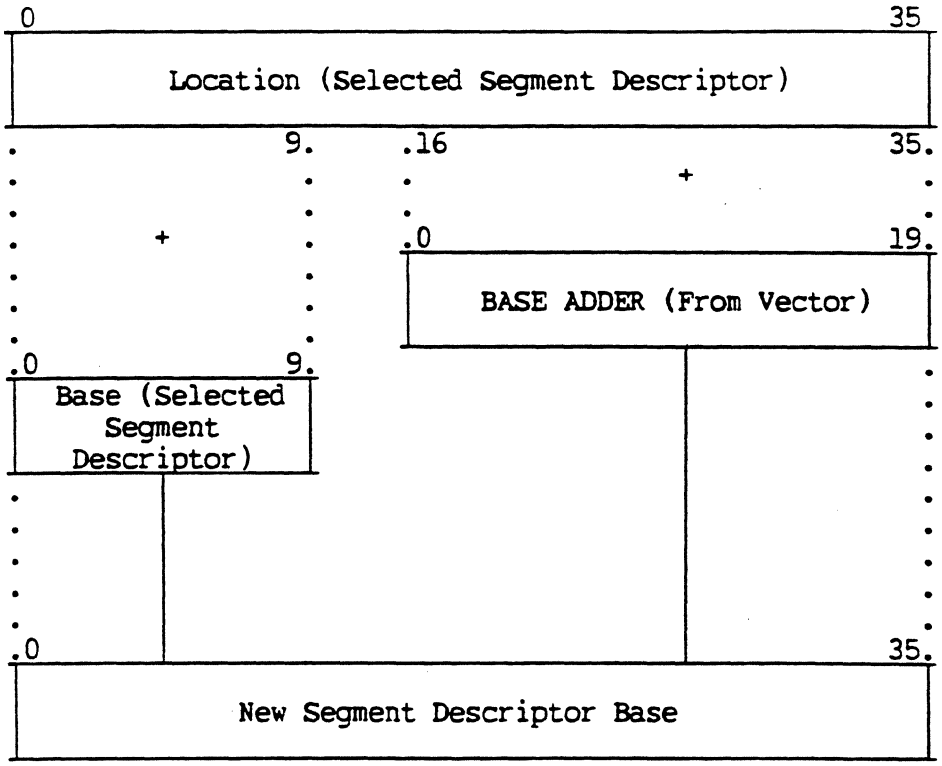
Flag bit 27 is not checked.

If this check is passed, a new base and bound are generated.



The processing is described in the diagram that follows relative to the base and bound fields of the selected descriptor.

LDDn



The new bound = SIZE. The new base and size field from the vector are loaded in the base and bound field of DRn.

The new flags field is formed in the same manner as for the standard descriptor. SEGIDn is set as for the standard descriptor shrink; ARn is zero-filled.

c. V = 10000 Extended Shrink

For extended shrink operations, the same conditions which exists for normal shrink operations must be satisfied. If a fault occurs during a shrink operation, DRn, ARn, and SEGIDn remain unchanged.

Standard Segment Descriptors

A 4-word vector subscript (SCPT) is used when the new segment descriptor base and bound are generated.

LDDn

- The following check is performed.

```
(BASE ADDER + SCPT) + (SIZE - SCPT) <= bound
                    |          |
                    |          |__Bound fault occurs
                    |             with borrow
                    |
                    |__Bound fault occurs with carry
                    |
                    |__Carry is ignored; a negative value is
                       permitted as the BASE ADDER (i.e., a
                       very large positive value).
```

- If this check is passed, a new base and bound are generated.

```
New base = base + (BASE ADDER + SCPT)
                  |              |
                  |              |__Carry is ignored.
                  |
                  |__Bound fault occurs with carry.

New bound = SIZE - SCPT
                   |
                   |__Bound fault occurs with borrow.
```

The new base and bound are loaded into DR<u>n</u>.

As described in the discussion on normal shrink of a standard segment descriptor, a new flag field is generated. SEGID<u>n</u> and AR<u>n</u> are set in the same way.

<u>Super Segment Descriptors</u>

The SCPT field is used as described in the discussion on standard segment descriptors.

The following check is performed.

LDDn

```
Location + (BASE ADDER + ACPT) + (SIZE - SCPT) <= bound
           |                |    |        |__Bound fault
           |                |    |           occurs with
           |                |    |           borrow.
           |                |    |
           |                |    |__Carry transferred to
           |                |       the location field.
           |                |
           |                |__Carry is ignored.
           |
           |__Bound fault occurs with carry.
```

If this check is passed, a new base and bound are generated.

```
New Base = base + (location + (BASE ADDER + SCPT))
                  |           |               |__Carry is
                  |           |                  ignored.
                  |           |
                  |           |__Bound fault occurs with
                  |              carry.
                  |
                  |__Bound fault occurs with carry.
```

```
New bound = SIZE - SCPT
                |__Bound fault occurs with borrow.
```

The new base and bound are loaded into DRn.

- A new flag field is generated as with a standard segment descriptor.
- DRn type T is set as follows.

  1) If old T = 4, then new T = 0

  2) If old T = 6, then new T = 2.

- The corresponding ARn is set to zero.
- SEGIDn is set to be self-identifying (DRn). The flag bit 27 of the selected segment descriptor is not checked.

LDDn

d. V = 10001 Special Extended Shrink

The differences between the special extended shrink and the extended shrink (V = 10000) are as follows.

If the type T of the fetched segment descriptor is not equal to 0, 1, 2, or 3, an IPR fault occurs. The SIZE field (bits 0 - 17) of the vector is ignored, and the following check is made.

BASE ADDER + SCPT <= old bound
|
|__Carry is ignored.

A new base and bound are created as follows.

New base = old base - (BASE ADDER + SCPT)
| |
| |__Carry is ignored
|
|__Bound fault occurs if a borrow is generated.

New bound = old bound - (BASE ADDER + SCPT)
| |
| |__Carry is ignored
|
|__Bound fault occurs if a borrow is generated.

LDDn

e. V = 11XXX Data Stack Shrink

When bits 29 and 30 of the first word in the vector are 11, the instruction performs the data stack shrink operation. The second word in the vector is ignored. DSDR, DSAR, and the SIZE and flag field of the first word in the vector are used to generate the new segment descriptor.

- The value in the SIZE field of the vector is checked to determine whether the area between the location currently specified by the DSAR and the value specified by the DSDR bound is equal or greater than the SIZE field. The lower three bits of the vector SIZE field are set to 1 to indicate an even-word boundary (i.e., it is rounded to a double-word expression as the DSAR always specifies an even-word boundary.) DSAR + SIZE (rounded-up) <= DSDR bound is then checked. If the left portion of this expression exceeds the DSDR bound, or if carry occurs as a result of the addition to the left, a Bound fault is generated. In this case DRn, ARn, and SEGIDn are not changed.

- If this check is passed, the DSAR content is added to the DSDR base and a new base is generated. If carry occurs, a bound fault occurs and the register content is not changed.

- The new base (DSAR + DSDR base) is then loaded into the DRn base field and the vector SIZE (before rounding) is loaded into the DRn bound field.

- The new flag field values are generated from the vector flag field and the DSDR flag field following the same method as that described for normal shrink of standard segment descriptors.

- The content of the DSDR W and T fields are moved to the DRn W and T fields.

- The corresponding ARn is set to zero.

- SEGIDn is set to be self-identifying (DRn), as with normal shrink.

LDDn

- The following value is loaded into DSAR.

  New DSAR = DSAR + SIZE (rounded-up) + 1 (byte)

  As wraparound is not permitted for the DSAR, a bound fault occurs if carry occurs with the above addition.

3. SHRINK FOR EXTENDED SEGMENT DESCRIPTORS

   a. V = 10100 Normal Shrink with Type Change

   The segment descriptor indicated by the S, D fields of a vector is fetched in the same way as by the copy function. If the type T of the fetched segment descriptor is not 12 or 14, an IPR fault occurs. For a valid segment descriptor, the shrink operation is performed as follows.

   - The following check is made.

     BASE ADDER + SIZE <= bound (11.....1)
                                 12 bits

     If the sum of the BASE ADDER and SIZE exceeds the value obtained by extending the bound of the fetched segment descriptor 12 "1" bits to the right, or if the addition produces a carry from the most significant bit, a bound fault occurs.

   - After this check, a new base and bound are created.

     New base = old base + BASE ADDER
                           |
                           |__Bound fault occurs if carry is generated.

     New bound = SIZE

   - A new flag field is created in the same way as for

     V = 01XXX normal shrink.

   - A new type T is set as follows.

     If old T = 12, then new T = 0.

     If old T = 14, then new T = 2.

LDDn

- o SEGIDn and ARn are set in the same way as for normal shrink.

b. V = 10101 Normal Shrink with No Type Change

The segment descriptor indicated by the S, D fields of a vector is obtained in the same way as for the copy function. An IPR fault occurs if the type T of the fetched segment descriptor is not 12 or 14. For a valid descriptor, the shrink operation is performed as follows.

```
                              12 bits
BASE ADDER + (SIZE 00.......0+base lower-order 12 bits)

                12 bits
<= bound 11........1
```

where the base denotes the value of the base field of the fetched segment descriptor.

First, the sum of the value obtained by extending the SIZE 12 bits to the right and the low-order 12 bits of the base is obtained. If this sum plus the BASE ADDER exceeds the value obtained by extending the bound of the descriptor 12 bits to the right, or if a carry is generated by the addition, a Bound fault occurs.

- o After the check, a new base and bound are created.

  New base = old base + BASE ADDER

  Bound fault occurs if carry is generated.

  New bound = SIZE

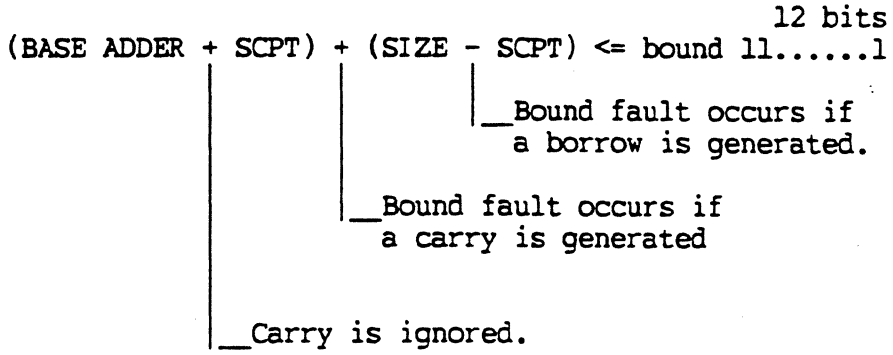
- o SEGIDn and ARn are set in the same way as for normal shrink.

LDDn

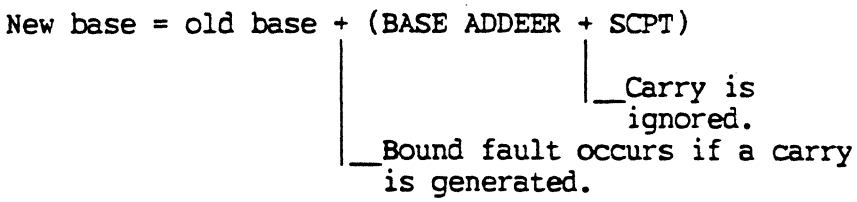
c. V = 10110 Extended Shrink with Type Change

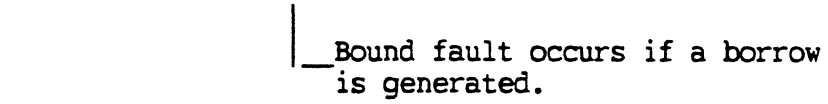
The segment descriptor indicated by the S, D fields of a vector is obtained in the same way as for the copy function. An IPR fault occurs if the type T of the fetched segment descriptor is not 12 or 14. For a valid segment descriptor, the shrink operation is performed as follows.

- The following checks are made on the BASE ADDER and SIZE fields of the vector.



- After the check, a new base and bound are created.





- A new flag field is created in the same way as for a normal shrink (V = 01XXX).

- A new type is set as follows.

  If old T = 12, then new T = 0.

  IF old T = 14, then new T = 2.

- SEGIDn and ARn are set in the same way as for a normal shrink.

d. V = 10111 Extended Shrink with No Type Change

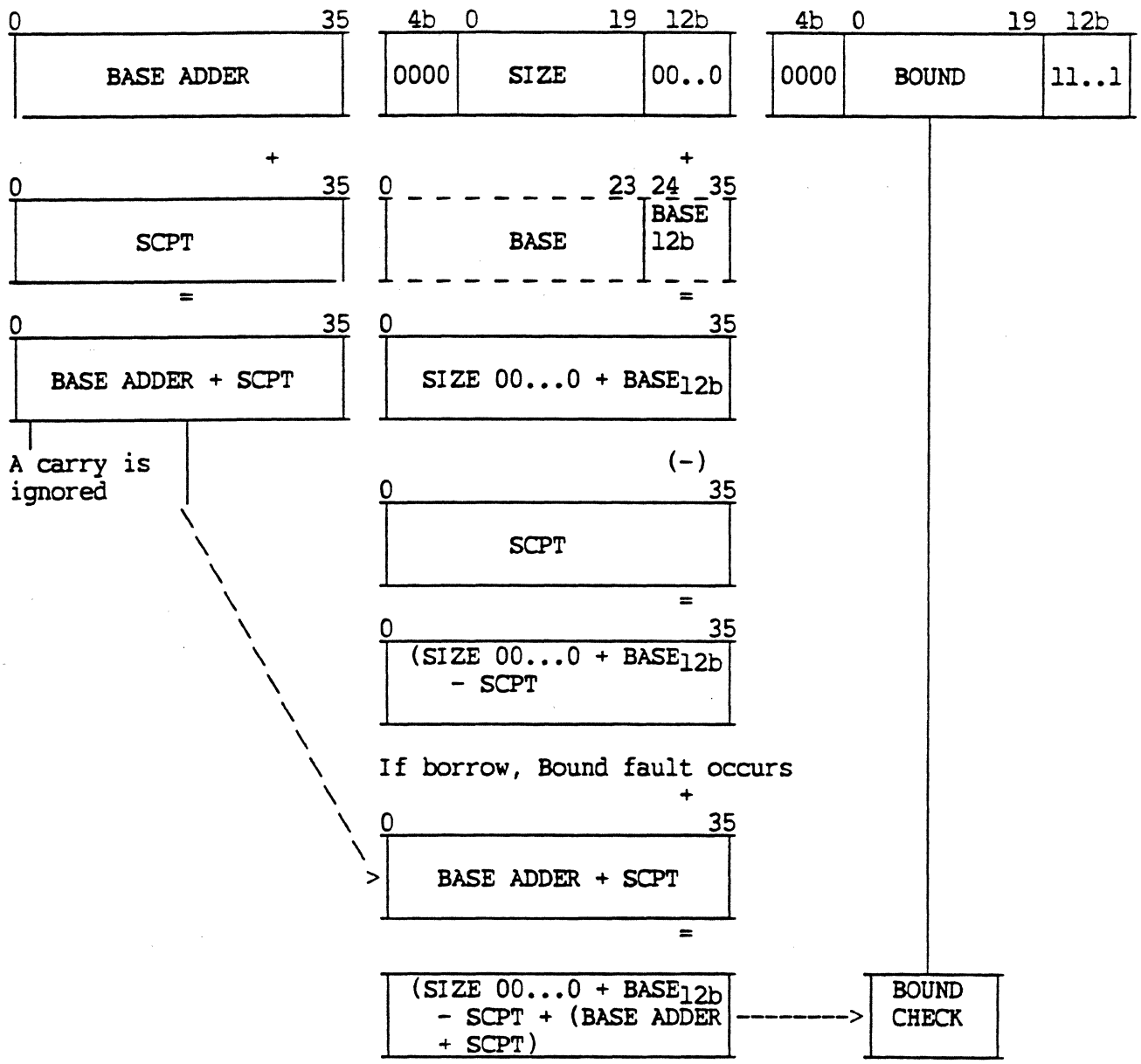
The segment descriptor indicated by the S, D fields of a vector is obtained as for the copy function. An IPR fault occurs if the type T of the fetched segment descriptor is not 12 or 14. For a valid descriptor the shrink operation is performed as follows.

- The following check is made on the BASE ADDER and SIZE fields of the vector.

```
(BASE ADDER + SCPT)
              |
              |__Carry is ignored.

              12 bits
 + [(SIZE 00........0 + base low-order 12 bits)-SCPT]
 |                                                  |
 |                          Bound fault occurs if_ |
 |                          a borrow is generated.
 |
 |__Bound fault occurs if a carry is generated.

               12 bits
  <= bound 11........1
```

First, the sum of the value obtained by extending SIZE 12 bits to the right and the low-order 12 bits of the base of the fetched segment descriptor is obtained. The difference between this sum and SCPT is obtained. The difference is added to the sum of the BASE ADDER and SCPT.

Second, this sum is compared to the value obtained by extending the bound of the fetched descriptor 12 bits to the right. This operation is illustrated as follows.



If a carry is generated, Bound fault occurs.

- After the check, a new base and bound are created.

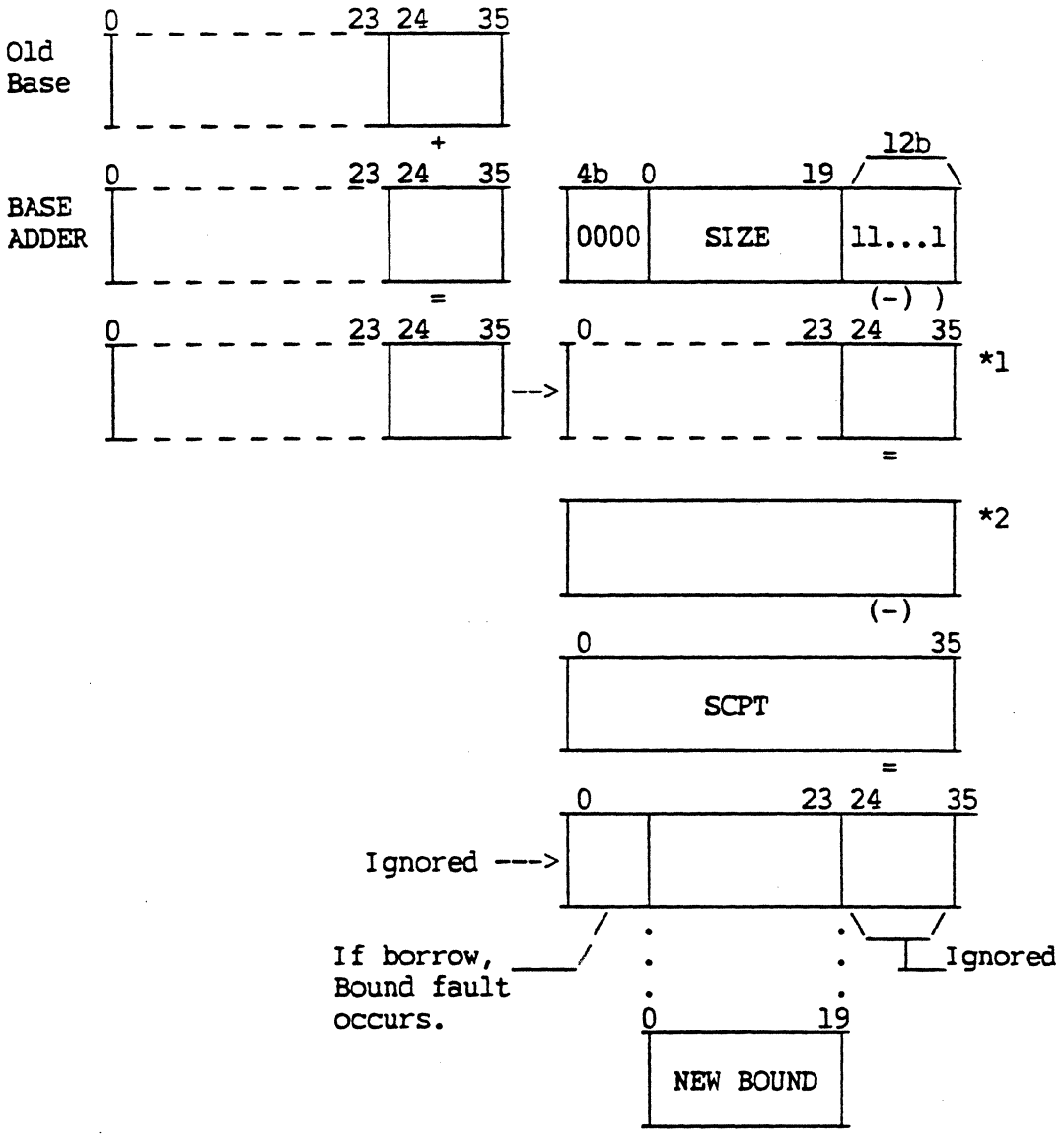
LDDn

New base = old base + (BASE ADDER + SCPT)
                      | |__Carry is
                      | ignored.
                      |
                      |__Bound fault occurs if a carry
                         is generated.

                          12 bits
New bound = [{SIZE 11.....1
              - (old base$_{\text{low-order 12 bits}}$
              + BASE ADDER$_{\text{low order 12 bits}}$}
              - SCPT]$_{4-23}$
                |
                |__Bound fault occurs if a borrow
                   is generated.

The following illustrates locating the new bound.

LDDn



*1: (Old base + BASE ADDR)$_{\text{low-order 12 bits}}$
12 bits

*2: SIZE 11.....1 - (old base + BASE ADDER)$_{\text{low 12 bits}}$
Flag fields are handled as a normal shrink.

- A new type T is the same as the original (old) type T.
- SEGIDn and ARn are set in the same way as for normal shrink.

LDDn

e. V = 10010 Normal Base Shrink with No Type Change

The segment descriptor indicated by the S, D fields of a vector is obtained in the same way as for the copy function. An IPR fault occurs if the type T of the fetched segment descriptor is not 0, 2, 12, or 14.

The SIZE field of the vector is ignored in the processing for a valid descriptor illustrated below.

- The following check is made on the BASE ADDER of the vector.

  16 bits
  BASE ADDER <= 00.....0 bound

  If the condition in the above check is not met, a Bound fault occurs.

- After the check, a new base and bound are created as follows.

  New base = old base + BASE ADDER
  |
  |__Bound fault occurs if a carry is generated.

  16 bits
  New bound = [00.....0 bound - BASE ADDER]$_{16-35}$

- A new flag field is created the same as for a normal shrink (V = 01XXX).

- A new type T is the same as the original (old) type T.

- SEGIDn and ARn are set in the same way as for a normal shrink.

For a segment descriptor with T = 12 or 14, the shrink operation is performed as follows.

LDDn

- The following check is made on BASE ADDER of the vector.

```
                                          4 bits      12 bits
BASE ADDER+base low-ord 12 bits <=0000 bound 11....1|
           |
           |__Bound fault occurs if a
              carry is generated.
```

where the low-order 12 bits of base are the low-order 12-bits of the base field of the fetched segment descriptor.

If the above condition is not met, a Bound fault occurs.

- After the check, a new base and bound are created as follows.

```
New base = old base + BASE ADDER
                    |
                    |__ Bound fault occurs if a carry is
                       generated.

                 4 bits          12 bits
New bound = [(0000 old bound 11.......1
                - old base low-order 12 bits)
                - BASE ADDER]4-23
```

- A new flag field is created in the same way as for a normal shrink (V = 01XXX).

- The new type T is the same as the original (old) type T.

- SEGIDn and ARn are set in the same way as for the normal shrink.

f. V = 10011 Extended Base Shrink with No Type Change

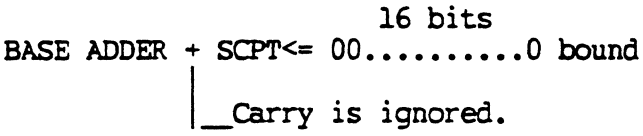
The segment descriptor indicated by the S, D fields of a vector is located in the same way as for the copy function. An IPR fault occurs if the type T of the fetched descriptor is not 0, 2, 12, or 14.

LDDn

The SIZE field of the vector is ignored in the processing described below.

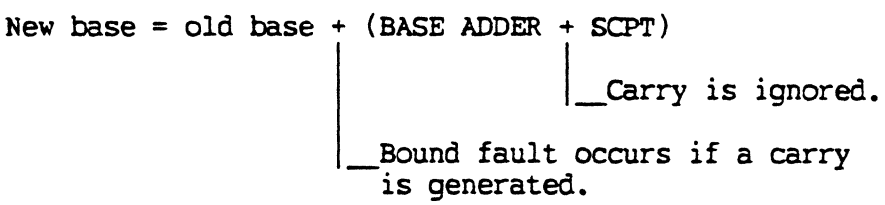
For a segment descriptor with T = 0 or 2, the shrink operation is performed as follows.

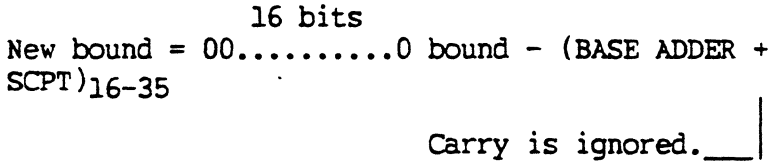
- The following check is made on the BASE ADDER and the SCPT of the vector.



  If these conditions are not met, a Bound fault occurs.

- After the check, a new base and bound are created as follows.





  A new flag field is created in the same way as for a normal shrink (V = 01XXX).

  The new type T is the same as the original (old) type T.

  SEGIDn and ARn are set in the same way as for normal shrink.

For a segment descriptor with T = 12 or 14, the shrink operation is performed as follows.

- The following check is made on the BASE ADDER and SUBSCRIPT (SCPT) of the vector.

LDDn

```
(BASE ADDER + SCPT) + base low-order 12 bits
                    |       |__Bound fault occurs if a carry
                    |          is generated.
                    |
                    |__Carry is ignored.

  4 bits       12 bits
<= 0000 bound 11.......1   * (Referred to by NOTE
below.)
```

Where the base low-order 12 bits are the low-order 12 bits of the base field of the fetched segment descriptor. If this condition is not met, a bound fault occurs.

- After the check, a new base and bound are created as follows.

```
New base = old base + (BASE ADDER + SCPT)
                    |               |__Carry is ignored.
                    |
                    |__Bound fault occurs if a carry
                       is generated.

                4 bits           12 bits
New bound = [(0000 old bound 11.......1
              old base low-order 12 bits)
            - (BASE ADDER + SCPT)]4-23
            |               |__Carry is ignored.
            |
            |__Bound fault occurs if a borrow is
               generated.
```

NOTE: This Bound fault will never occur if the starred (*) check condition above has been met.

- A new flag field is created in the same way as for a normal shrink (V = 01XXX).
- A new type T is the same as the original type T.
- SEGIDn and ARn are set in the same way as for a normal shrink.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, IR, RI, IT, CI, SC, SCR (See NOTES for explanation.)

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Illegal Procedure (IPR) Faults can be caused by any of the following conditions:

   a. Modifications RI, IR, IT, DU, and DL when the DRm segment descriptor type T = 1 or 3

   b. Modifications DU, DL, CI, SC, SCR when the DRm segment descriptor type T = 0, 2, 4, 6, 12, or 14

   c. Illegal repeats

   d. Vector fields S = 0 and D = 1760 (octal)

   e. If vector bits 29 and 30 are 01 or 10 and descriptor obtained is type T=5 or 7-15

   f. If a carry occurs when a T = 4 or 6 super descriptor is loaded into DRn, and it is converted by hardware to a standard segment descriptor. (Refer to description of "Super Descriptors" in Section 3.)

   g. When instruction word bit 29 = 1 and DRm segment descriptor is type T = 5 or 7-11, 13, 15

2. Command Faults can be caused by any of the following conditions:

   a. If the CPU is not in Privileged Master mode, when S = 0 and D = 1761, 1763, or 1764 (octal)

   b. If the CPU is not in Privileged Master mode, when bits 29 and 30 of the first word in the vector do not specify data stack shrink (V = 11XXX) and the vector S and D fields specify DSDR

      NOTE: When CPU is in the Privileged Master mode, the segment descriptor from DSDR is used to execute the specified operation. In this instance, DSDR and DSAR remain unchanged.

LDDn

3. Bound Faults can be caused by any of the following conditions:

   a. When S = 0 and D > PSR bound

   b. When S = 2 and D > ASR bound

   c. When S = 1 or 3 and D > LSR bound

   d. When BASE ADDER + vector SIZE > DRn bound with shrink operation for standard descriptors

   e. When DRn location + vector BASE ADDER + vector SIZE > DRn bound with shrink operation for super descriptors

   f. When an illegal carry or borrow occurs while a base and bound are generated, while a size check is performed, or while a new DSAR is generated

   g. In addition, general fault conditions also apply when segment descriptors and page tables are accessed. These conditions are noted in the individual vector procedures descriptions.

4. Security Fault, Class 1 can be caused by the following condition:

   a. If the housekeeping bit of the page which includes the selected descriptor is OFF when a descriptor is loaded with the LDD instruction

EXAMPLES:

Direct Load:

```
1       8       16              32
        LDD0    0,,7            Load DR0 from location zero
                                of descriptor segment
                                framed by DR7 1770 --> SEGID0
                                zeros --> AR0
```

Copy:

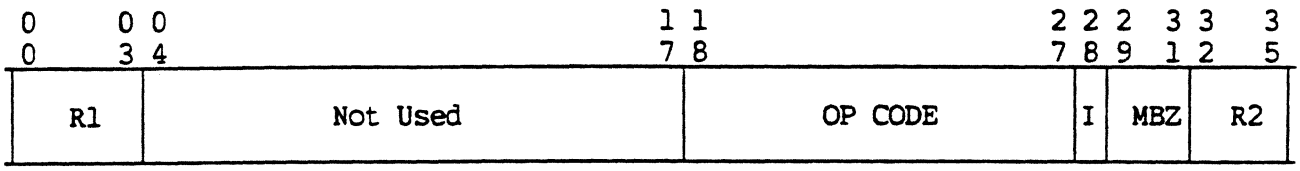
```
1       8       16              32
        LDD0    CPYDR7          Copy DR7 into DR0 1777-->SEGID0
                                zeros-->AR0
         .
         .
CRYDR7  CVEC    .DR7
```

Normal Shrink:

```
1       8       16              32
        LDD0    BUFVEC
         .
         .
BUFFER  BSS     320
BUFLEN  EQU     *-BUFFER
BUFVEC  VEC     .ISR,BUFFER,BUFLEN,READ
```

LDDR	Load Double Register to Register Pair	433 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
LDDR    R1,,R2
```

OPERATING MODES: Executes in ES mode only

SUMMARY: R1, R2, = 0, 2, 4, 6, AQ

C(R2-pair) --> C(R1-pair)

C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1-pair) = 0, then ON; otherwise, OFF

Negative - If $C(R1\text{-pair})_0 = 1$, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

LDDSA	Load Data Stack Address Register	170 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master Mode

SUMMARY: Bits 0-16 of C(Y) --> C(DSAR)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The DSAR is a 17-bit register that holds an even-word address.
2. An IPR fault occurs if illegal address modifications and illegal repeats are executed.
3. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

EXAMPLE:

```
1       8       16
        LDP     P,PSH,SD.PSH,DL
        LDP     P,PSH,.CTYP,DL
        LDDSD   PH.ADS,,P.PSH
        STZ     TEMP,,P.DSR
        LDDSA   TEMP,,P.DSR
```

LDDSD	Load Data Stack Descriptor Register	571 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master Mode

SUMMARY: C(Y-pair) --> C(DSDR)

EXPLANATION: The double-word memory operand is fetched from even and odd memory locations Y and Y+1. The operand must be in standard descriptor format with a type field of T = 0. The lower three bits of the base of this segment descriptor must be zero (i.e., the descriptor in the DSDR specifies the segment beginning from the boundary of an even word). The flag bit 22 must be zero.

When these conditions are met, the obtained descriptor is loaded into the DSDR. If one or more of the above conditions are not met, an IPR fault occurs and the DSDR content remains unchanged.

The lower three bits of the descriptor bound field should all be ones to ensure that the area specified with the DSDR is a multiple of word pairs. Hardware does not check these three bits.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Any of the following conditions causes an IPR fault (the DSDR remains unchanged):

   a. Illegal address modification

   b. Illegal repeats

   c. If type field T is not equal to 0.

   d. If the base is not modulo 2 words.

   e. If the descriptor flag bit 22 is not = 0.

2. If the processor is Master or Slave mode, the execution of this instruction causes a Command fault.

EXAMPLE:

```
1       8       16                  32
EXP     LDP     P0,SD.PSH,DL
        LDD     P0,PH.USL,,P0
        LDP     P0,.CTYP,DL
        ADLA    UL.ISR+1,,P0
        STA     S.ISR+1,QU,P4
        LDD     P1,S.ISR,QU,P3      P1 = sub-dispatch ISR
        LDAS    S.APR,,P4           load special registers
        LDPS    S.APR,,P4
        LDDSD   S.DSR,,P4
        LDDSA   SBDH
        LDSS    .KLSDS,PN*,P.KL     load SSR for sub-disp by processor number

        STX6    .KLPRG,7,P.KL       set processor flags for sub-disp
        SXL3    .KLPRG,7,P.KL
        LDD     P2,S.ENT,QU,P3      P2 = entry descriptor to climb with
        LCQ     =0204020,DL
        ANSQ    .QFST,3,P6          clear fault status bits
```

LDE	Load Exponent Register	411 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: $C(Y)_{0-7}$ --> C(E); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - Set OFF

Negative - Set OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

LDEAn	Load Extended Address n	61n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Y) --> location field of Descriptor Register (DRn)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. This set of eight instructions enables the loading of the location field of a descriptor register (DRn) from memory address Y. The DRn must contain a super descriptor (type field T must be 4 or 6); otherwise, an IPR fault occurs.

2. If T = 4 or 6, if a carry occurs when creating the base (DRn base+location field) or, if a borrow occurs when creating the bound (DRn bound-location field), an IPR fault occurs.

3. Any of the following conditions causes an IPR fault:

   a. Illegal address modifications

   b. Illegal repeats

   c. If descriptor type field T of DRn is not 4 or 6

LDEAn

EXAMPLE:

```
1       8       16              32
MSCN7   NULL
        EAX2    1,2
        CMPX2   4,DU            is defective memory table full?
        TZE     ESCN            yes
        LDA     .KLMSZ,,KLS     no
        ANA     =O777777,DL     isolate real memory size
        AOS     ADDRS           advance page number
        CMPA    ADDRS           is this page the last?
        TZE     ESCN            yes
        LDEA    RMS,SUPAD       loading location field of super descriptor
        LDA     1K*4,DL         adjust byte
        ASA     SUPAD
        TRA     MSCN2           next page scan
```

LDI	Load Indicator Register	634 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: $C(Y)_{18-32}$ --> C(IR); C(Y) unchanged

EXPLANATION: The relation between bit positions of C(Y) and the indicators is as follows:

Bit Position	Indicator (or Mask)
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent overflow
23	Exponent underflow
24	Overflow mask
25	Tally runout
26	Parity error
27	Parity mask
28	Master mode
29	Truncation
30	Multiword instruction interrupt
31	Reserved for exponent underflow mask
32	Hexadecimal mode
33-35	Undefined

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Master mode (IR bit 28) not affected

All others: If corresponding bit in C(Y) = 1, then ON; otherwise, OFF

LDI

NOTES:

1. The Tally Runout indicator reflects bit 25 of C(Y) regardless of what address modification is performed on the LDI instruction for tally operations.

2. Master Mode cannot be changed by the LDI instruction.

3. An Overflow Fault does not occur when the overflow indicator, exponent overflow indicator, or exponent underflow indicator is set ON via the LDI instruction, even if the overflow mask indicator is OFF.

4. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

5. Hexadecimal mode is controlled by bit 32 of the IR and bit 33 of the mode register.

6. The parity mask, bit 27, masks SCU interface parity errors and internal CPU parity errors in Master mode. In Slave mode, only SCU interface parity errors are masked. The test mode register control can be used to mask internal parity errors.

LDO

LDO	Load Option Register	172 (1)

FORMAT: Single-word instruction format (see Figure 8-1).

OPERATING MODES: Any. See Explanation below.

EXPLANATION: When the CPU is in Privileged Master mode:

Data Stack Clear Flag (DSCF) is loaded from $C(Y)_{18}$. DSCF controls memory clear operation when data stack shrink is executed with the CLIMB instruction.

0 = do not clear
1 = clear

Safe Store Bypass Flag (SSBF) is loaded from $C(Y)_{19}$. SSBF controls ICLIMB safe store bypass.

0 = bypass safe store
1 = perform safe store

If the CPU is in Master or Slave mode, DSCF and SSBF are unchanged.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Although this instruction is legal in all processor modes, the setting of the two flag bits occurs only in Privileged Master mode.

2. An IPR fault occurs if illegal address modification or illegal repeats are executed.319

EXAMPLE:

```
1       8       16                  32
*       LOAD SAFE STORE REGISTER AND OPTION REGISTER; Privileged Master
                                                          mode only
        LDSS    CPOSS
        LDO     =0200000,DL         SSBF ON
        TRA     MSFRM
SLVSS   LDSS    CPNOSS
        LDO     =0400000,DL         DSCF ON
        .
        .
```

LDPn	Load Pointer Register n	47n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: This set of eight instructions is similar to the LDDn instruction with the copy option; however, no vector is required and ARn may be loaded with a value other than all zeros.

Processing for these instructions differs between NS and ES modes.

NS Mode

If DU or DL modifications are not used

$C(Y)_{0-23}$ --> C(ARn)

C(descriptor specified by S, D) --> C(DRn)
or the DRn type field is changed.

$C(Y)_{24-35}$ interpreted as S,D field

If DU modification is used

Y_{0-17} --> $C(ARn)_{0-17}$

00...0 --> $C(ARn)_{18-23}$

00...0 interpreted as S,D field

If DL modification is used

00...0 --> $C(ARn)_{0-17}$

Y_{0-5} --> $C(ARn)_{18-23}$

Y_{6-17} interpreted as S,D field

LDPn

ES Mode

If DU or DL modifications are not used

$C(Y)_{0-35}$ --> C(ARn)

C(descriptor specified by S, D) --> C(DRn)
or the DRn type field is changed.

$C(Y+1)_{0-11}$ interpreted as S,D field

$C(Y+1)_{12-35}$ ignored

If DU modification is used

Y_{16-33} --> $C(ARn)_{0-17}$

00...0 --> $C(ARn)_{18-35}$

00...0 interpreted as S,D field

If DL modification is used

Y_{0-21} --> $C(AR)_{14-35}$

00...0 --> $C(ARn)_{0-13}$

Y_{22-33} interpreted as S and D

In both the NS and ES modes, interpretation of the S and D fields and the corresponding operation is the same as that for the LDDn instruction vector S and D fields specified by the copy function. The descriptor is loaded into DRn. (When S = 0 and D = 1761, the type in DRn is changed; the value described with the LDDn instruction copy function is loaded into SEGIDn.)

The S and D fields of the pointer locate the descriptor to be loaded into DRn as follows:

LDPn

When S = 0:

For D = 0000 through 1757 (octal) and D <= PSR bound, the descriptor is loaded from the parameter segment and D is used as an index to the desired descriptor. The value in D is the number of the descriptor to be loaded and can be treated as a modulo 8 index; that is, D can be converted to a byte address by appending three zeros as the three least significant bits.

For D = 1760 through 1777 (octal), the descriptors referenced by S, D are contained in selected registers and copied to DRn.

D = 1760	Undefined, IPR fault
D = 1761	Change Descriptor Type Field in DRn
D = 1762	Instruction Segment Register (ISR)
D = 1763	Data Stack Descriptor Register (DSDR)
D = 1764	Safe Store Register (SSR)
D = 1765	Linkage Segment Register (LSR)
D = 1766	Argument Stack Register (ASR)
D = 1767	Parameter Segment Register (PSR)
D = 1770	DR0, Descriptor Register 0
D = 1771	DR1, Descriptor Register 1
D = 1772	DR2, Descriptor Register 2
D = 1773	DR3, Descriptor Register 3
D = 1774	DR4, Descriptor Register 4
D = 1775	DR5, Descriptor Register 5
D = 1776	DR6, Descriptor Register 6
D = 1777	DR7, Descriptor Register 7

NOTE: When D = 1761 (octal) and the processor is in Privileged Master mode, if the descriptor contained in DRn is type 1 or 3, the type is changed to 0 or 2, respectively; however, if the descriptor is not type 1 or 3, no change is made and no fault occurs.

When S = 2:

The Dn descriptor of the current argument segment is selected. A relative byte offset is formed by extending the D field by 3 zeros.

LDPn

When S = 1 or 3:

The Dn descriptor of the current linkage segment is selected. A relative byte offset is formed by extending the D field by 3 zeros.

For all values of S, the loading of DRn affects the nth address register (ARn) and the nth segment identity register (SEGIDn) as follows:

a. ARn is set to zero.

b. If DRn was loaded from another DR or the instruction segment register (ISR), the associated segment identity content is transferred to SEGIDn; otherwise, SEGIDn is set to the S and D value contained in the pointer.

c. If an IPR or Bound fault occurs, DRn, ARn, and SEGIDn are not changed.

The segment descriptor (SD) compare funtionality increases the averrage speed of this instruction in both NS and ES modes. A comparison is made between the SD number of the instruction and the SD number in the SEGIDn register. If a match occurs, the memory access for the descriptor and the descriptor register load is bypassed, because the match indicates that the descriptor register is already correctly loaded. The address register level load is independent of a match.

The compare is not done if SD - 00,1760 to 00,1777.

A compare flag is provided for each descriptor register. All flags are set OFF, disallowing compares by instructions which can store descriptors, change characteristics of virtual spac, or change mode to slave. No provision is made for broadcasting this action to other processors within these instructions.

LDPn

The instructions which set these flags off follow.

ICLIMB
LTRAS
LTRAD
OCLIMB
LDAS
LDPS
LDWS
LPDBR
PAS
STDn if DRm type = 1,3
RET
TSS

Flag n is set ON by execution of LDPn.

In addition, the instruction, SPCF, turns the flags OFF.

The compare function is enabled or disabled under control of the CPU mode registers bits 24 and 25. Bit 24 enables compares in Slave mode; bit 25 enables compare in Master and Privileged Master modes. (Two controls are provided to allow the GCOS 8 software flexibility in removing code which would cause erroneous SD number matches.)

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if bit 29=1 and the operand segment is not type T = 0, 2, 4, or 6.

2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

3. A Command fault occurs as with the LDDn instruction copy function.

4. Other faults occur as with the LDDn copy function.

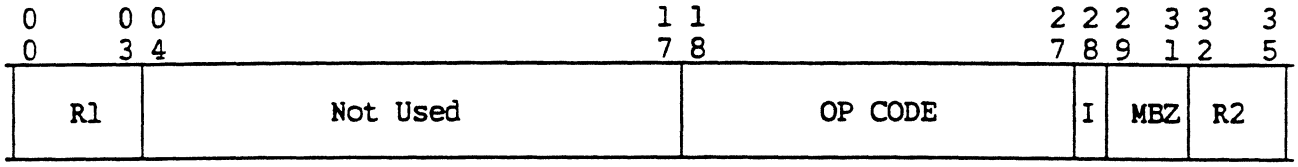
EXAMPLE:

```
1       8       16              32
TPUTEX  SZN     TRAPTR          test for trap in use
        TZE     TRAPOK          no trap enabled
        LDP6    TRAPTR          trapping -- get location (ensuring that
                                address register has offset and
                                descriptor is type 0) of cell to be
                                monitored in AR via P6; mask it for
                                desired pattern, and compare it with bad
                                value
        SAR6    TRAPCT
        LDP6    TRAPCT
        LDA     0,,P6
        ANA     TRAPMK
        CMPA    TRAPVL
        TZE     GOTCHA          trap has sprung
TRAPOK  LDP6    SD.SSA,DL       reload P.SSA (here if no/OK trap)
*                               TRA monitor if monitor active
        TRA     0,4             exit
```

LDPR

LDPR	Load Positive Register to Register	432 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
        LDPR    R1,,R2
```

OPERATING MODES: Executes only in ES mode.

SUMMARY: R1, R2 : 0, 1, 2, 3, 4, 5, 6, 7, A, Q

|C(R2)| --> C(R1)

C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - Set to OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

LDPS	Load Parameter Segment Register	771 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(Y-pair) --> C(PSR); C(Y-pair) unchanged

EXPLANATION: The descriptor is fetched from even/odd memory locations Y and Y+1. The hardware performs the following checks on the descriptor.

- o Type field must have a value of T = 1.
- o Base must be 0 modulo 8 bytes.
- o If flag bit 27 = 1 (bound valid), bound must be 7 modulo 8 bytes.

If these conditions are met, the descriptor is loaded into PSR. During PSR load, PSR bound field bits 0-6 are forced to zero by the hardware rather than being loaded from the memory operand. Also, if flag bit 27 of the operand descriptor is equal to zero, the entire bound field of the PSR is forced to zero, independent of any value the operand descriptor bound field may contain, and the bound check is bypassed.

This instruction is identical with LDAS, except that it loads the parameter segment register (PSR) instead of the argument stack register (ASR).

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Any of the following conditions cause an IPR fault:
   a. Illegal address modifications
   b. Illegal repeats
   c. Descriptor type field T is not 1
   d. If the base and bound limits of the operand descriptor are not modulo 2 words (only when flag bit 27 = 1).
2. If the processor is in Master or Slave mode, the execution of this instruction causes a Command fault.

EXAMPLE:

```
1       8       16                  32
        LDP     P.SSR,.SSR,DL       (Load descriptor of fault
                                     frame in safe store stack)
        LDP     P.SSR,.CTYP,DL      (Change to type 0)
        LDAS    .WASR,,P.SSR        (Restore ASR from safe store)
        LDPS    .WPSR,,P.SSR        (Restore PSR from safe store)
```

LDQ	Load Q-Register	236 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Y) --> C(Q); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

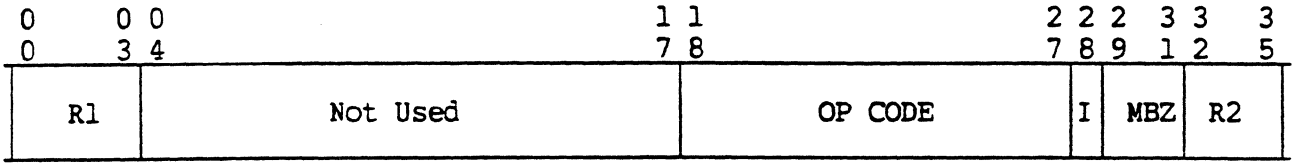
ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0 = 1$, then ON; otherwise, OFF

LDRR	Load Register from Register	430 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
        LDRR    R1,R2
```

OPERATING MODES: Executes in ES mode only.

SUMMARY: R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R2) --> C(R1)

C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

LDSS	Load Safe Store Register	773 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master Mode

SUMMARY: $C(Y)_{0-35}$ --> $C(SSR)_{0-35}$

$C(Y+1)_{0-32}$ --> $C(SSR)_{36-68}$

000 --> $C(SSR)_{69-71}$

EXPLANATION: The operand is fetched from even and odd memory locations Y and Y+1. The operand must be a standard descriptor with type T = 1 or 3. The following checks are performed on the descriptor:

a. For T = 1, flag bits 20, 21, 27, and 28 = 1 and flag bits 25 and 26 = 0.

b. For T = 3, flag bits 20 and 21 = 1.

If these conditions are met, the descriptor is loaded into the safe store register (SSR). The lower three bits of the SSR base are forcibly set to zero. If one or more of the above conditions is not satisfied, the instruction is terminated and an IPR fault is generated. In this case, the SSR remains unchanged.

Each successful execution of LDSS initializes the 2-bit stack control register (SCR) as follows. (The SCR is associated with the SSR and contains a code that denotes the size of the last frame on the stack.)

If $C(Y+1)_{34,35}$ = 00/01/11, then 11 --> C(SCR)
(size of save store frame = 64 words)

If $C(Y+1)_{34,35}$ = 10, then 10 --> C(SCR)
(size of save store frame = 80 words)

(Refer to Safe Store Stack in discussion of CLIMB instruction.)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Any of the following conditions causes an IPR fault:
   a. Illegal address modification
   b. Illegal repeats
   c. If T is not equal to 1 nor 3.
   d. If either the flag bit or the base checks fail.
2. If the processor is not in Master or Slave mode, the execution of this instruction causes a Command fault.

EXAMPLE:

```
1       8       16              32
FANY    STZ     .SVFLT,,P.SSA
        LDX0    .ST2CS,,P.SSA
        TZE     NEPRA           Not type 2 critical
        STSS    .STEMP+6,,P.SSA
        LDAQ    SSRXX
        ADLAQ   .STEMP+6,,P.SSA backup safe store to prior frame
        STAQ    .STEMP+6,,P.SSA
        LDSS    .STEMP+6,,P.SSA
        LDP     P0,.SSR,DL
        LDX0    =0377001,DU
        STX0    .WREGS,,P0
        TRA     RETOUT
```

LDT	Load Timer Register	637 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: $C(Y)_{0-26}$ --> C(TR); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The use of this instruction in the Master or Slave mode causes a Command fault.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

LDWS	Load Working Space Registers	772 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: When EA_{17} (NS Mode) or EA_{33} (ES Mode) = 0

$C(Y)_{0-8}$ --> C(WSR0)

$C(Y)_{9-17}$ --> C(WSR1)

$C(Y)_{18-26}$ --> C(WSR2)

$C(Y)_{27-35}$ --> C(WSR3)

When EA_{17} (NS Mode) or EA_{33} (ES Mode) = 1

$C(Y)_{0-8}$ --> C(WSR4)

$C(Y)_{9-17}$ --> C(WSR5)

$C(Y)_{18-26}$ --> C(WSR6)

$C(Y)_{27-35}$ --> C(WSR7)

EXPLANATION: The contents of memory location Y replace the contents of working space registers (WSRs) 0, 1, 2, and 3 or WSR 4, 5, 6, and 7 based on the value of bit 17 (NS mode) or 33 (ES mode) of the effective address.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if illegal address modification or illegal repeats are used.

2. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

3. If the LDWS instruction is used to change the contents of the WSR that is currently the WSR for the instruction segment, then the LDWS must be followed immediately by a TRA *+1 to ensure that the new contents of the WSR take effect immediately.

EXAMPLE:

```
1       8       16              32
        EVEN
WS03    VFD     9/001, 9/001, 9/013, 9/27
WS47    VFD     9/45, 9/45, 9/63, 9/510
        .
        .
        .
        LDWS    WS03                    Load WSR 0-3 from EVEN word
        LDWS    WS47                    Load WSR 4-7 from Odd word
```

LDXn	Load Index Register n from Upper	22n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,...,7 as determined by op code

$C(Y)_{0-17}$ --> C(Xn); C(Y) unchanged

ES Mode

For n = 0,1,...,7 as determined by op code

$C(Y)_{0-35}$ --> C(GXn); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, or RPL of LDX0

INDICATORS: Zero - If C(Xn/GXn) = 0, then ON; otherwise, OFF

Negative - If $C(X_n/GX_n)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. DL modification executes with all zeros for data in the NS mode.
2. An Illegal Procedure fault occurs if illegal address modification or illegal repeats are used.

LIMR	Load Interrupt Mask Register	553 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY:

$C(A)_{0-7}$ --> Port interrupt level masks; 1 enables interrrupts

$C(A)_8$ --> All mask, conditionally as described in Explanation below; 1 enables interrupts

$C(A)_9$ --> Port connect mask; 1 enables connects

$C(A)_{10}$ --> All mask load control

C(A), C(Y) --> Unchanged

EXPLANATION: The SCU is selected by the control SCU bit. (Refer to SCU configuration register in Section 4.)

The operation of the All mask control is as follows:

$C(A)_{10}$	$C(A)_8$	
X	1	1 --> All mask
0	0	All mask is unchanged
1	0	0 --> All mask

The effective address (Y) is not used by the LIMR instruction.

Only masks associated with the issuing port are loaded.

The all mask bit is a single mask, associated with all ports. All masks are set to enable interrupts and connects at initialization.

Processor behavior on the next and all other instructions following the execution of LIMR is consistent with the mask setting indicated by the LIMR.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Prior to executing this instruction, the SCU must be "selected" by using the LCPR instruction to set or reset bit 22 in the CPU mode register.
2. The use of this instruction in other than Privileged Master mode causes an IPR fault.
3. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

LLR	Long Left Rotate	777 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: Rotate C(AQ) left by the number of positions indicated by bits 11-17 of Y (Y modulo 128) (NS mode) or Y_{27-33} (ES mode). Enter each bit leaving bit position 0 of AQ in bit position 71 of AQ.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. The rotate count comes from the value of Y. To "right-rotate" n bits, use LLR 72-n.

2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

LLS	Long Left Shift	737 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: Shift C(AQ) left by the number of positions indicated by bits 11-17 of Y (Y modulo 128) (NS mode) or Y_{27-33} (ES mode); fill vacated positions with zeros.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

Carry - If bit 0 of C(AQ) changes during the shift, then ON; otherwise OFF.

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

LPDBR	Load Page Table Directory Base Register	171 (1))

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: $C(Y)_{0-18}$ --> C(PDBR) (Mod 512)

EXPLANATION: The contents of bits 0-18 of Y are loaded into the 19-bit PDBR . Associative memory (AM) is cleared, if it is enabled, and C(Y) is unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs when illegal address modifications or illegal repeats are used.
2. If the processor is in Master or Slave mode, the execution of this instruction causes a Command fault.

LPL	Load Pointers and Lengths	467 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        LPL     LOCSYM,R,AM
```

OPERATING MODES: Any

SUMMARY: C(Y), C(Y+1),... ,C(Y+5) --> C(P&L)

C(Y+6), C(Y+7) --> C(LOR)

EXPLANATION: Pointer and length storage (P&L) is used by hardware to store control information to continue execution after an interruptible multiword instruction has been interrupted during execution. The low operand register (LOR) is a register used with quadruple-precision instructions.

The location of Y must be a multiple of 8. A fault does not occur when the lower 3 bits of Y are not 000. For purposes of execution, the hardware forces these bits to 000 (modulo 8).

In the LPL instruction, the contents of the eight words beginning at location Y are stored into scratch pad memory. (Refer to SPL for a description of the contents of these words.)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, RI, IR, IT

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used. The contents of the pointer and lengths storage is changed when the illegal execution of RPT, RPD, RPL, XEC, XED, and indirect modification IT occurs.

2. The pointer and length storage is used to recover from an interrupt or a Missing Page fault. Because the content depends upon hardware, the software must not change the contents of the pointer and lengths storage.

LREG	Load Registers	073 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

```
Bits  0-17 of C(Y)   --> C(X0)
Bits 18-35 of C(Y)   --> C(X1)
Bits  0-17 of C(Y+1) --> C(X2)
Bits 18-35 of C(Y+1) --> C(X3)
Bits  0-17 of C(Y+2) --> C(X4)
Bits 18-35 of C(Y+2) --> C(X5)
Bits  0-17 of C(Y+3) --> C(X6)
Bits 18-35 of C(Y+3) --> C(X7)
Bits  0-35 of C(Y+4) --> C(A)
Bits  0-35 of C(Y+5) --> C(Q)
Bits  0-7  of C(Y+6) --> C(E)
```

ES Mode

```
C(Y)   --> C(X0)
C(Y+1) --> C(X1)
C(Y+2) --> C(X2)
C(Y+3) --> C(X3)
C(Y+4) --> C(X4)
C(Y+5) --> C(X5)
C(Y+6) --> C(X6)
C(Y+7) --> C(X7)
C(Y+8) --> C(A)
C(Y+9) --> C(Q)
Bits  0-7 of C(Y+10) --> C(E)
```

EXPLANATION: Memory (location Y) is accessed on a double-word boundary by setting the lower three bits of the effective address Y to zero, adding a base address to it, and truncating the least-significant word address bit.

ILLEGAL ADDRESS MODIFICATIONS:	DU, DL, CI, SC, SCR
ILLEGAL REPEATS:	RPT, RPD, RPL
INDICATORS:	None affected
NOTE:	An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

LRL	Long Right Logical Shift	773 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: NS Mode

Shift C(AQ) right by the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with zeros.

ES Mode

Shift C(AQ) right by the number of positions indicated by bits 27-33 of Y (Y modulo 128); fill vacated positions with zeros.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

LRMB	Load Reserve Memory Base	712 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(Y) --> C(RMBR)

EXPLANATION: This instruction places the contents of the effective address into the Reserve Memory Base Register (RMBR). The RMBR is located in the PATROL half of processor scratch pad memory at location 73. Initialization firmware sets RMBR to zero. GCOS software sets the RMBR to zero when the processor is released as required by the CAMP instruction. (Refer to RMBR in Section 4.)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPD, RPL, and RPT

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if execution is attempted in Master or Slave mode.

2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

LRS	Long Right Shift	733 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

Shift C(AQ) right by the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with the content of bit 0 of C(AQ).

ES Mode

Shift C(AQ) right by the number of positions indicated by bits 27-35 of Y (Y modulo 128); fill vacated positions with the content of bit 0 of (AQ).

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

LXLn	Load Index Register n from Lower	72n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,...,7 as determined by op code

$C(Y)_{18-35}$ --> C(Xn); C(Y) unchanged

ES Mode

For n = 0,1....,7 as determined by op code

$C(Y)_{18-35}$ with sign extended --> C(GXn); C(Y) unchanged

Bit 18 of C(Y) is extended to bits 0-17 and loaded into GXn.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, or RPL of LXL0

INDICATORS: Zero - If C(Xn/GXn) = 0, then ON; otherwise, OFF

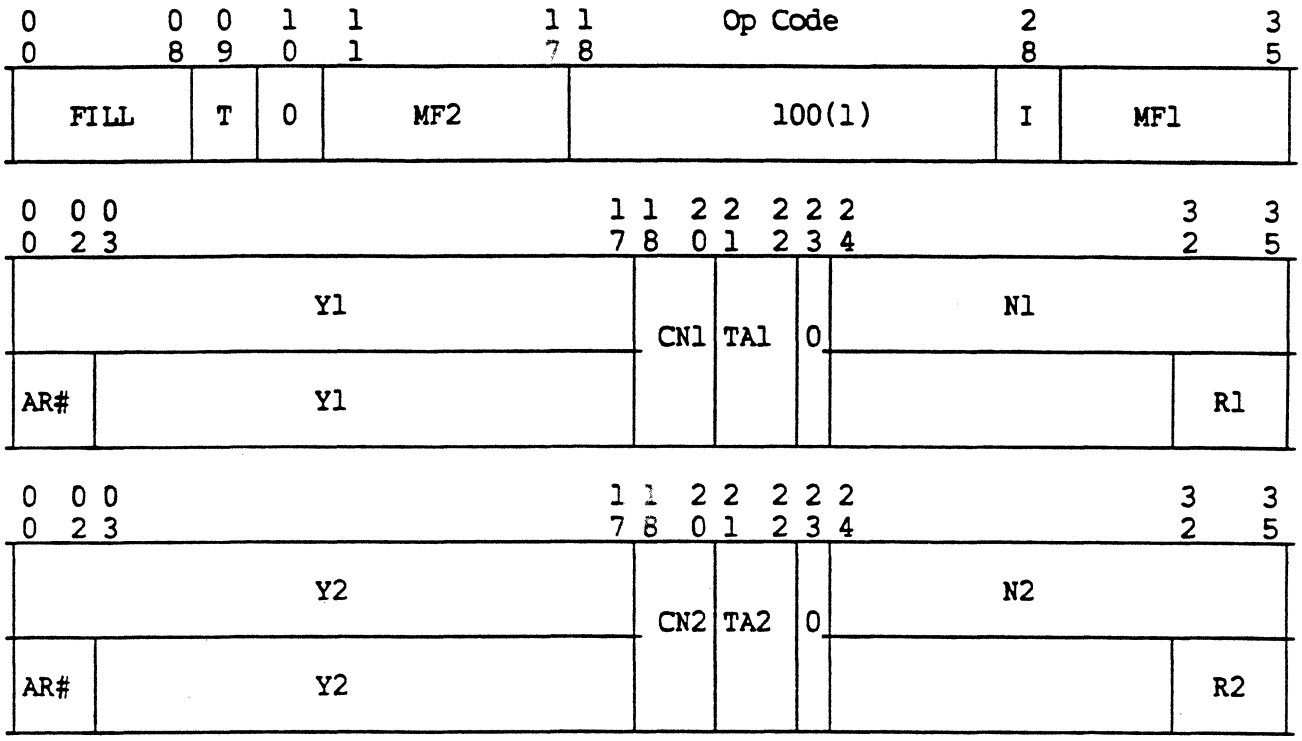
Negative - If $C(Xn/GXn)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. DU modification executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

MLR	Move Alphanumeric Left to Right	100 (1)

FORMAT:



CODING FORMAT: The MLR instruction is coded as follows:

```
1       8       16
-------------------------------------------

        MLR     (MF1),(MF2),FILL,T
        ADSCn   LOCSYM,CN,N,AM
        ADSCn   LOCSYM,CN,N,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) --> C(string 2)

EXPLANATION: Starting at location YC1, the alphanumeric characters of data type TA1 of string 1 replace, from left to right, the alphanumeric characters of data type TA2 of string 2 that starts at location YC2. If TA1 and TA2 differ, each character has high-order truncation or zero-fill, as appropriate.

If L1 is greater than L2, the least significant (L1-L2) characters are not moved and the Truncation indicator is set. If L1 is less than L2, bits 0-8, 3-8, or 5-8 of the FILL character (depending on TA2) are inserted as the least significant (L2-L1) characters. If L1 is less than L2, bit 0 of C(FILL) = 1, TA1 = 01, and TA2 = 10 (6-4 move); the hardware looks for a 6-bit overpunched sign. If a negative overpunch sign is found, a negative sign (octal 15) is inserted as the last FILL character. If a negative overpunch sign is not found, a positive sign (octal 14) is inserted as the last FILL character.

L2 = 0 does not necessarily mean that the instruction functions as a no-op, because the Truncation indicator may be affected.

The contents of string 1 remain unchanged except in cases of string overlap.

MF1 and MF2 (Multiword Modification Fields) are 7-bit fields specifying address modifications to be performed on the operand descriptors. They are broken into four subfields represented as (bit1, bit2, bit3, Index-register) in the instruction. They may be coded as follows:

If bit1 = 0	No address register is used
bit1 = 1	The address register is defined in the operand descriptor address field (e.g., ADSC9 ,,,AR)
If bit2 = 0	Operand length is specified in the N field of the operand descriptor (e.g., ADSC6 ,,24,)
bit2 = 1	Operand length is contained in the register specified by the code in the N field of the operand descriptor (e.g., ADSC4 ,,X4,)

MLR

If bit3 = 0	The operand descriptor follows the instruction word in its memory location.
bit3 = 1	The operand descriptor location following the instruction in memory points to the operand descriptor.
Index-register	The address modification register is defined as 0, 1, 2, 3, 4, 5, 6, 7, AU, QU, A, or Q.

See "Multiword Modification Field" and "Alphanumeric Operand Descriptors" in Section 5, and "Alphanumeric Instructions" under "Multiword Instructions" in Section 7 for additional information.

For speed, the MLR and MRL instructions operate on four double-words at a time. This mode of operation does not cause a problem when moving between either nonoverlapped strings or between any normal combination of any length overlapped strings. (In the latter case, software must choose between MLR and MRL to ensure that the overlapped sending characters are moved before they are moved into because they are also receiving characters.) This mode of operation can cause a problem when MLR or MRL is used to replicate a pattern across a string.

For example, one procedure used to replicate a pattern of K characters across a string of L characters is to

- store the K characters into character positions 1 through K of the string
- move" a string of length L - K and starting position 1 to the same length string starting at position K + 1. In this way, the last L - K sending characters are created "on the fly".

The mode of operating on four double-words at a time does not allow this creation "on the fly" for K less than four double-words of characters (when K starts on a word boundary or is less than eight double-words of characters and does not start on a word boundary).

To replicate a pattern between two characters and four double-words of characters, additional instructions must be used to initialize the first four double-words of the string of L characters. To replicate a 1-character pattern (most common application), a simple move with fill from a zero-length string can be used. (See examples below.)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Truncation - If L1 is > L2, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2 or if illegal repeats are used. A Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is a 1. A fault does not occur even when $L_2 = 0$. $L_2 = 0$ does not mean NOP; the truncation indicator may be affected.

EXAMPLES:

```
1       8       16              32
-------------------------------------------------------------

        MLR     ,,20            move with blank fill
        ADSC6   FLD1,,12        sending descriptor
        ADSC6   FLD2,4,14       receiving descriptor
        USE     CONST.          memory contents
FLD1    BCI     2,ABCDEFGHIJKL
FLD2    BSS     3               xxxxABCDEFGHIJKLØØ  (Result)
        USE

        MLR     ,,400           move with sign captured
        ADSC6   FLD1,3,9        sending descriptor
        ADSC4   FLD2,6,10       receiving descriptor
        USE     CONST.

FLD1    BCI     2,ØØØ12345678R
FLD2    BSS     2               xxxxxx123456789-  (Result)
        USE

1       8       16              32
-------------------------------------------------------------

        MLR     (1,0,0,),(,,,QU) move 24 words from P.IOQ to A+QU bytes.
        ADSC9   0,0,24,P.IOQ
        ADSC9   A,,24
```

MME	Master Mode Entry Fault	001 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: This instruction generates a MME fault which causes the processor to switch to Privileged Master mode and to execute an Inward CLIMB using the entry descriptor from the word pair in memory locations 32 and 33 (octal).

If the safe store bypass flag in the option register = 1, a safe store frame is generated. The size of this safestore frame is determined by the type of the entry descriptor. A code of 00010 in bits 12-16 of word 5 in the safe store frame indicates the occurrence of the MME fault.

The wired-in CLIMB instruction functions as though the second word of the CLIMB instruction had the following characteristics:

E = 0	No parameters
C_{18}	Do not load X0
C_{19}	No effect. Turn Master mode indicator ON.
C_{22-23}	= 0 Inward CLIMB
S, D	No effect

The entry descriptor specifies a descriptor to be obtained from the linkage segment for loading into the instruction segment register (ISR). The entry descriptor also specifies the value to be loaded into the instruction counter (ID).

The processor is placed in Privileged Master mode for the execution of the wired-in CLIMB. Upon completion of the CLIMB, the processor remains in Privileged Master mode if flag bit 26 of the new ISR = 1 (privileged); otherwise, the processor changes to Master mode.

ILLEGAL ADDRESS MODIFICATIONS: Not executed. CI, SC, SCR generate an illegal condition that causes the history registers to be locked if mode register bit 31 = 1. No IPR fault occurs because the MME fault has higher priority.

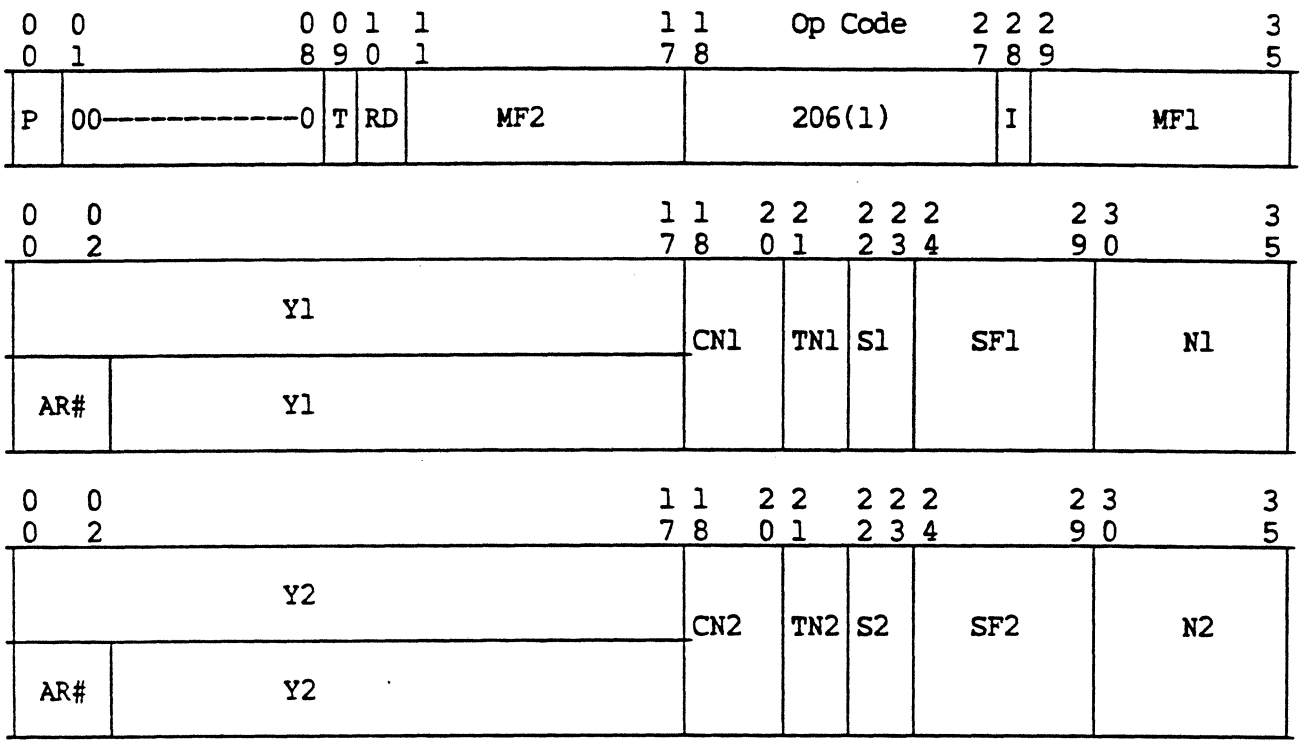
ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Master mode - ON

NOTE: An IPR fault occurs if an illegal repeat is used.

MP2D	Multiply Using Two Decimal Operands	206 (1)

FORMAT:



CODING FORMAT: The MP2D instruction is coded as follows:

```
1       8       16
_____________________________________

        MP2D    (MF1),(MF2),RD,P,T
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) * C(string 1) --> C(string 2)

EXPLANATION: Same as for MP3D except that the product is stored using YC2, TN2, S2 and, if S2 indicates a scaled format, SF2.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero	-	If result equals zero, then ON; otherwise, OFF
Negative	-	If result is negative, then ON; otherwise, OFF
Truncation	-	If, in the preparation of the final result, one or more least significant digits (zero or nonzero) are lost and rounding is not specified, then ON. Otherwise (i.e., no least significant digits lost or rounding is specified), OFF
Exponent Overflow	-	If exponent of floating-point result is > 127, then ON; otherwise, unchanged
Exponent Underflow	-	If exponent of floating-point result is < -128, then ON; otherwise, unchanged
Overflow	-	If data is lost in most significant positions then ON; otherwise, unchanged

NOTES:

1. A Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is a 1.

2. An Illegal Procedure fault occurs if:

   a. Illegal address modification is specified for MF1 or MF2, or illegal repeats are used.

   b. Any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.

The values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.

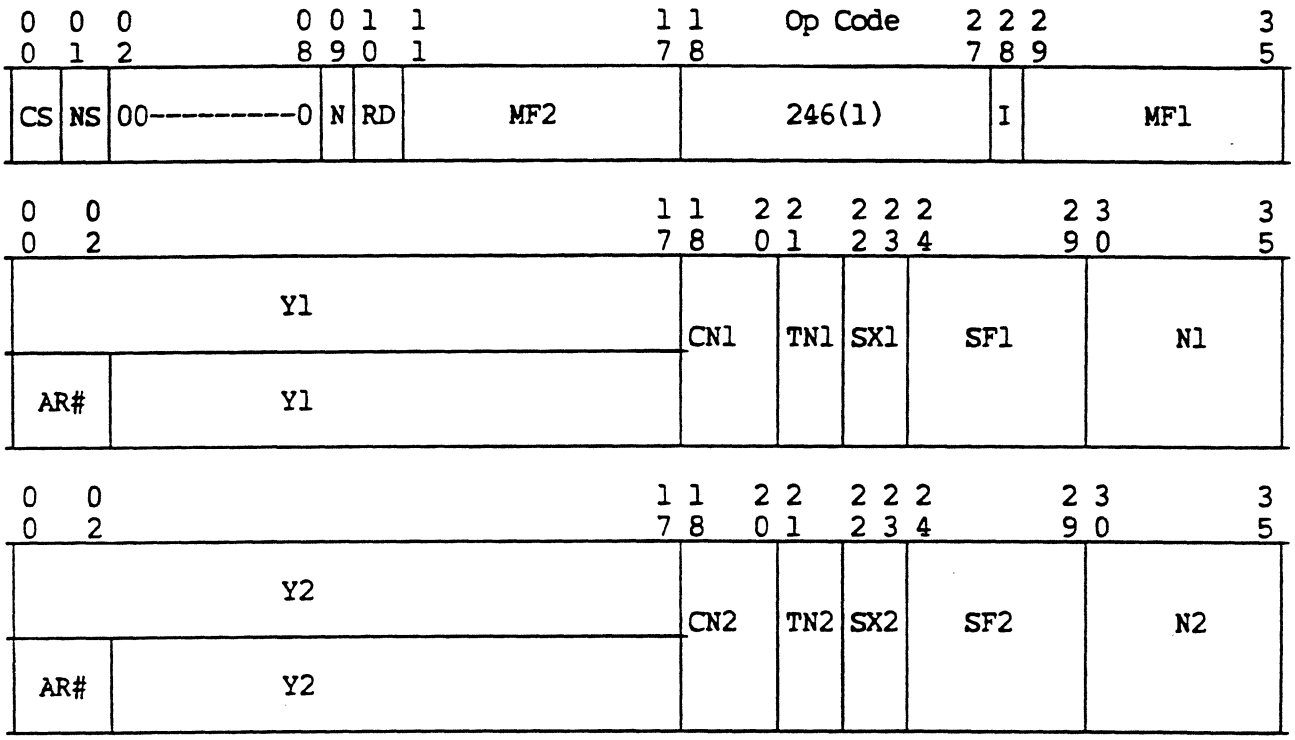
3. If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.

EXAMPLES:

```
1       8       16              32
        MP2D    ,,1,1           rounding and plus sign options
        NDSC9   FLD1,0,4,2,-3   multiplier operand descriptor
        NDSC4   FLD2,0,8,1,-2   multiplicand operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    4A2+            0 0 2 +
FLD2    EDEC    8P+1234567      +1234567
        USE                     +0002469  (Product)
*                               indicators on?   none

        MP2D    ,,1             rounding option
        NDSC4   FLD1,0,8,3,-2   multiplier operand descriptor
        NDSC4   FLD2,0,8        multiplicand operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    8P10            00000010
FLD2    EDEC    8P+123.45       +12345-2
        USE                     +12345-3   (Product)
*                               indicators on?   none
```

MP2DX	Multiply Using Two Decimal Operands Extended	246 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
MP2DX  (MF1),(MF2),RD,CS,T,NS
NDSCn  LOCSYM,CN,N,SX,SF,AM
NDSCn  LOCSYM,CN,N,SX,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) * C(string 1) --> C(string 2)

EXPLANATION: Same as for MP3DX except that the product is stored using YC2, TN2, SX2 and, if SX2 indicates a scaled format, SF2.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or MF2

ILLEGAL REPEATS: RPT, RPD, RPL

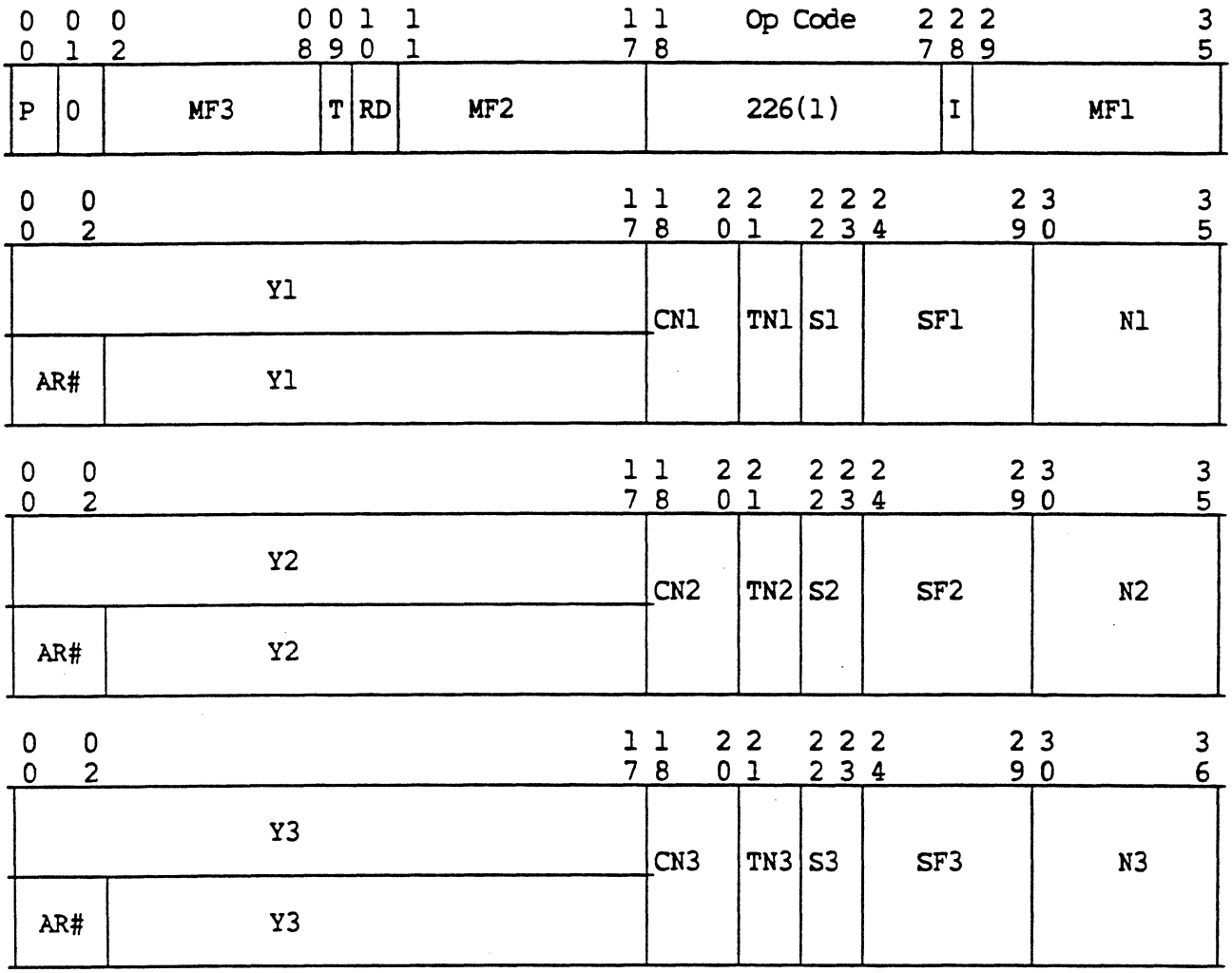
INDICATORS: Same as for MP2D.

NOTES:

1. Notes of MP3D apply.
2. See MVNX for information about coding of overpunched signs.

MP3D	Multiply Using Three Decimal Operands	226 (1)

FORMAT:



CODING FORMAT: The MP3D instruction is coded as follows:

```
1       8       16
-------------------------------------------

        MP3D    (MF1),(MF2),(MF3),RD,P,T
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) * C(string 1) --> C(string 3)

EXPLANATION: The decimal number of data type TN2, sign and decimal type S2, and starting location YC2, is multiplied by the decimal number of data type TN1, sign and decimal type S1, and starting location YC1. The product is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type S3.

If S3 indicates a fixed-point format, the results are stored using SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most-significant digit overflow or least-significant digit truncation to occur.

If S3 indicates a floating-point format, the result is right-justified to preserve the most significant nonzero digits even if this causes least significant truncation. In this case, the most-significant-digit of the mantissa (except for the sign digit) is set to a number digit other than 0.

If P=1, positive signed 4-bit results are stored using octal 13 as the plus sign. If P=0, positive signed 4-bit results are stored with octal 14 as the plus sign. If RD is a 1, rounding takes place prior to storage.

Provided that string 1, string 2, and string 3 are not overlapped, the contents of strings 1 and 2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Zero - If result equals zero, then ON; otherwise, OFF

Negative - If result is negative, then ON; otherwise, OFF

MP3D

Truncation - If, in the preparation of the final result, one or more least-significant digits (zero or nonzero) are lost and rounding is not specified, then ON; otherwise (i.e., no least-significant digits lost or rounding specified), OFF

Exponent Overflow - If exponent of floating-point result is > 127, then ON; otherwise, unchanged

Exponent Underflow - If exponent of floating-point result is < -128, then ON; otherwise, unchanged

Overflow - If data is lost in most-significant positions, then ON; otherwise, unchanged

NOTES:

1. A Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is a 1.

2. An Illegal Procedure fault occurs if:

   a. DU or DL modification is specified for MF1, MF2, or MF3, or if illegal repeats are used.

   b. Any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.

   c. The values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.

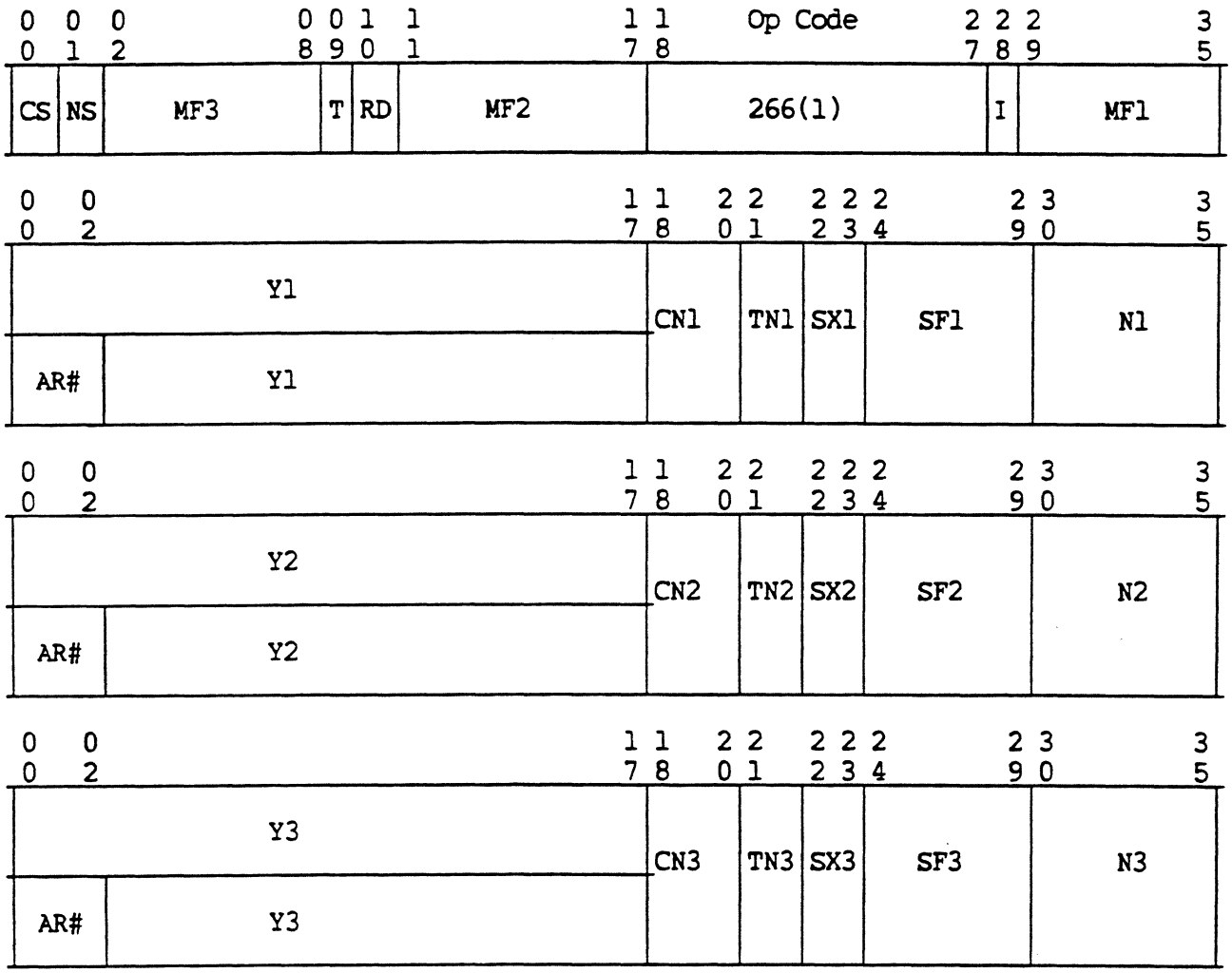
3. If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.

EXAMPLES:

```
1       8       16              32
        MP3D    ,,,1            with rounding option
        NDSC4   FLD1,6,2,2      multiplier operand descriptor
        NDSC4   FLD2,0,8,1,-3   multiplicand operand descriptor
        NDSC9   FLD3,1,7,1,-2   product operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    8P5+            0000005+
FLD2    EDEC    8P+1234567      +1234567
FLD3    BSS     2               +617284   (Product)
        USE                     indicators on?   none

        MP3D    ,,,,1
        NDSC4   FLD1,0,2,3,-2   multiplier operand descriptor
        NDSC4   FLD2,0,8,1,-3   multiplicand operand descriptor
        NDSC4   FLD3,1,7        product operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    2PL25           25000000
FLD2    EDEC    8P-1234567      -1234567
FDL3    EDEC    8P+0            +-3086-1   (Product)
        USE                     instruction fault?   no
*                               indicators on?  truncation and negative
```

MP3DX	Multiply Using Three Decimal Operands Extended	266 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
MP3DX   (MF1),(MF2),(MF3),RD,CS,T,NS
NDSCn   LOCSYM,CN,N,SX,SF,AM
NDSCn   LOCSYM,CN,N,SX,SF,AM
NDSCn   LOCSYM,CN,N,SX,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) * C(string 1) --> C(string 3)

EXPLANATION: The decimal number of data type TN2, sign and decimal type S2, and starting location YC2, is multiplied by the decimal number of data type TN1, sign and decimal type S1, and starting location YC1. The product is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type S3.

If SX3 indicates a fixed-point format, the results are stored using SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most-significant-digit overflow or least-significant-digit truncation to occur.

If SX3 indicates a floating-point format, the result is right-justified to preserve the most-significant-nonzero digits even if this causes least-significant truncation. In this case, the most-significant digit of the mantissa (except for the sign digit) is set to a number digit other than 0.

The character set is defined by CS. Placement of over punched sign in the output is controlled by NS. (Refer to introductory pages of this section for definition of NS.) If RD is 1, rounding takes place prior to storage.

Provided that string 1, string 2, and string 3 are not overlapped, the contents of strings 1 and 2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Same as for MP3D.

NOTES:

1. Notes of MP3D apply.
2. See MVNX for information about coding of overpunched signs.

MPF	Multiply Fraction	401 (0)

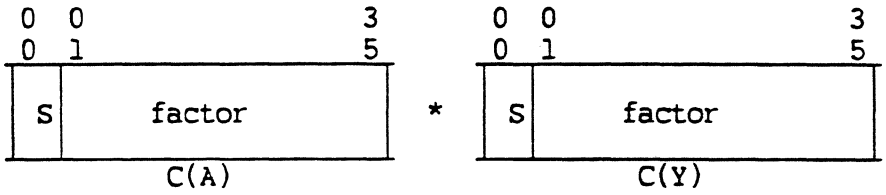
FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

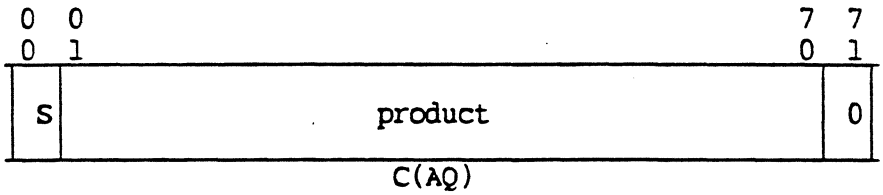
SUMMARY: C(A) * C(Y) --> C(AQ), left justified; C(Y) unchanged

EXPLANATION: This instruction multiplies two 36-bit fractional factors (including sign) to form a 71-bit fractional product (including sign). The product is stored in AQ, left-justified. Bit 71 of C(AQ) is filled with a zero bit.

Overflow can occur only when A and Y both = -1 and the result exceeds the range of the AQ-register.



yielding:



ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

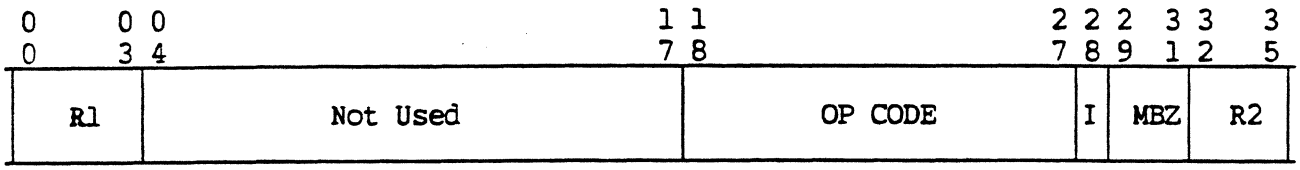
Negative - If bit 0 of C(AQ) = 1, then ON; otherwise, OFF

Overflow - If range of AQ is exceeded, then ON

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

MPRR	Multiply Register Pair by Register	530 (1)

FORMAT:



CODING FORMAT:

```
1       8       16
        MPRR    R1,,R2
```

OPERATING MODES: Executes only in ES mode.

SUMMARY:

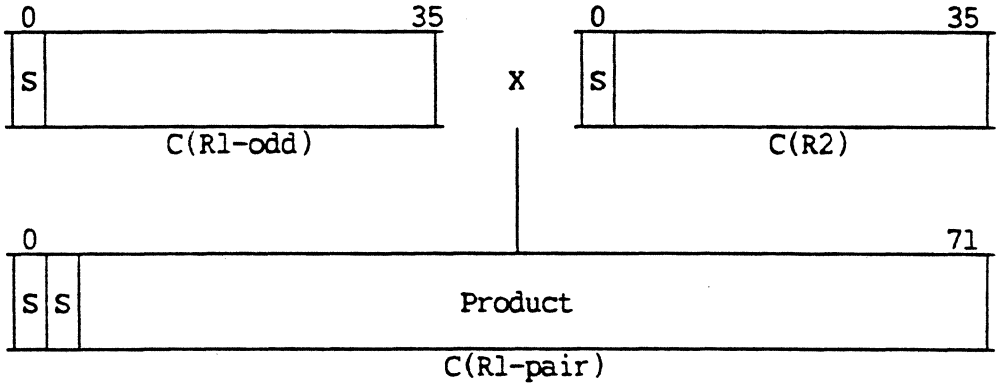
for R1-odd R1: 1, 3, 5, 7, Q
for R1-pair R1: 0, 2, 4, 6, A

C(R1-odd) x C(R2) --> C(R1-pair)

C(R2) unchanged

EXPLANATION

A register pair is specified in R1. The product of the content of the odd-numbered register (Q if A,Q specified) and that of R2 is taken and the result is loaded, right-justified into the R1-pair.



ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1-pair) = 0, then ON; otherwise, OFF

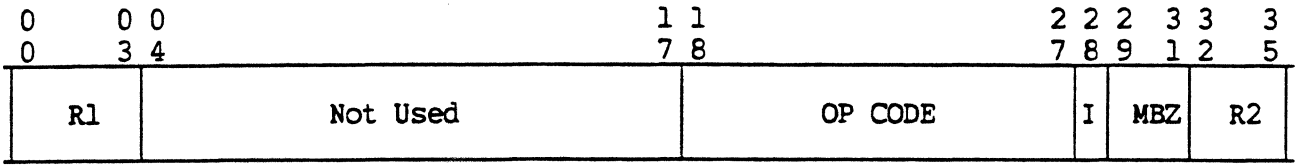
Negative - If $C(R1\text{-pair})_0 = 1$, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

MPRS	Multiply Single Register by Register	531 (1)

FORMAT:



CODING FORMAT: 1 8 16

MPRS R1,,R2

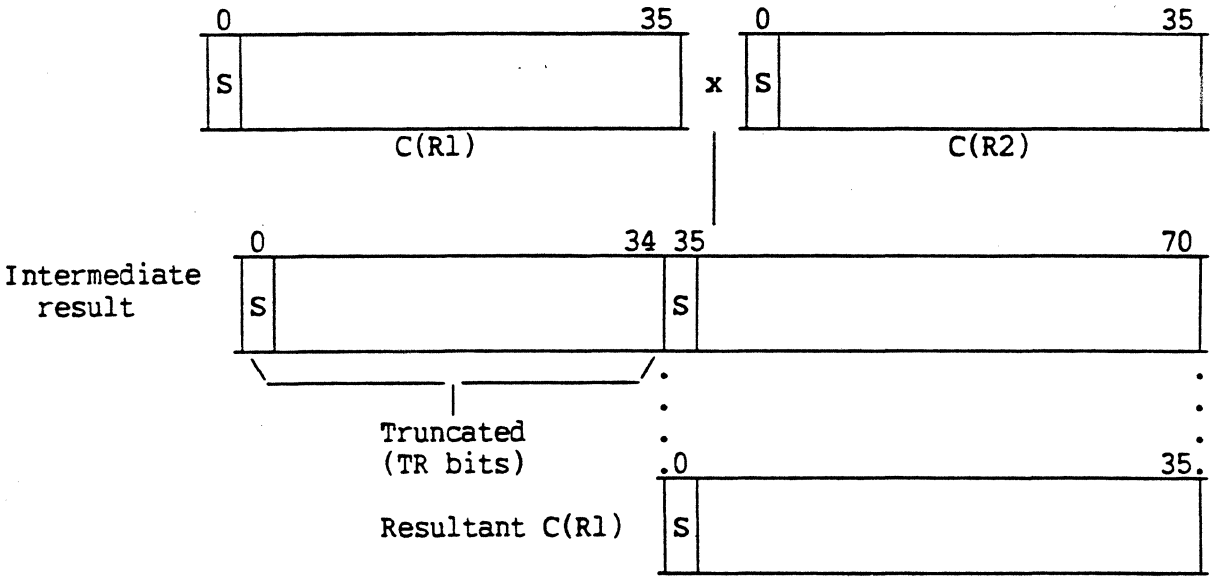
OPERATING MODES: Executes in ES mode only

SUMMARY: R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) x C(R2) --> C(R1)

C(R2) unchanged

EXPLANATION The product of the content of R1 and R2 is taken. The low-order 36 bits of the result are loaded into R1.



The multiplication is performed on the two's complement data to obtain 71-bit two's complement data as an intermediate result. The low-order 36 bits of this intermediate result are loaded into R1.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS:

Zero - If the intermediate result is 0, then ON; otherwise OFF

Negative - If the intermediate result)$_0$ is 1, then ON, otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.
3. No overflow check for the final result is performed; therefore, the Zero and Negative indicators are set by the state of the intermediate result.

MPX	Multiply GXn	04n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

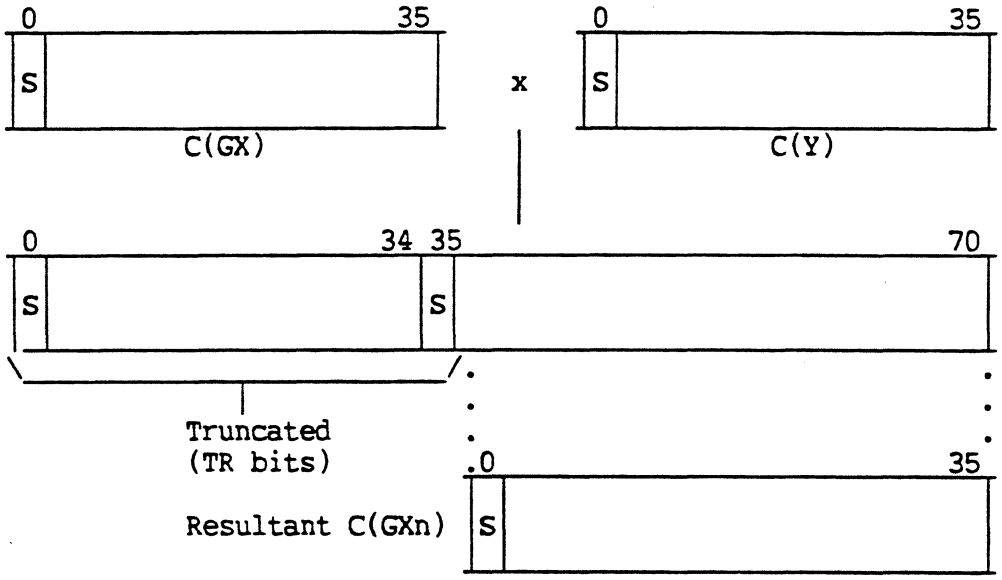
CODING FORMAT:

```
1       8       16
        MPX     n,Y,R,AM
```

OPERATING MODES: Executes in ES mode only

SUMMARY: C(GXn) x C(Y) --> GXn

EXPLANATION: The product of the content of GXn and that of the one word at memory location Y is taken. The low-order 36 bits of the result is loaded into GXn .



The multiplication is performed on the two's complement data to obtain 71-bit two's complement data as an intermediate result. The low-order 36 bits of this intermediate result are loaded into the GXn.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

ILLEGAL EXECUTES: If the instruction is executed in NS mode

INDICATORS:

Zero - If the intermediate result is 0, then ON; otherwise OFF.

Negative - If the (intermediate result)$_0$ is 1, then ON; otherwise OFF.

NOTES:

1. An IPR fault occurs if illegal address modification are used or if the instruction is executed in NS mode.

2. No overflow check for the final result is performed, therefore, the Zero and Negative indicators are set by the state of the intermediate result.

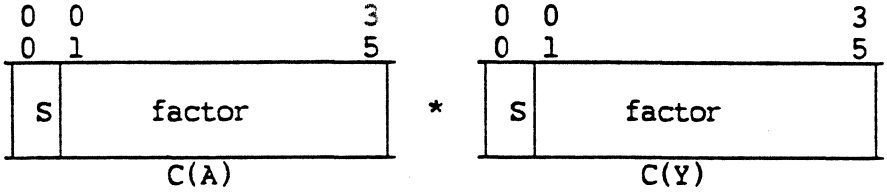
MPY	Multiply Integer	402 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

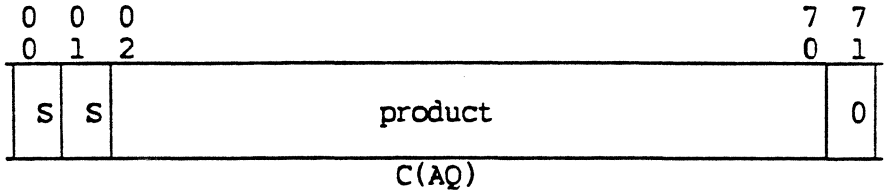
OPERATING MODES: Any

SUMMARY: C(Q) * C(Y) --> C(AQ), right justified; C(Y) unchanged

EXPLANATION: This instruction multiplies two 36-bit integral factors (including sign) to form a 71-bit integral product (including sign). The product is stored in AQ, right-justified. Bit 0 of C(AQ) is filled with an "extended sign" bit.



yielding:



When (-2**35) * (-2**35) = +2**70, bit 1 of AQ is used to represent the product rather than the sign and no overflow occurs.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

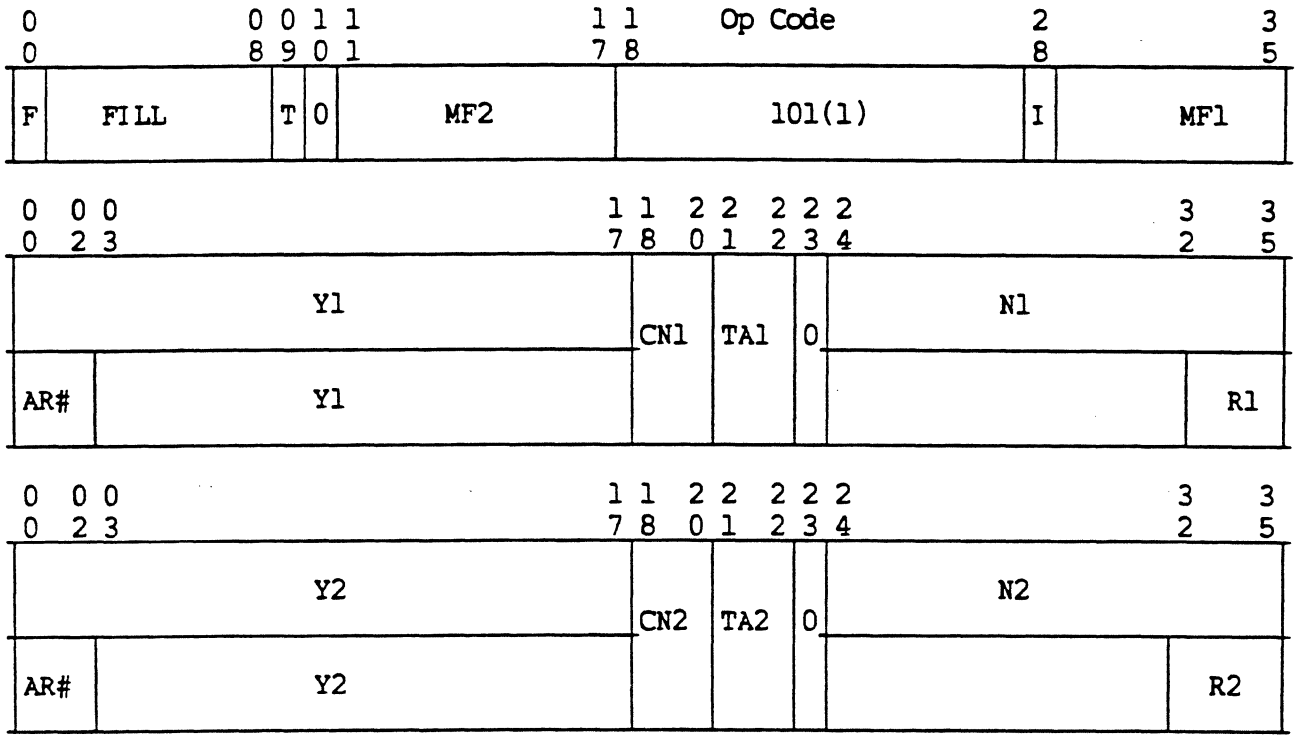
INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If bit 0 of C(AQ) = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

MRL	Move Alphanumeric Right to Left	101 (1)

FORMAT:



CODING FORMAT: The MRL instruction is coded as follows:

```
1       8       16
        MRL     (MF1),(MF2),FILL,T
        ADSCn   LOCSYM,CN,N,AM
        ADSCn   LOCSYM,CN,N,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) --> C(string 2)

EXPLANATION: This instruction is identical with MLR except that the starting locations are YC1 + (L1-1) and YC2 + (L2-1) and the movement is from right to left (from least-significant character toward most-significant character). Consequently, any truncation or fill is of the most-significant characters.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Truncation - If L1 is > L2, then ON; otherwise, OFF

NOTES:

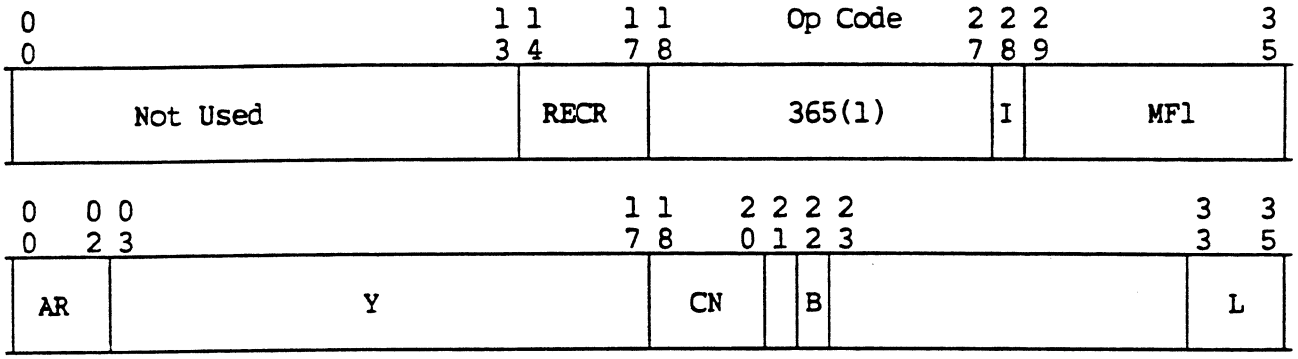
1. An Illegal Procedure fault occurs if illegal address modification or illegal repeats are used.
2. A Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is a 1.
3. Refer to Note 3 of the MLR instruction for information on string replication.
4. L2 = 0 does not necessarily mean that the instruction functions is a no-op because the truncation indicator may be affected.

EXAMPLE:

```
1       8       16              32
-------------------------------------------------------------------

        MRL     ,,20            move with blank fill
        ADSC6   FLD1,,12        sending descriptor
        ADSC6   FLD2,4,14       receiving descriptor
        USE     CONST.          memory contents
FLD1    BCI     2,ABCDEFGHIJKL
FLD2    BSS     3               xxxxØØABCDEFGHIJKL  (Result)
        USE

        MRL     ,,400           move with sign and fill
        ADSC6   FLD1,3,9        sending descriptor
        ADSC4   FLD2,4,12       receiving descriptor
        USE     CONST.          memory contents
FLD1    BCI     2,ØØØ12345678R
FLD2    BSS     2               xxxx-00123456789  (Result)
        USE
```

MTM	Move to Memory	365 (1)

FORMAT:



CODING FORMAT: The MTM instruction is coded as follows:

```
1       8       16
_____________________

        MTM     (MFl),RECR
        SDSCn   Y,CN,L,,AM
```

EXPLANATION: This instruction moves one, two, three, or four 9-bit characters into memory from the register specified in the RECR field of the instruction. MTM is the inverse of MTR.

The move from the register into memory is done from right to left beginning at YCN + (L-1). (L must be 0-4.)

The setting of the B field shown in the descriptor diagram above, is determined by the contents of the n in SDSCn. (A 9 in the n field sets B = 0; an 8 sets B = 1.) This setting determines the functions of the move operation as follows.

- o If B = 0 The 9-bit characters are fetched at once from the specified register and moved into memory without modification.

- If B = 1 — 8-bits (1 byte) are fetched from the specified register and 0 is concatenated to the most-significant bit position to form a 9-bit character. Then the character is moved to memory. Up to L characters can be moved.

An A, Q, or X0-X7, GX0-GX7 register may be specified in the RECR field.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL specified in MF

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Refer to "Character Move To/From Register Instructions" in Section 7 for a description of the fields in the operand descriptor (SDSC).

2. An IPR fault occurs under the following conditions.

   - If RECR specifies X0-X7 and L > 2. (X0-X7 can only hold 2 bytes.)
   - If RECR specifies A or Q or GX-GX7 and L > 4.
   - If illegal address modifications or illegal repeats are used.

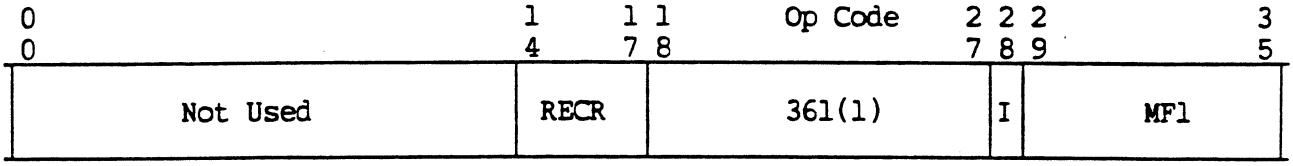
3. The RL bit of the MF field is ignored. The character length must be specified in the L field of the operand descriptor.

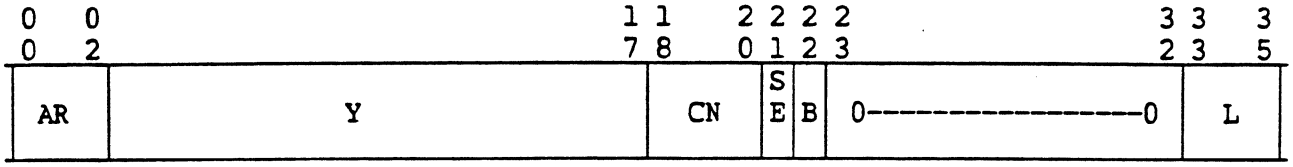
4. When L = 0, the MTM instruction functions as a NOP.

5. Refer to Explanation under the MTR instruction for the codes allowed in the RECR field.

MTR	Move to Register	361 (1)

FORMAT:





CODING FORMAT: The MTR instruction is coded as follows:

```
1       8       16
_______________________

        MTR     (MF1),RECR
        SDSCn   Y,CN,L,SE,AM
```

EXPLANATION: This instruction moves one, two, three, or four 9-bit characters from the memory location beginning at YCN + (L-1) to a register specified by the RECR field (bits 14-17) of the instruction word. MTR is the inverse of MTM.

The moved characters are right-justified in the specified register.

The setting of the B field shown in the descriptor diagram above, is determined by the contents of the n in SDSCn. (SDSC9 sets B = 0; SDSC8 sets B = 1.) The SE field is specified by the user. These settings determine the character positioning functions of the move operation as follows.

- o If B = 0 The 9-bit characters from memory are moved to the specified register without modification. If L is less than the character size capacity of the specified register, the vacant high-order character positions of the register are filled as follows.

MTR

SE = 0 The remaining character positions are filled with 0.

SE = 1 Bit 0 of the last character moved is regarded as a sign and the value of this bit is extended to fill the remaining character positions of the register.

o If B = 1 Bit 0 of each 9-bit character moved from memory is removed and the resulting 8-bit bytes are moved in a right-justified string into the specified register. The SE field affects the result of the move as follows.

SE = 0 The remaining bit positions of the specified register are filled with 0.

SE = 1 Bit 0 of the last 8-bit byte moved to the specified register is extended to fill the remaining high-order bits of the register.

An A, Q, or X0-X7, GX0-GX7 register may be specified in the RECR field. The code of these registers is the same as for the register code specified in the REG portion of the MF field. An invalid specification results in an IPR fault.

The RECR codes are displayed below.

RECR Code	Register (NS Mode)	(ES Mode)
0000	IPR	IPR
0001	A	A
0010	Q	Q
0011	IPR	IPR
0100	IPR	IPR
0101	IPR	IPR
0110	IPR	IPR
0111	IPR	IPR
1000	X0	GX0
1001	X1	GX1
1010	X2	GX2
1011	X3	GX3
1100	X4	GX4
1101	X5	GX5
1110	X6	GX6
1111	X7	GX7

The number of characters to be moved is specified in the L field of the operand descriptor.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL specified in MF

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero - ON if C(register) = 0; otherwise, OFF.

Negative - ON if bit 0 of C(register) = 1; otherwise, OFF.

NOTES:

1. Refer to "Character Move To/From Register Instructions" in Section 7 for a description of the fields in the operand descriptor (SDSC).

2. An IPR fault occurs under the following conditions.

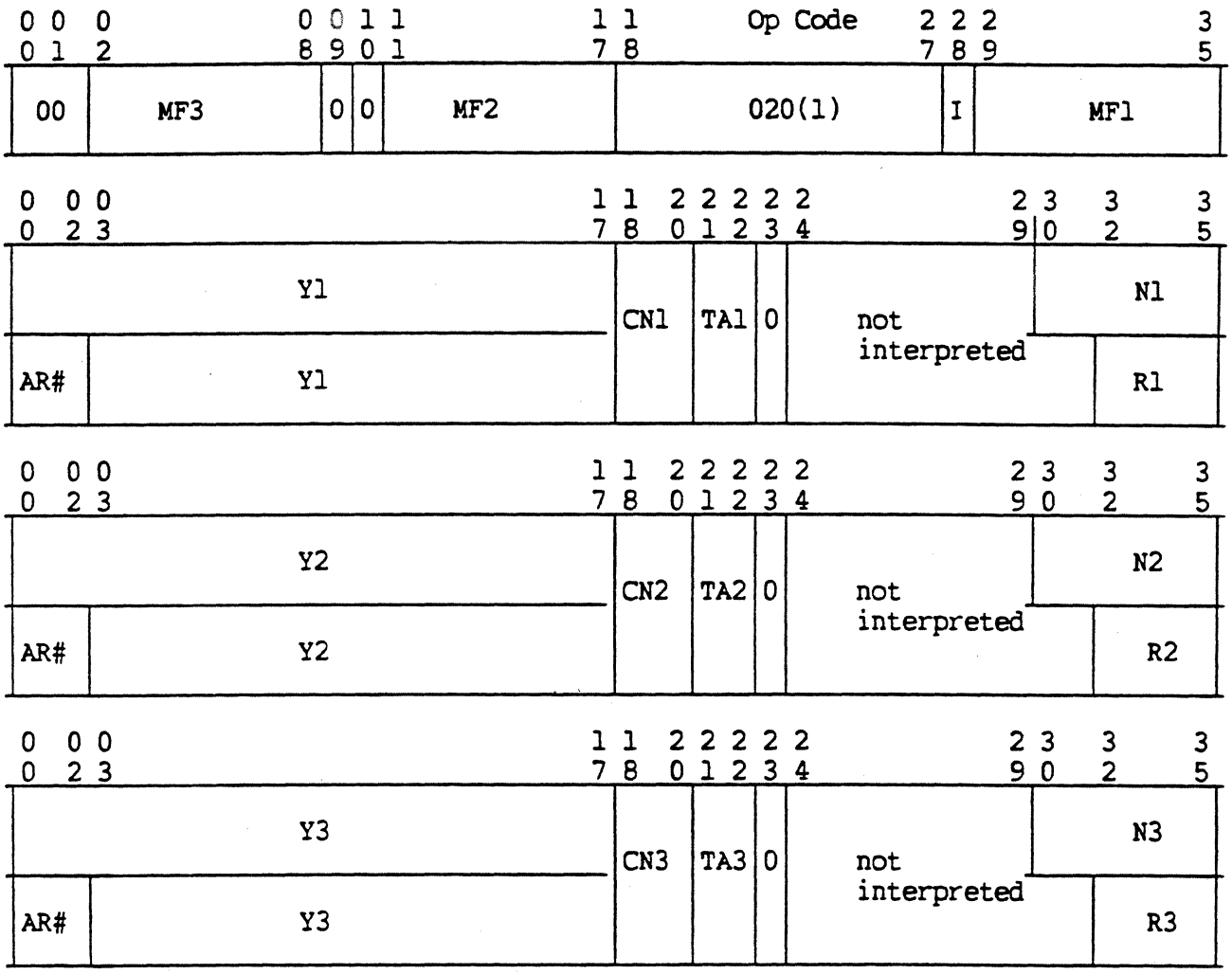
   - o If RECR specifies X0-X7 and L > 2. (X0-X7 can only hold 2 bytes.)
   - o If RECR specifies A or Q or GX-GX7 and L > 4.
   - o If illegal address modifications or illegal repeats are used.

2. The RL bit of the MF field is ignored. The character length must be specified in the L field of the operand descriptor.

3. If L = 0, the contents of the receiving register is set to 0, the Zero indicator to ON, and the Negative indicator to OFF.

MVE	Move Alphanumeric Edited	020 (1)

FORMAT:



CODING FORMAT: The MVE instruction is coded as follows:

```
1       8       16
-------------------------------------------

        MVE     (MF1),(MF2),(MF3)
        ADSCn   LOCSYM,CN,N,AM
        ADSC9   LOCSYM,CN,N,AM
        ADSCn   LOCSYM,CN,N,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY:

```
                    string 2 control
C(string 1) ----------------> C(string 3)
```

EXPLANATION: Starting at location YC1, the string of alphanumeric characters of data type TA1 is moved to the string of alphanumeric characters of data type TA3 starting at location YC3. The move is under control of the micro operation sequence of length L2 and type TA2 = 00 that starts at location YC2. (Refer to "Micro Operations" in this section.)

Maximum allowable length for L1, L2, and L3 is 63; they are not checked for length greater than 63. Only the rightmost six bits (30-35) are interpreted for length. Likewise, when a register is specified as containing the length, only the rightmost six bits of the register are interpreted.

The operation stops when L3 is exhausted.

The result is unpredictable when strings are overlapped.

The contents of the alphanumeric character string that starts at YC1 and the micro operation sequence that starts at YC2 remain unchanged.

On the processor, L3 = 0 is the normal termination; thus, at the start of the instruction, if L3 = 0 and there are no faults (see Note), no operation is performed and the instruction terminates normally, independently of whether L1 or L2 equals zero, because the hardware does not access these fields when L3 = 0.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE:

1. An Illegal Procedure fault occurs under the following conditions.
   - o If illegal address modification is used
   - o If illegal repeats are used.
   - o If an illegal mirco operation is executed.
   - o If TA2 is not = 0.
   - o If an attempt is made to access string 2 when $L_2 = 0$.
2. Refer to "Micro Operations for Edit Instructions" in Section 7.

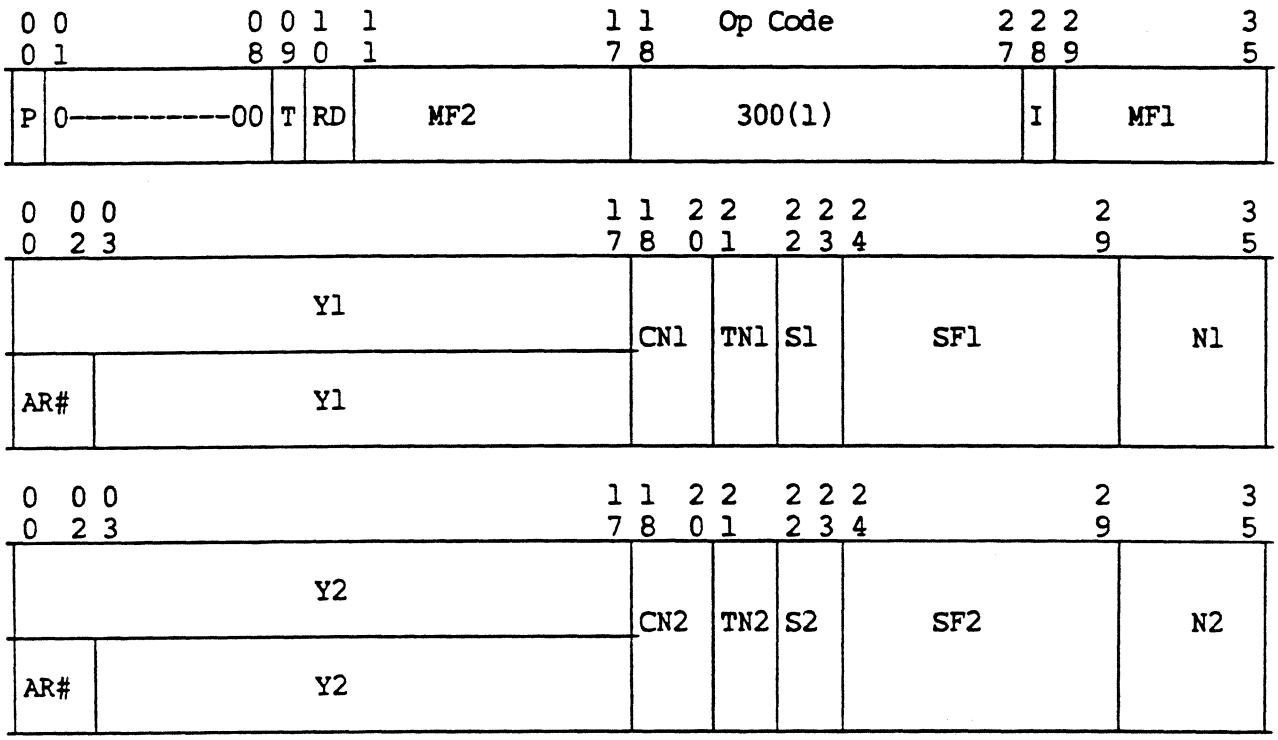
EXAMPLES:

```
1       8       16              32
-------------------------------------------------------------

        MVE                     move alphanumeric edited
        ADSC6   FLD1,2,20       sending field operand descriptor
        ADSC9   FLD2,0,25       micro-op string operand descriptor
        ADSC6   FLD3,0,30       receiving field operand descriptor
        USE     CONST.
FLD1    BCI     4,12SMITHROGERWILLIAMS25AB
FLD2    MICROP  (CHT,0),8H*,.-ØØØØ,(SES,8),(INSB,1),(INSB,5)
        MICROP  (MVC,10),(INSB,2),(INSB,5),(MVC,7)
        MICROP  (INSB,5),(MVC,1),(INSB,3),(INSB,5)
        MICROP  (INSB,4),(INSB,5),(INSB,0),1H#,(MCV,2)
*
* The following table explains the above micro-operation sequence

*       (CHT,0),8H*,.-ØØØØ - Change Edit Table to these 8 Hollerith
                             characters
*       (SES,8)  - Set End Suppression Flag ON
*       (INSB,1) - Insert Edit Table Entry #1 (*)
*       (INSB,5) - Insert Edit Table Entry #5 (Ø)
*       (MVC,10) - Move 10 characters from FLD1 (SMITHROGER)
*       (INSB,2) - Insert Edit Table Entry #2 (,)
*       (INSB,5) - Insert Edit Table Entry #5 (Ø)
*       (MVC,7)  - Move 7 characters from FLD1 (WILLIAM)
*       (INSB,5) - Insert Edit Table Entry #5 (Ø)
*       (MVC,1)  - Move 1 character from FLD1 (S)
*       (INSB,3) - Insert Edit Table Entry #3 (.)
*       (INSB,5) - Insert Edit Table Entry #5 (Ø)
*       (INSB,4) - Insert Edit Table Entry #4 (-)
*       (INSB,5) - Insert Edit Table Entry #5 (b)
*       (INSB,0),1H# - Insert specified character (#)
*       (MVC,2)  - Move 2 characters from FLD1 (25)
*                             memory contents in BCD characters
FLD     BSS     5             *ØSMITHROGER,ØWILLIAMbS.Ø-Ø#25
        USE

        MVE                     move alphanumeric edited
        ADSC9   FLD1,0,7        sending field operand descriptor
        ADSC9   FLD2,0,6        micro-op string operand descriptor
        ADSC9   FLD3+1,1,7      receiving field operand descriptor
        USE     CONST.
FLD1    ASCII   2,ERROR-2
FLD2    MICROP  (LTE,1),1A#,(MVC,5),(INSM,1),(IGN,1),(MVC,1)
*                             memory contents in ASCII characters
FLD3    ASCII   3,CODE          codeØerror#2     (Result)
```

```
1       8       16                  32
        MVE
        ADSC9   RDWRK,2,6
        ADSC9   MOPSC,0,11
        ADSC9   A9,1,7
        MVT
        ADSC9   A9,1,7
        ADSC9   A,1,7               NDSC9   A,1,7,2
        ARG     TABLE-12
        USE     CONST.
MOPSC   MICROP  (LTE,3),10000,(LTE,4),10100
        MICROP  (MSES,6),(LTE,3),1A+,(LTE,4),1A-,(SES),(ENF)
        OCT     000000000053,000055000000  05X
TABLE   OCT     060061062063,064065066067  06X
        OCT     070071000000,000000000000  07X
        OCT     000000000000,000000000000  10X
        OCT     000000061062,063064065066  11X
        OCT     067070071000,000000000000  12X
        OCT     000000000000,000000060000  13X
        OCT     000000000000,000000000000  14X
        OCT     000000061062,063064065066  15X
        OCT     067070071000,000000000000  16X
        OCT     000000000000,000000000000  17X
        USE
```

MVN	Move Numeric	300 (1)

FORMAT:



CODING FORMAT: The MVN instruction is coded as follows:

```
1       8       16
_____________________

        MVN     (MF1),(MF2),RD,P,T
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) --> C(string 2)

EXPLANATION: Starting at location YC1, the decimal number of data type TN1 and sign and decimal type S1 is moved, properly scaled, to the decimal number of data type TN2 and sign and decimal type S2 that starts at location YC2.

If S2 indicates a fixed-point format, the results are stored as L2 digits using scale factor SF2, and thereby may cause most-significant-digit overflow and/or least-significant-digit truncation.

If P = 1, positive signed 4-bit results are stored using octal 13 as the plus sign. Rounding is legal for both fixed-point and floating-point formats. If P = 0, positive signed 4-bit results are stored using octal 14 as the plus sign.

Provided that string 1 and string 2 are not overlapped, the contents of the decimal number that starts in location YC1 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero	-	If result equals zero, then ON; otherwise, OFF
Negative	-	If result is negative, then ON; otherwise, OFF
Truncation	-	If least significant truncation without rounding, then ON; otherwise, OFF
Exponent Overflow	-	If exponent of floating-point result is > 127, then ON; otherwise, unchanged
Exponent Underflow	-	If exponent of floating-point result is < -128, then ON; otherwise, unchanged
Overflow	-	If fixed point integer overflow, then ON; otherwise, unchanged.

NOTES:

1. Truncation fault occurs if the truncation indicator is set and the truncation fault enable (T) bit is 1.

2. An Illegal Procedure fault occurs if:

   - o Illegal address modification is specified or illegal repeat is used.
   - o Any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.
   - o The values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.

3. Refer to Explanation of the MLR instruction for information on string replication.

4. If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.

EXAMPLES:

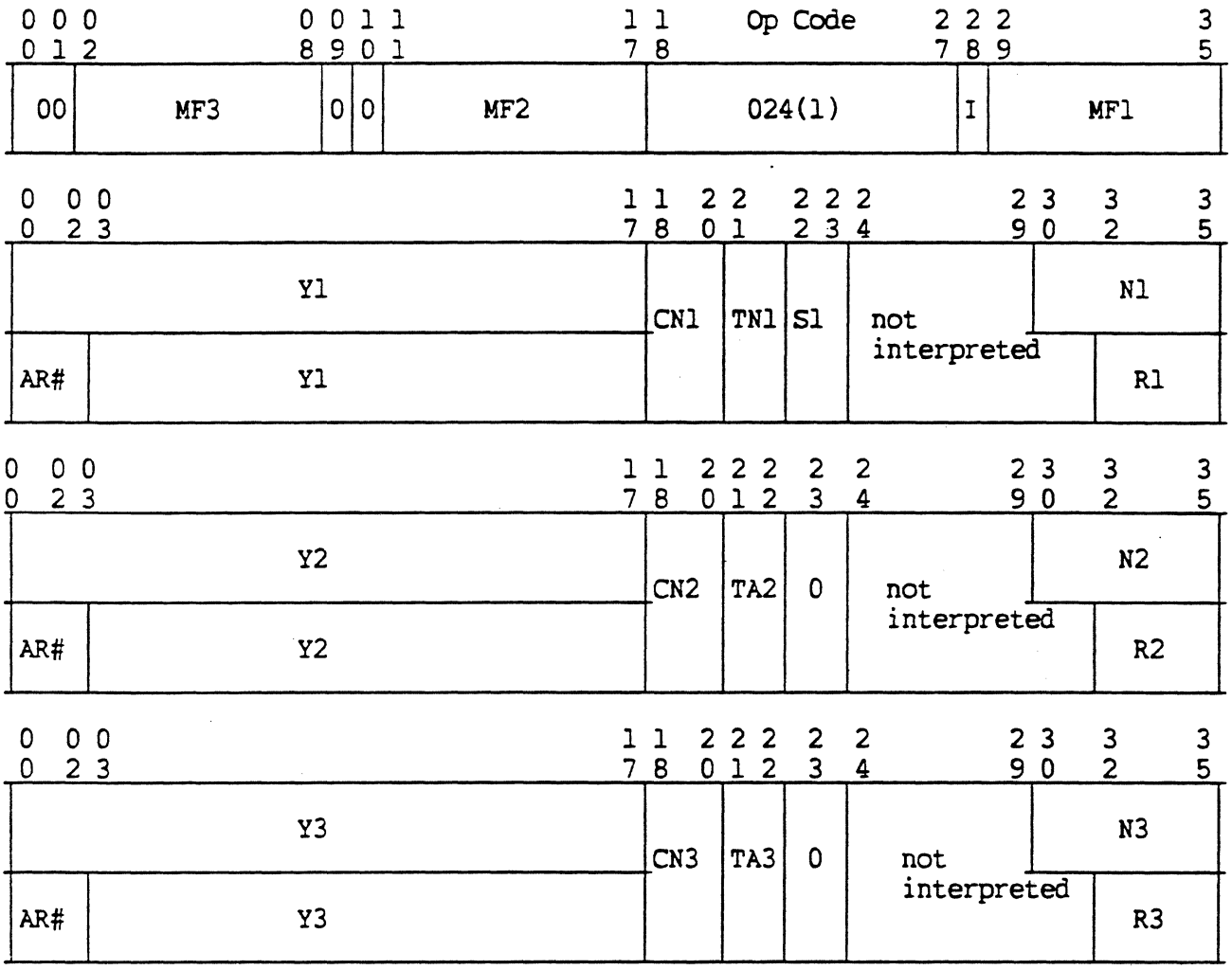
```
1       8       16              32
        MVN     ,,1             with rounding option
        NDSC4   FLD1,0,8,2,-3   sending field operand descriptor
        NDSC4   FLD2,1,7,1,-2   receiving field operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    8P1234567+      1 2 3 4 5 6 7 +
FLD2    EDEC    8P0             0 + 1 2 3 4 5 7  (Result)
        USE                     no indicators set ON

        MVN     ,,,,1           with truncation fault enable option
        NDSC9   FLD1,3,9,2,-2   sending field operand descriptor
        NDSC4   FLD2,0,8,0      receiving field operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    12A12345678-    0 0 0 1 2 3 4 5 6 7 8 -
FLD2    BSS     1               - 1 2 3 4 5 + 1  (Result)
        USE                     negative and truncation set ON
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16              32
-------------------------------------------------------------------
        EAX1    1               load character address into X1
        EAX2    2               load address modifier into X2
        EAX7    7               load FLD1 length into X7
        EAX4    FLD1            load FLD1 address into X4
        AWDX    0,4,4           put FLD1 address into AR4
        MVN     (1,1,,1),(,,1),1,1 - with rounding and plus sign options
        NDSC9   0,,X7,2,-2,4    FLD1's operand descriptor (FLD1,1,7,2,-2)

        ARG     FLD2+1          pointer to indirect operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    8A123456+       0 1 2 3 4 5 6 +
FLD2    EDEC    8P0             0 0 0 0 1 2 3 5  (Result)
        NDSC4   FLD2,2,6,3,-2   recg. field indirect operand descriptor
        USE                     no indicators set ON
```

MVNE	Move Numeric Edited	024 (1)

FORMAT:



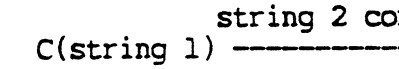
CODING FORMAT: The MVNE instruction is coded as follows:

```
1       8       16
_______________________

        MVNE    (MF1),(MF2),(MF3)
        NDSCn   LOCSYM,CN,N,S,,AM
        ADSC9   LOCSYM,CN,N,AM
        ADSCn   LOCSYM,CN,N,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY:



EXPLANATION: Starting at location YC1, the string of numeric characters of data type TN1 is moved to the string of alphanumeric characters of data type TA3 starting at location YC3. The move is under control of the micro-operation sequence of length L2 and type TA2 = 00 that starts at location YC2. (Refer to "Micro Operations" in this section).

Maximum allowable length for L1, L2, and L3 is 63; they are not checked for length greater than 63. Only the rightmost 6 bits (30-35) are interpreted for length. Likewise when a register is specified as containing the length, only the rightmost 6 bits of the register are interpreted.

The operation stops when L3 is exhausted.

The results are not guaranteed when strings are overlapped.

The sign and decimal type of the sending field is given by S1. The contents of the numeric character string that starts at YC1 and the micro-operation sequence that starts at YC2 remain unchanged.

On the processor, L3 = 0 is the normal termination; thus, at the start of the instruction, if L3 = 0 and there are no faults (see Note 1), no operation is performed and the instruction terminates normally, independently of whether L1 or L2 equals zero, because the hardware does not access these fields when L3 = 0.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

MVNE

NOTES:

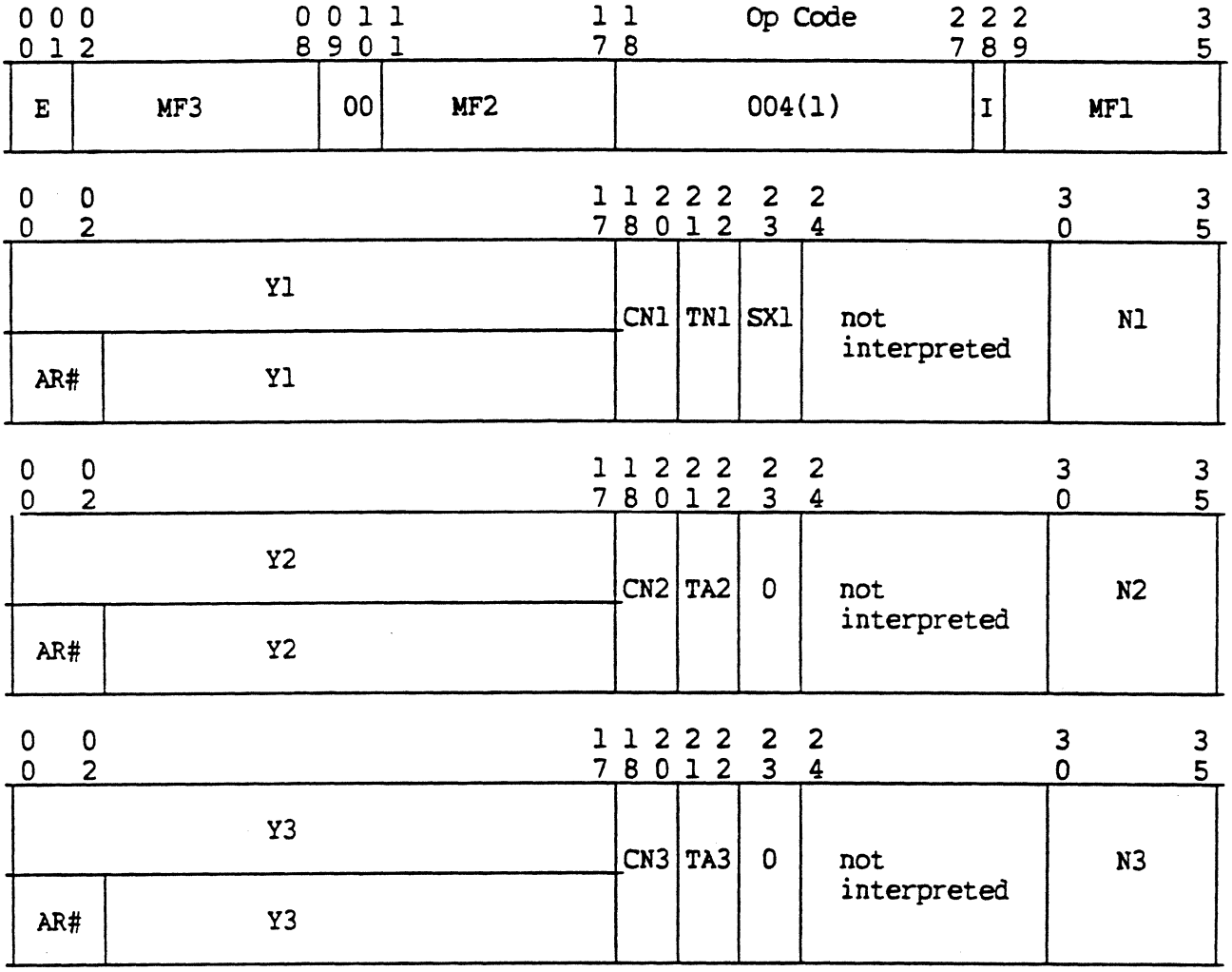
1. An Illegal Procedure fault occurs under the following conditions.
   - o If illegal address modification is used
   - o If illegal repeats are used
   - o If an illegal micro operation is used
   - o If TA2 is not = 0
   - o If an attempt is made to access string 2 when $L_2 = 0$
2. Refer to Micro Operations for Edit Instructions in Section 7.

EXAMPLES:

```
1       8       16                  32
        MVNE                        with ($) float and (.) inserted
        NDSC9   FLD1,0,10,2         sending field operand descriptor
        ADSC9   FLD2,0,14           micro-op string operand descriptor
        ADSC6   FLD3,0,12           receiving field operand descriptor
        USE     CONST.              memory contents in ASCII characters
FLD1    EDEC    10A300405-          000300405-00
FLD2    MICROP  (CHT,0),8HØ*+-$,.0,(MFLC,7),(ENF,8),(INSB,7)
        MICROP  (MVC,2),(INSN,4) memory contents in BCD characters

FLD3    BSS     2                   Ø Ø Ø $ 3 0 0 4 . 0 5 -     (Result)
        USE

        MVNE                        with (*) protection and (.) insertion
        NDSC4   FLD1,0,8,2          sending field operand descriptor
        ADSC9   FLD2,0,6            micro-op string operand descriptor
        ADSC9   FLD3,0,12           receiving field operand descriptor
        USE     CONST.              memory contents in packed decimal
FLD1    EDEC    8P250509-           025059-
FLD2    MICROP  (MVZA,5),(SES,8),(INSA,7),(MVC,2)
        MICROP  (INSN,4),(INSM,3)
*                                   memory contents in ASCII characters
FLD3    BSS     3                   * 2 5 0 5 . 0 9 - Ø Ø Ø (Result)
        USE

        MVNE                        +1234 ----> 1234
        NDSC4   6PACK,3,5,1         -1234 ----> 123M
        ADSC9   MOPS,0,6
        ADSC6   PRTOUT,0,4
        MVT
        ADSC6   PRTOUT,0,4
        ADSC9   APRINT,0,4
        ARG     TABLE
        USE     CONST.
MOPS    MICROP  (MVC,3),(LTE,3),10000,(LTE,4),10040,(MORS,1)
TABLE   ASCII   2,01234567          0X
        VFD     A18/89,18/0,36/0    1X
        OCT     0,0                 2X
        OCT     0,0                 3X
        UASCI   2, JKLMNOP          4X
        VFD     U18/QR,18/0,36/0    5X
        OCT     0,0                 6X
        OCT     0,0                 7X
        USE
```

MVNEX	Move Numeric Edited Extended	004 (1)

FORMAT:



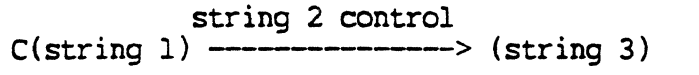
CODING FORMAT: 1 8 16

```
MVNEX   (MF1),(MF2),(MF3),E
NDSCn   LOCSYM,CN,N,S,,AM
ADSC9   LOCSYM,CN,N,AM
NDSCn   LOCSYM,CN,N,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY:



EXPLANATION: The function of this instruction is similar to the MVNE instruction, but with the added capability of initializing an edit insertion table. (See Table 7-2). A 2-bit code entered in field E (bits 0 and 1) specifies the character set associated with the edit insertion table as follows.

E-Bits 0 and 1

00	EBCDIC
01	BCD
10	ASCII
11	Illegal, IPR fault

TN1 determines whether the input data is unpacked (0) or packed (1). TA3 determines the character size (9, 6, or 4 bits) of the output data. It is the user's responsibility to make TA3 consistent with bits 0 and 1 of the instruction. S determines the location of the sign of the input data (leading, trailing, overpunched, separate). Refer to the Explanation for MVNE for additional information.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS: RPT, RPD, RPL

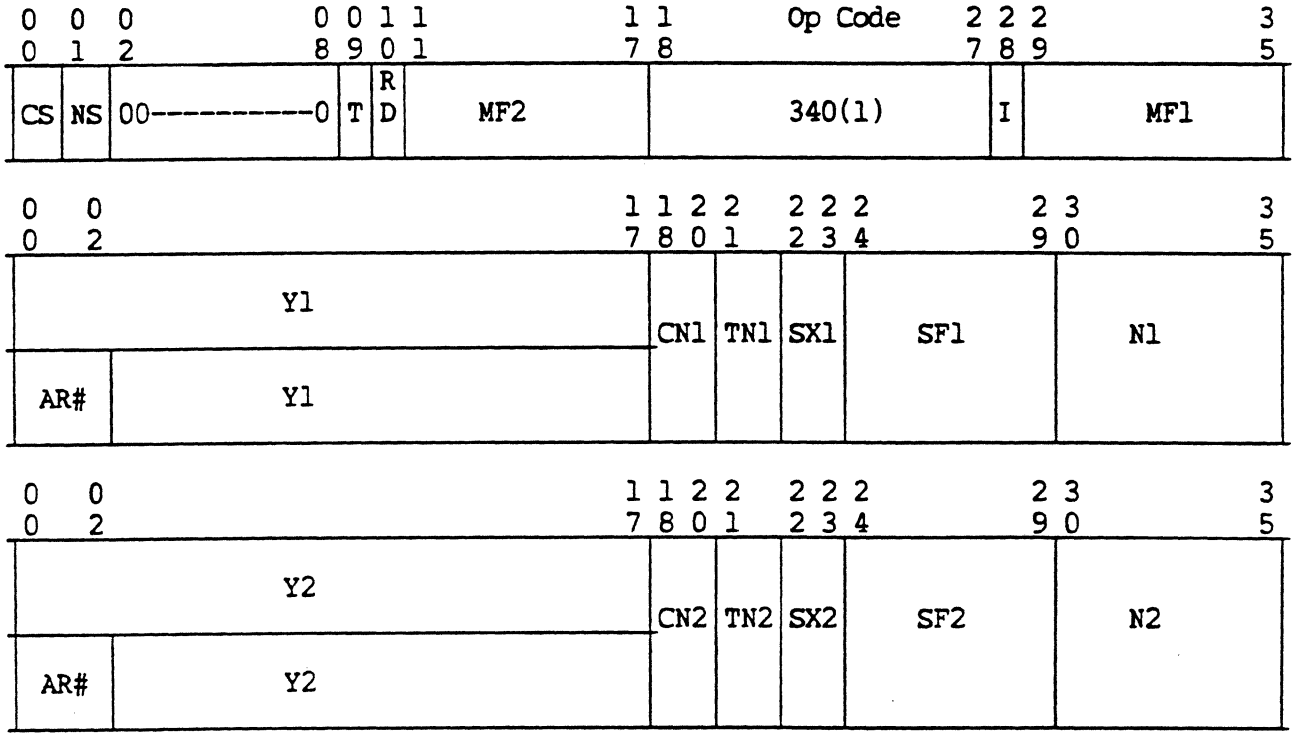
INDICATORS: None affected

NOTES:

1. Notes for MVNE apply to MVNEX.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.
3. Refer to "Micro Operations for Edit Instructions" in Section 7.

MVNX	Move Numeric Extended	340 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
MVNX    (MF1),(MF2),RD,CS,T,NS
NDSCn   LOCSYM,CN,N,SX,SF,AM
NDSCn   LOCSYM,CN,N,SX,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) --> C(string 2)

EXPLANATION: Starting at location YC1, the decimal number of data type TN1 and sign and decimal type SX1 is moved, properly scaled, to the decimal number of data type TN2 and sign and decimal type SX2 that starts at location YC2.

The character set is defined by CS (EBCDIC/ASCII). Placement of an overpunched sign in the output is controlled by NS. (Refer to the definition of the NS field in the beginning of Section 8.)

If SX2 indicates a fixed-point format, the result is stored as L2 digits using scale factor SF2, and thereby may cause most-significant-digit overflow and/or least-significant-digit truncation.

Rounding is legal for both floating and scaled formats. The contents of the decimal number that starts in location YC1 remain unchanged.

The SX field is interpreted as follows:

TN = 0: Unpacked data (9 bits)

SX

00: LS*, OVP*, scaled
01: LS, separate, scaled
10: TS*, separate, scaled
11: TS, OVP, scaled

TN = 1: Packed data (4 bits)

SX

00: LS, separate, floating-point
01: LS, separate, scaled
10: TS, separate, scaled
11: No sign, scaled

* LS.... Leading sign
OVP... Overpunched
TS.... Trailing sign

Bits 0 and 1 of the instruction word are interpreted as follows:

MVNX

Bit 0 of instruction word: Specifies the character set.

=0: EBCDIC data (but not the strict EBCDIC sign)
=1: ASCII data (but not the strict ASCII sign)

Bit 1 of instruction word: Specifies no-sign output.

=0: The instruction execution is not affected.
=1: The sign character in the receive field where the result is to be placed is affected as follows:

If the operand descriptor of the receive field contains TN = 0 and SX = 00 or 11 (indicating that output is an overpunched sign), the overpunched sign is not placed in the specified field. Instead, an appropriate decimal number (0-9) is placed in the receive field irrespective of whether the sign of the calculated result is positive or negative. This is a no-sign output.

For values of SX and TN, bit 1 is ignored. This applies both to EBCDIC and ASCII.

The hardware recognizes an implied plus sign on input data. For unpacked data (TN=0) with indicated overpunched sign (SX1 = 00 or 11), if the hardware does not find a plus or minus overpunched sign character in the overpunched sign character position, the hardware checks for a numeric digit (0-9). The zone bits are not included in the check; only the lower-order 4 bits are checked. If this check indicates a numeric digit from the appropriate character set, the hardware accepts the digit and assumes the sign to be plus. Otherwise an IPR fault is generated.

Table 8-2 shows the character codes for ASCII and EBCDIC overpunched signs.

MVNX

Table 8-2. Character Codes For ASCII and EBCDIC Overpunched Signs

Card Punch Code	Normal Interp.	Ovrpnch Interp.	ASCII Code	EBCDIC Code
0	0	0	060	360
1	1	1	061	361
2	2	2	062	362
3	3	3	063	363
4	4	4	064	364
5	5	5	065	365
6	6	6	066	366
7	7	7	067	367
8	8	8	070	370
9	9	9	071	371
12	+	+0	053	NA
space	space	+0	040	NA
12-0	{	+0	173	300
12-1	A	+1	101	301
12-2	B	+2	102	302
12-3	C	+3	103	303
12-4	D	+4	104	304
12-5	E	+5	105	305
12-6	F	+6	106	306
12-7	G	+7	107	307
12-8	H	+8	110	310
12-9	I	+9	111	311
11	-	-0	055	NA
11-0 (GBCD)	^	-0	136	NA
11-0(ASCII)	}	-0	175	320
11-1	J	-1	112	321
11-2	K	-2	113	322
11-3	L	-3	114	323
11-4	M	-4	115	324
11-5	N	-5	116	325
11-6	O	-6	117	326
11-7	P	-7	120	327
11-8	Q	-8	121	330
11-9	R	-9	122	331

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

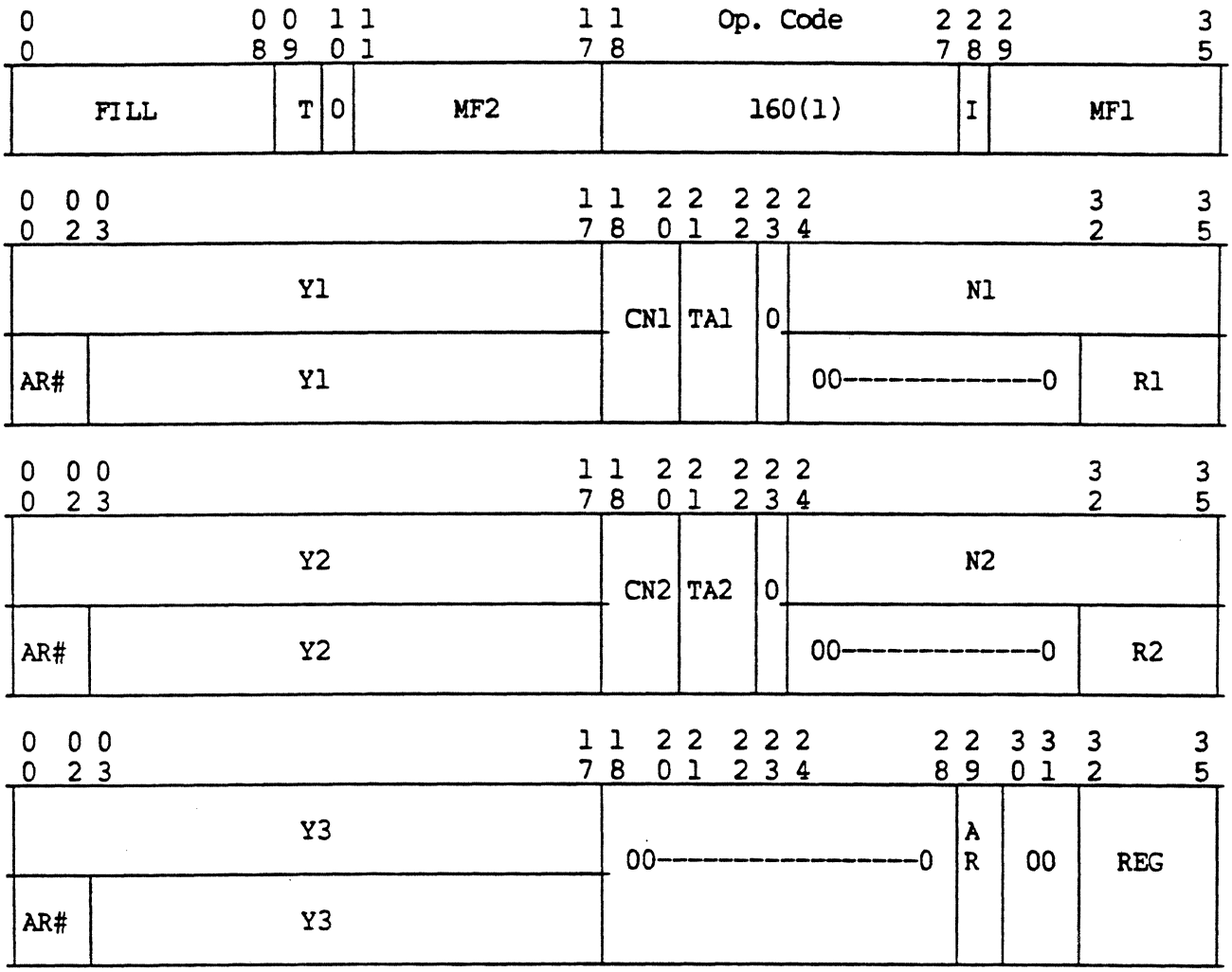
Zero	-	If result is zero, then ON; otherwise, OFF
Negative	-	If result is negative, then ON; otherwise, OFF
Truncation	-	If least-significant truncation without rounding, then ON; otherwise, OFF
Overflow	-	If fixed-point integer overflow, then ON; otherwise, unchanged
Exponent Overflow	-	If exponent of floating-point result > 127, then ON; otherwise, unchanged
Exponent Underflow	-	If exponent of floating-point result < -128, then ON; otherwise, unchanged

NOTES:

1. A Truncation fault occurs if the truncation indicator is set and the truncation fault enable bit (T) is a 1.
2. An IPR fault occurs if any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.
3. An IPR fault occurs if the values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.
4. An IPR fault occurs illegal address modifications or illegal repeats are used.
5. Refer to Note 3 of MLR for information on string replication.
6. If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.

MVT	Move Alphanumeric with Translation	160 (1)

FORMAT:



CODING FORMAT: The MVT instruction is coded as follows:

1 8 16

```
MVT     (MF1),(MF2),FILL,T
ADSCn   LOCSYM,CN,N,AM
ADSCn   LOCSYM,CN,N,AM
ARG     TABLE,REG,AM
```

OPERATING MODES: Any

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

EXPLANATION: Starting at location YC1, the alphanumeric characters of data type TA1 are used as an index to a table of contiguous 9-bit characters that start at location Y3 (character position 0). The octal code of the character of string-1 is used as an index to string-3. The indexed 9-bit characters (or right-justified 4- or 6-bit characters) of string-3 replace the contents of string 2, starting at location YC2. If TA1 and TA2 are dissimilar, each character will have high-order truncation. If L1 is less than L2, the FILL character (the entire 9 bits) is used as the index to the table to replace the L2-L1 least significant characters of string 2. The contents of string 1 remain unchanged except in cases of string overlap. When the 9-bit character translate table and the string are overlapped, the result is unpredictable.

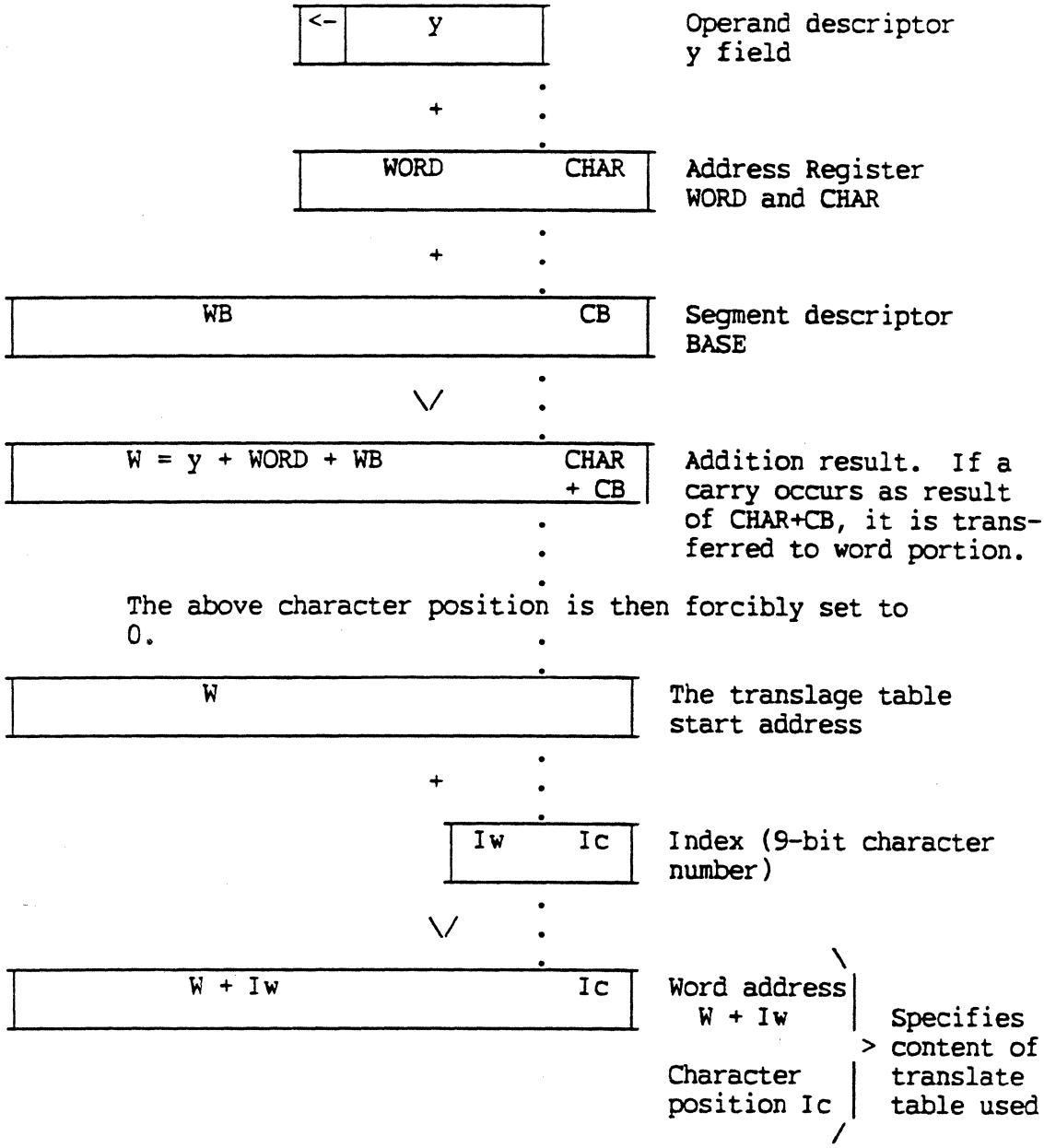
L2 = 0 does not necessarily mean that the instruction functions as a NOP because the truncation indicator may be affected.

If $L_1 < L_2$, and type TA_1 is 4 or 6-bit, the low-order 4 or 6 bits of the fill character (9-bit) in the instruction word are defined as a table index.

The translation table must begin at a word boundary at character position 0. The index (expressed by the number of 9-bit characters) is added to the starting word address of the table. It is computed in the same way as for normal address modification; however, the computed address is then used as a word address (with character position ignored). The index is added to this word address as a 9-bit number.

The translation table length is determined by the highest possible index character octal value that may be found in the indexing data string. The table is always indexed in 9-bit increments, regardless of the data type being moved. The 9-bit character represented in the table must be the same data type as the receiving field. (See Examples for MVT.)

When address register modification is specified, the translation table address is generated as follows.



When index register modification is specified, the content of that register is added to the word portion.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, and REG field for Y3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Truncation - If L1 is > L2, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or illegal repeats are executed.
2. A Truncation fault occurs if the truncation indicator is set and the truncation fault enable (T) bit is a 1.
3. Refer to Explanation of the MLR instruction for information on string replication.

EXAMPLES:

```
1       8       16                  32
        MVT     ,,52                with fill index a minus
        ADSC6   FLD1,4,7            indexing operand descriptor
        ADSC4   FLD2,0,8            receiving operand descriptor
        ARG     TABLE               pointer to 4-bit table
        USE     CONST.              memory contents
FLD1    BCI     2,ÞÞÞÞÞ123456       202020202001020304050620
FLD2    BSS     1                   0123456-  (Result)
TABLE   NULL
        OCT     000001002003,004005006007     0X
        OCT     010011017017,017017017017     1X
        OCT     000017017017,017017017017     2X
        OCT     017017017017,017017017017     3X
        OCT     017017017017,017017017017     4X
        OCT     017017015017,017017017017     5X
        OCT     014017017017,017017017017     6X
        OCT     017017017017,017017017017     7X
        USE
        MVT
        ADSC4   FLD3,,8
        ADSC4   FLD4,,8
        ARG     TAB
        USE     CONST.
FLD3    OCT     022064126317        123456++
FLD4    BSS     1                   022064126314  (Result)
TAB     NULL
        OCT     000001002003,004005006007
        OCT     010011014014,014015014014
        USE
```

```
1       8       16                32
        MVT     ,,040             blank fill
        ADSC6   FLD1,0,18
        ADSC9   FLD2,0,20
        ARG     TABLE9            pointer to translation table
        USE     CONST.
FLD1    BCI     3,TTYMESSAGE201
FLD2    BSS     5
TABLE9  EDITP   SAVE,ON
        UASCI   2,01234567        0X
        UASCI   2,89[#@:>?        1X
        UASCI   2,ØABCDEFG        2X
        UASCI   2,HI&.](<\        3X
        UASCI   2,^JKLMNOP        4X
        UASCI   2,QR-$*);'        5X
        UASCI   2,/STUVWX         6X
        UASCI   2,YZ_,%="!        7X
        EDITP   RESTORE
        USE
```

NOTE: The translation table length in the above example is determined by the highest octal value for the characters of the indexing string (Field 1). The characters in the above translation table are represented in 9-bit ASCII code, the same data type as the receiving field (Field 2). Also, the table is 64 characters in length, in direct relation to the BCD character set (highest value octal 77).

NARn	Numeric Descriptor to Address Register n	66n (1))

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        NARn    LOCSYM,RM,AM
```

OPERATING MODES: Execution in NS mode only

SUMMARY: For n = 0,1,...,or 7 as determined by op code

$C(Y)_{0-17}$ --> $C(ARn)_{0-17}$

$C(Y)_{18-20}$ ---translated---> $C(ARn)_{18-23}$; C(Y) unchanged

EXPLANATION: The numeric descriptor is fetched from the computed effective address Y and the TN bit is examined. If TN = 0 (9-bit characters), bits 18 and 19 of the CN field go to the corresponding positions of ARn and zeros fill bits 20-23 of ARn. If TN = 1, the 4-bit character contained in the CN field, is converted to bit string representation and placed in bits 18-23 of ARn. In either case, the descriptor word address field (0-17) goes to bits 0-17 of ARn.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.
2. An IPR occurs if an attempt is made to execute this instruction in the ES mode.

EXAMPLES:

```
1       8       16              32
        NAR2    DESCR           load data string address into AR2
                .
                .
                .
DESCR   NDSC4   FLD1,7,8,3,2    0 3 2 4 2 6 7 7 0 2 1 0  - descriptor
*                               0 3 2 4 2 6 6 5          - result in AR2
```

NEG	Negate (A-Register)	531 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: -C(A) --> C(A) if C(A) ≠ 0

EXPLANATION: This instruction changes the number in A to its negative (if ≠ 0). The operation is executed by forming the two's complement of the string of 36 bits.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of A is exceeded, then ON

NOTE: An Illegal Procedure fault occurs when an illegal repeat is used.

NEGL	Negate Long (AQ-Register)	533 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY - C(AQ) --> C(AQ) if C(AQ) ≠ 0

EXPLANATION: This instruction changes the number in AQ to its negative (if ≠ 0). The operation is executed by forming the two's complement of the string of 72 bits.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of AQ is exceeded, then ON

NOTE: An Illegal Procedure fault occurs when an illegal repeat is used.

NOP	No Operation	011 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: No operation takes place; the effective address is always prepared.

EXPLANATION: No operation takes place but address preparation is performed according to the specified modifier, if any. If modification other than DU or DL is used, the generated addresses may cause faults.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: The use of Indirect then Tally modifiers ID, DI, IDC, DIC, SCR, or SC changes the address and tally fields of the referenced indirect words; the Tally Runout indicator may be set ON.

NOTES:

1. An Illegal Procedure fault occurs when an illegal repeat is used.

2. Because address preparation takes place, modification may result in a Bounds fault.

ORA	OR to A-Register	275 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35,

$C(A)_i$ OR $C(Y)_i$ --> $C(A)_i$; C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0$ = 1, then ON; otherwise, OFF

ORAQ	OR to AQ-Register	277 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 71,

$C(AQ)_i$ OR $C(Y\text{-pair})_i$ --> $C(AQ)_i$; C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

ORQ	OR to Q-Register	276 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35,

$C(Q)_i$ OR $C(Y)_i$ --> $C(Q)_i$; C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

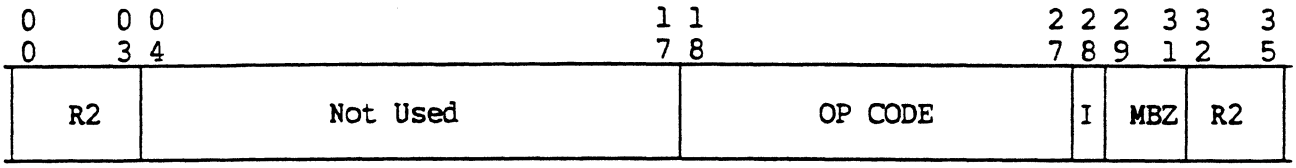
ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0$ = 1, then ON; otherwise, OFF

ORRR	OR Register to Register	536 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
        ORRR   R1,,R2
```

OPERATING MODES: Executes in ES mode only

SUMMARY: R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

$C(R1)_i$ OR $C(R2)_i$ --> $C(R1)_I$ i = 0,1,2,...,35

C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS: Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

ORSA	OR to Storage from A-Register	255 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35,

$C(A)_i$ OR $C(Y)_i$ --> $C(Y)_i$; C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

ORSQ	OR to Storage from Q-Register	256 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: For i = 0 to 35,

$C(Q)_i$ OR $C(Y)_i$ --> $C(Y)_i$; C(Q) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

ORSXn	OR to Storage from Index Register n	24n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

NS Mode

For n = 0,1,...,or 7 as determined by op code

For i = 0 to 17, $C(Xn)_i$ OR $C(Y)_i$ --> $C(Y)_i$

C(Xn) and C(Y)18-35 unchanged

ES Mode

For n = 0,1,...,or 7 as determined by op code

For i = 0 to 35, $C(GXn)_i$ OR $C(Y)_i$ --> $C(Y)_i$;

C(GXn) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, or RPL of ORSX0

INDICATORS: NS Mode

Zero - If $C(Y)_{0-17}$ = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

ES Mode

Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then OFF; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

ORXn	OR to Index Register n	26n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n=0,1,...,or 7 as determined by op code

For i = 0 to 17, $C(Xn)_i$ OR $C(Y)_i$ --> $C(Xn)_i$;

C(Y) unchanged

ES Mode

For n=0,1,...,or 7 as determined by op code

For i = 0 to 35, $C(GXn)_i$ OR $C(Y)_i$ --> $C(GXn)_i$;

C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, or RPL of ORX0

INDICATORS: Zero - If C(Xn/GXn) = 0, then ON; otherwise, OFF

Negative - If $C(XN/GXn)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged illegal but executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

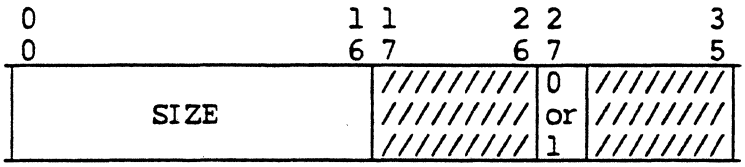
PAS	Pop Argument Stack	176 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: Modifies bound field of the argument stack register (ASR).

EXPLANATION: This instruction provides a means of modifying the bound field of the ASR. The 1-word operand is obtained from memory location Y. The memory operand has the following format:



If ASR flag bit 27 = 0 nothing occurs. The argument segment is empty and the instruction terminates.

If ASR flag bit 27 = 1, the instruction proceeds. The SIZE field is the number of descriptors to be framed, minus 1 (i.e., the number of double-word memory locations).

The descriptor SIZE field is converted to number of bytes by appending three 1-bits as the least-significant bits, producing a 20-bit byte size (SIZE-bytes). Accordingly, a memory operand SIZE field of zero means frame one descriptor. Using the 20-bit SIZE-bytes, the instruction proceeds as follows (shaded area is ignored):

If memory operand bit 27 = 0, ASR flag bit 27 and ASR bound field are set to zero and the instruction terminates.

If memory operand bit 27 = 1, the SIZE-bytes is compared to the bound field of the ASR as follows:

- If SIZE-bytes < Bound, then SIZE-bytes replaces contents of ASR Bound field.
- If SIZE-bytes >= Bound, then ASR remains unchanged.

- C(HWMR) is unchanged for all cases.
- Bits 17-26 and 28-35 of the operand are ignored by the hardware.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An IPR fault occurs if illegal address modifications or illegal repeats are executed.

EXAMPLE:

```
1       8       16              32
        INHIB   ON
SVPTR1  STAS    SAVE1           store argument stack
        SDR     P1,0            save descriptor register 1
        STP     P1,SAV11        store pointer to descriptor register 1
        TRA     0,5

RTPTR1  NULL
        LDP     P1,SAV11        locates and restores descriptor register 1
        PAS     SAVE1           restores argument stack
        TRA     0,5
```

PULS1	Pulse One	012 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: No operation takes place

EXPLANATION: The PULS1 instruction is identical to the NOP instruction except that it causes certain unique external hardware monitoring synchronization signals to appear in the processor logic circuitry.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: The use of Indirect then Tally modifiers ID, DI, IDC, DIC, SCR, or SC changes the address and tally fields of the referenced indirect words; the Tally Runout indicator may be set ON.

NOTES:

1. An Illegal Procedure fault occurs when illegal repeats are used.
2. This instruction is for use only in external hardware monitoring equipment and not in normal coding.

PULS2	Pulse Two	013 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: No operation takes place

EXPLANATION: The PULS2 instruction is identical to the NOP instruction except that it causes certain unique external hardware monitoring synchronization signals to appear in the processor logic circuitry.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: The use of Indirect then Tally modifiers ID, DI, IDC, DIC, SCR, or SC changes the address and tally fields of the referenced indirect words; the Tally Runout indicator may be set ON.

NOTES:

1. An Illegal Procedure fault occurs when illegal repeats are used.
2. This instruction is for use only in external hardware monitoring equipment and not in normal coding.

QFAD	Quadruple-Precision Floating Add	476 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ, LOR) + C(Y 4 words)] normalized --> C(EAQ, LOR)

EXPLANATION: The exponent underflow indicator is not set when the low-order exponent (E_L) is in underflow (E_U-15 < -128). At this time, the correct value + 256 is loaded into E_L and the correct value into the low-order mantissa (M_L - LOR_{8-71}).

When the mantissa (both the high-order and the low-order portions) of the operation result is 0, then -128 is loaded into E_U and E_L.

When the low-order mantissa, but not the high-order mantissa, of the operation result = 0, then -128 is loaded into E_L.

In any other case, E_U -15 is loaded into E_L.

In quadruple-precision arithmetic operations, an additional digit (4 bits) called a guard digit, is assumed next to the low-order position. An operation is performed in which the intermediate result that includes the guard digit is normalized. The high-order 124 bits are loaded into the EAQ and LOR registers.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, SC, SCR, CI

ILLEGAL REPEATS: RPT, RPD, RPL

QFAD

INDICATORS:	Zero	- If $[C(AQ)_{0-63}, C(LOR)_{8-71}] = 0$, then ON; otherwise, OFF
	Negative	- If $C(A)_0 = 1$, then ON; otherwise OFF
	Exponent Overflow	- If exponent > +127, then ON
	Exponent Underflow	- If exponent < -128, then ON

NOTE: An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QFLD	Quadruple-Precision Floating Load	432 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(Y 4 words) --> C(EAQ, LOR)

Bits 0-7 of C(Y 4 words) --> C(E)

Bits 8-71 of C(Y 4 words) --> Bits 0-63 of C(AQ)

00.....0 --> bits 64-71 of C(AQ)

Bits 72-143 of C(Y 4 words) --> C(LOR)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, SC, SCR, CI

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Zero - If $[C(AQ)_{0-71}, C(LOR)_{12-71}] = 0$,then ON; otherwise OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise OFF

NOTE: An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QFMP	Quadruple-Precision Floating Multiply	462 (0)

FORMAT: Single-word instruction format (see Figure 8-1).

OPERATING MODES: Any

SUMMARY: [C(EAQ, LOR) x C(Y 4 words)] normalized --> C(EAQ, LOR)

EXPLANATION: The exponent underflow indicator is not set when the low-order exponent (E_L) is in underflow (E_U-15 < -128). At this time, the correct value + 256 is loaded into E_L and the correct value into the low-order mantissa (M_L - LOR_{8-71}).

When the mantissa (both the high-order and the low-order portions) of the operation result is 0, then -128 is loaded into E_U and E_L.

When the low-order mantissa, but not the high-order mantissa, of the operation result = 0, then -128 is loaded into E_L.

In any other case, E_U -15 is loaded into E_L.

In quadruple-precision arithmetic operations, an additional digit (4 bits), called a guard digit, is assumed next to the low-order position. An operation is performed in which the intermediate result which includes the guard digit are normalized. The high-order 124 bits are loaded into the EAQ and LOR registers.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, SC, SCR, CI

ILLEGAL REPEATS: RPT, RPD, RPL

QFMP

INDICATORS:

Zero	-	If $[C(AQ)_{0-63}, C(LOR)_{8-71}] = 0$, then ON; otherwise, OFF
Negative	-	If $C(A)_0 = 1$, then ON; otherwise OFF
Exponent Overflow	-	If exponent > +127, then ON
Exponent Underflow	-	If exponent < - 128, then ON

NOTE: An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QFSB	Quadruple-Precision Floating Subtract	576 (0)

FORMAT: Single-word instruction format (see Figure 8-1).

OPERATING MODES: Any

SUMMARY: [C(EAQ, LOR) - C(Y 4 words)] normalized —> C(EAQ, LOR)

EXPLANATION: The exponent underflow indicator is not set when the low-order exponent (E_L) is in underflow (E_U-15 < -128). At this time, the correct value + 256 is loaded into E_L and the correct value into the low-order mantissa (M_L - LOR_{8-71}).

When the mantissa (both the high-order and the low-order portions) of the operation result is 0, then -128 is loaded into E_U and E_L.

When the low-order mantissa, but not the high-order mantissa, of the operation result = 0, then -128 is loaded into E_L.

In any other case, E_U -15 is loaded into E_L.

In quadruple-precision arithmetic operations, an additional digit (4 bits), called a guard digit, is assumed next to the low-order position. An operation is performed in which the intermediate result that includes the guard digit is normalized. The high-order 124 bits are loaded into the EAQ and LOR registers.

During the operation, a two's complement of the subtrahend is justified and added.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, SC, SCR, CI

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero	-	If $[C(AQ)_{0-63}, C(LOR)_{8-71}] = 0$, then ON; otherwise, OFF
Negative	-	If $C(A)_0 = 1$, then ON; otherwise, OFF
Exponent Overflow	-	If exponent > +127, then ON
Exponent Underflow	-	If exponent < - 128, then ON

NOTE: An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QFST	Quadruple-Precision Floating Store	453 (0)

FORMAT: Single-word instruction format (see Figure 8-1).

PROCEDURE MODE: Any

SUMMARY: [C(EAQ, LOR) --> C(Y 4 words)] normalized

C(E) --> bits 0-7 of C(Y 4 words)

Bits 0-63 pf C(AQ) --> bits 8-71 of C(Y 4 words)

Bits 64-71 of C(AQ) are ignored

C(LOR) --> bits 72-143 of C(Y 4 words)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, SC, SCR, CI

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QFSTR	Quadruple-Precision Floating Store Rounded	466 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

PROCEDURE MODE: Any

SUMMARY: [Bits 0-63 of C(AQ), bits 12-71 of C(LOR)] rounded, normalized --> C(Y-pair)

EXPLANATION: Arithmetic operation procedure

$[C(AQ)_{0-63}$ + Carry] normalized --> C(Y-pair)

If C(AQ, LOR) is positive then Carry = 0

If C(AQ, LOR) is negative;

if $C(LOR)_{13-71} = 0$, then Carry = 0

if $C(LOR)_{13-71} \neq 0$, then Carry = $C(LOR)_{12}$

Using the above processing, positive and negative data with an equal absolute value are rounded to give values with equal absolute value.

If the mantissa of the result = 0 by rounding, -128 is stored in $C(Y\text{-pair})_{0-7}$.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, SC, SCR, CI

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero - If $C(Y\text{-pair})_{8-71} = 0$, then ON; otherwise, OFF

Negative - If $C(Y\text{-pair})_8 = 1$, then ON; otherwise, OFF

Exponent Overflow - If exponent > +127, then ON

Exponent Underflow - If exponent < -128, then ON

NOTE: An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QLR	Q-Register Left Rotate	776 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: Rotate C(Q) left by the number of positions indicated by bits 11-17 (NS mode) or 27-33 (ES mode) of Y (Y modulo 128). Enter each bit leaving bit position 0 of Q into bit position 35 of Q.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. The rotate count in the instruction must be a decimal number. To "right-rotate" n bits, use QLR 36-n.

2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

QLS	Q-Register Left Shift	736 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: Shift C(Q) left by the number of positions indicated by bits 11-17 (NS mode) or 27-33 (ES mode) of Y (Y modulo 128). Fill vacated positions with zeros. The shift count in the instruction must be a decimal number.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0 = 1$, then ON; otherwise, OFF

Carry - If $C(Q)_0$ changes during the shift, then ON; otherwise, OFF. When the carry indicator is ON, the algebraic range of Q has been exceeded.

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

QRL	Q-Register Right Logical Shift	772 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: Shift C(Q) right by the number of positions indicated by bits 11-17 (NS mode) or 27-33 (ES mode) of Y (Y modulo 128). Fill vacated positions with zeros. The shift count in the instruction must be a decimal number.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

QRS	Q-Register Right Shift	732 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: Shift C(Q) right by the number of positions indicated by bits 11-17 or 27-33 (ES mode) of Y (Y modulo 128). Fill vacated positions with bit 0 of C(Q). The shift count in the instruction must be a decimal number.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

QSMP	Quadruple-Precision Floating Multiply with Double-Precision Operands	460 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) x C(Y 4 words)] normalized --> C(EAQ, LOR)

EXPLANATION: The exponent underflow indicator is not set when the low-order exponent (E_L) is in underflow (E_U-15 < -128). At this time, the correct value + 256 is loaded into E_L and the correct value into the low-order mantissa (M_L - LOR_{8-71}).

When the mantissa (both the high-order and the low-order portions) of the operation result is 0, then -128 is loaded into E_U and E_L.

When the low-order mantissa, but not the high-order mantissa, of the operation result = 0, then -128 is loaded into E_L.

In any other case, E_U -15 is loaded into E_L.

In quadruple-precision arithmetic operations, an additional digit (4 bits), called a guard digit, is assumed next to the low-order position. An operation is performed in which the intermediate result which includes the guard digit are normalized. The high-order 124 bits are loaded into the EAQ and LOR registers.

The 72 bits of $C(AQ)_{0-71}$ are used for the mantissa of the multiplicand.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, SC, SCR, CI

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero	- If $[C(AQ)_{0-63}, C(LOR)_{8-71}] = 0$, then ON; otherwise, OFF
Negative	- If $C(A)_0 = 1$, then ON; otherwise, OFF
Exponent Overflow	- If exponent > +127, then ON
Exponent Underflow	- If exponent < - 128, then ON

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

RCW	Read Connect Word Pair	250 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(Connect Queue Entry) --> C(AQ)

If the queue is empty

0 --> C(AQ)

The SCU selected is the control SCU.

EXPLANATION; The SCU is selected by the control SCU bit. (Refer to SCU configuration register in Section 4.)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI,

ILLEGAL REPEATS: RPD, RPL, and RPT

INDICATORS: Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. An IPR fault occurs if this instruction is executed in Master or Slave modes.
3. The SCU connect masks are not applied.
4. Bound checks on the address are not made.

RET	Return	630 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: $C(Y)_{0-17}$ --> C(IC)

$C(Y)_{18-32}$ --> C(IR)

$C(Y)_{33-35}$ are ignored

C(Y) unchanged

EXPLANATION: This instruction loads the content of the location specified by Y into the instruction counter and indicator register with bit 29 = 0. The RET instruction does not load the instruction segment register (ISR) and the SEGID(IS). The return is then within the current instruction segment. The RET instruction may be thought of as an LDI instruction followed by a transfer to the location specified by $C(Y)_{0-17}$.

The relation between the bit positions of C(Y) and the indicators is as follows:

C(Y) Bit Position	Indicator (or Mask)
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent overflow
23	Exponent underflow
24	Overflow mask
25	Tally runout
26	Parity error
27	Parity mask
28	Master mode
29	Truncation
30	Multiword instruction interrupt
31	Reserved for exponent underflow mask
32	Hexadecimal exponent mode
33-35	000

RET

With unconditional transfer of control instructions, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not affected. An IPR fault does not occur.
- When bit 29 of the instruction word = 1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transter does not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Master mode - If $C(Y)_{28}$ is 1, then no change; otherwise, OFF

All other indicators - If corresponding bit in C(Y) is 1, then ON; otherwise, OFF

NOTES:

1. An Overflow Fault does not occur when the overflow indicator, exponent overflow indicator, or exponent underflow indicator is set ON via the RET instruction, even if the Overflow Mask Indicator is OFF.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

RET

3. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

4. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

5. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

RIMR	Read Interrupt Mask Register	233 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: Port interrupt level masks --> $C(A)_{0-7}$

All mask --> $C(A)_8$

Port connect mask --> $C(A)_9$

0...0 --> $C(A)_{10-35}$

EXPLANATION: This instruction reads the masks in the SCU corresponding to the issuing port; the All Mask is also read.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, and SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The SCU is selected by the control SCU bit.
2. An IPR fault occurs when an attempt is made to execute this instruction in Slave or Master mode.
3. An Illegal Procedure fault occurs if illegal address modification or an illegal repeats are used.

RIW	Read Interrupt Word Pair	412 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: If an unmasked interupt queue in the SCU has an entry then,

C(Word Pair from queue) --> C(AQ)

If no unmasked queue has an entry then,

0...0 --> C(AQ)

EXPLANATION: If any unmasked interrupt queue in the control SCU has an entry, then the contents of the entry from the interrupt queue are moved into the AQ register. The entry from the interrupt queue contains the level number. If there is no unmasked queue from an entry, then zeros are moved into the AQ register.

The SCU interrupt-connect mask register (ICMR) allows masking of each port's interrupts and connects. Queues are maintained in the SCU for each of the eight interrupt levels. The queues are circular, first-in/first-out priority. No CPU address information is used.

ILLEGAL ADDRESS MODIFICATION: DU, DL, CI

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: Zero - If C(A) = 0, then ON; otherwise OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise OFF

NOTES:

1. An IPR fault occurs if this instruction is executed in Master or Slave mode.
2. An IPR fault occurs if illegal address modification or an illegal repeat is used.
3. Bound checks on the address are not made.

RMID	Read Memory ID Register	273 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(Memory ID Register) --> C(AQ)

EXPLANATION: This instruction provides program access to the memory ID register. SCU selection is based on the Control SCU bit (22) in the CPU mode register.

Address development is followed and transferred to the SCU to select the correct memory unit. The physical memory unit that is selected by the address is dependent upon the SCU's physical ID or logical ID based on the setting of the SCU configuration register.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if execution is attempted in the Slave or Master mode.

2. An IPR fault occurs if illegal address modification or an illegal repeat is used.

RMR	Read Memory Register	270 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(Memory Status Register) --> C(AQ)

0...0 --> C(Memory Status Register)

EXPLANATION: This instruction provides program access to the memory status register. This register consists of 8 bits (40-47) in a 72-bit register. (Refer to "Memory Error Status Register" in Section 4.) SCU selection is based on the control SCU bit in the CPU mode register.

Address development is followed and transferred to the SCU to select the memory unit. The memory unit is selected by physical ID or logical ID based on the setting of the SCU configuration register.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if execution is attempted in the Slave or Master mode.

2. An IPR fault occurs if illegal address modification or an illegal repeat is used.

RPAT	Run PATROL	611 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode and Master mode; NS mode only

EXPLANATION: This instruction operates like the DIS instruction. When PATROL is enabled, a full cycle of all test pages is run. The sampling for interrupts at completion of each test page is not done. Upon completion of the full cycle, the CPU returns to the execution of the next instruction.

When PATROL is disabled, no operation takes place. The CPU continues with the next instruction.

ILLEGAL ADDRESS MODIFICATIONS: None. Modification is performed, including modification of any indirect words specified. However, the effective address has no effect on the operation, including the final value of the instruction counter.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. A Command fault occurs if execution is attempted in the Slave or Master modes.
2. An IPR fault occurs if this instruction is executed in ES mode.
3. An IPR fault occurs if illegal repeats are executed.

RPD	Repeat Double	560 (0)

FORMAT:

0 0	0 7	0 8	0 9	1 0	1 1		1 7	1 8	Op Code	2 6	2 7	2 8	2 9	3 0	3 5
TALLY		A	B	C		TERM. COND.			560(0)		0	1	0	DELTA	

OPERATING MODES: Executes in NS mode only

CODING FORMAT: RPD N,I,k1,k2,...,k7. (A=B=C=1.) The command generated by the assembler from this format will cause the two instructions immediately following the RPD instruction to be iterated N times and the effective addresses of those two instructions to be incremented by the value I for each of N iterations. The meaning of the termination conditions of k1,k2,...,k7 are the same as for the RPT instruction. Since the repeat-double must fall in an odd location, the assembler will force this condition and a NOP instruction is used for a filler when needed.

RPDX ,I. (A=B=C=0.) This instruction operates just as the RPD instruction with the exception that A,B,N and the conditions for termination are loaded by the user into index register zero.

RPDA N,I,k1,k2,...,k7. (A=C=1. B=0.) This instruction operates just as the RPD instruction with the exception that only the effective address of the first instruction following the RPDA instruction will be incremented by the value of I for each of N iterations.

RPDB N,I,k1,k2,...,k7. (A=0. B=C=1.) This instruction operates just as the RPD instruction with the exception that only the effective address of the second instruction following the RPDB instruction will be incremented by the value I for each of N iterations.

EXPLANATION: The instructions from the next Y-pair are fetched and saved in the processor; they are executed repeatedly until a specified termination condition is met.

RPD

1. The RPD instruction must be stored in an odd memory location except when accessed via the XEC or XED instructions. In this case, the RPD instruction can be either even or odd, but the XEC or XED instruction must be in an odd memory location.

2. If C = 0, the tally and terminate conditions are loaded from X0/GX0.

   NS Mode

   Tally, terminate condition = $C(X0)_{0-17}$

   ES Mode

   Tally, terminate condition = $C(GX0)_{18-35}$
   $C(GX0)_{0-17}$ unchanged

3. If C = 1, then bits 0-17 of the RPD instruction are loaded into X0/GX0.

   NS Mode

   Bits 0-17 of the RPD instruction --> C(X0/GX0)

   ES Mode

   Bits 0-17 of the RPD instruction --> $C(GX0)_{18-35}$

   00....0 --> $C(GX0)_{0-17}$

4. The terminate condition(s) and tally from X0 control the repetition for the instructions following the RPD instruction. An initial tally of zero is interpreted as 256. A fault also causes an exit from the cycle.

5. The repetition cycle consists of the following steps:

   a. Execute the pair of repeated instructions.

   b. $C(X0)_{0-7}$ - 1 --> bits 0-7 of C(X0)

      or $C(GX0)_{18-25}$ - 1 --> $C(GX0)_{18-25}$

   c. If a terminate condition is met, set the Tally Runout indicator OFF and exit.

RPD

d. If bits 0-7 of C(X0) or bits 18-25 of C(GX0) = 0, set the Tally Runout indicator ON and exit.

e. If conditions in c. or d. are not met, go to a.

6. Many instructions cannot be repeated. If an instruction cannot be repeated, an illegal repeat causes on IPR fault to occur. Refer to the individual instruction descriptions to determine whether or not a particular instruction can be repeated.

7. Address modification for the pair of repeated instructions is as follows.

   For each of the two repeated instructions, only the modifiers R and RI and only the designators specifying X1,...,X7/GX1,...,GX7 are permitted. Address register modification is also permitted. All other modifier designations result in an IPR fault.

   When the effective address for R modification is Y, and when the indirect word address for RI modification is YI, the address is determined as follows.

   a. When AR modification is not indicated (bit 29 = 0)

      o For the first execution of each of the two repeated instructions:

        $Y + C(R) \rightarrow Y_1$ or YI_1

        Y_1 or $YI_1 \rightarrow C(R)$

      o For any subsequent execution of the two repeated instructions:

        For the first instruction of the pair

          If A=1, then DELTA + $C(R) \rightarrow Y_n$ or YI_n

          Y_n or $YI_n \rightarrow C(R)$

          If A=0, then $C(R) \rightarrow Y_n$ or YI_n, where n>1

        For the second instruction of the pair

          If B=1, then DELTA + $C(R) \rightarrow Y_n$ or YI_n;

          Y_n or $YI_n \rightarrow C(R)$

RPD

If B=0, then C(R) --> Y_n or YI_n, where n>1

b. When AR modification is indicated (bit 29 = 1)

- For the first execution of each of the two repeated instructions:

  (se)Y + C(R) + C(ARm) --> Y_1 or YI_1

  (se)Y + C(R) --> C(R)

  (se) is the extended address with bit 3 of y.

  ARm is the address register m selected by instruction bits 0, 1, 2.

- For any subsequent execution of the two repeated instructions:

  For the first instruction of the pair

  If A=1, then DELTA + C(R) + C(ARm) --> Y_n or YI_n;

  DELTA + C(R) --> C(R)

  If A=0, then C(R) + C(AR) --> Y_n or YI_n

  For the second instruction of the pair

  If B=1, then DELTA + C(R) + C(ARm) --> Y_n or YI_n

  DELTA + C(R) --> C(R)

  If B=0, then C(R) + C(ARm) --> Y_n or YI_n

  A and B are the contents of the X0 bits 8 and 9 or the GX0 bits 26 and 27.

RPD

When RI modification is specified in the repeated instruction, indirect reference is performed only once for each repeat. The tag field of the indirect word is ignored and processed as R modification (R = N).

8. The Exit Conditions:

An exit is made from the repeat cycle if one of the terminate conditions exists or if tally = 0 after the execution of the odd instruction of the repeated pair. Also, an exit is made when a fault occurs.

The program-controlled exit conditions are:

a. Tally = 0

b. Terminate Conditions:

The bit configuration in bit positions 11-17 of the RPD instruction defines the terminate conditions. If more than one condition is specified, the repeat terminates if any one of the specified conditions is met.

The carry, negative, and zero indicators each use two bits, one for the OFF condition and one for ON. A zero in both positions for one indicator causes this indicator to be ignored as a terminate condition. A 1 in both positions causes an exit after the first execution of the repeated instruction pair.

Bit 17 = 0: Ignore all overflows. The respective overflow indicator is not set ON, and an Overflow fault does not occur.

Bit 17 = 1: Process overflows. If overflow mask indicator is ON when an overflow occurs, then exit from the repetition cycle. If the overflow mask indicator is OFF when an overflow occurs, then an Overflow fault occurs.

RPD

Bit 16 = 1: Terminate if carry indicator is OFF.

Bit 15 = 1: Terminate if carry indicator is ON.

Bit 14 = 1: Terminate if negative indicator is OFF.

Bit 13 = 1: Terminate if negative indicator ON.

Bit 12 = 1: Terminate if zero indicator OFF.

Bit 11 = 1: Terminate if zero indicator ON.

c. Overflow Fault:

If bit 17 = 1 and an overflow occurs with the overflow mask indicator OFF, an Overflow fault occurs and an exit is made from the repetition cycle after the execution of the current instruction when the fault processor returns control.

A non-program-controlled exit from the repetition cycle occurs if any fault other than an Overflow occurs. If any fault (Overflow, Divide Check, Parity error on indirect word or operand fetch, etc.) occurs on the even instruction, the odd instruction will not be executed.

9. Status at termination of repeat

Bits 0-7 of C(X0) or bits 18-25 of C(GX0) contain the tally residue (i.e., the number of repeats remaining until a tally runout would have occurred). The terminate conditions in bits 11-17 remain unchanged.

If the exit was due to tally = 0 or a terminate condition, the Xn/GXn specified by the designator of each of the two repeated instructions will contain either:

a. The contents of the designated Xn/GXn after the last execution of the repeated pair plus the DELTA associated with each instruction, as A or B, the DELTA designators (bits 8 and 9 of X0) = 1, or

b. The contents of the designated Xn/GXn after the last execution of the repeated pair if A or B, respectively, is zero.

10. When $X0_{0-7}/GX0_{18-25}$ contain zeros and the terminate condition is not satisfied, the tally runout indicator set to ON; otherwise, it is set to OFF.

ILLEGAL ADDRESS MODIFICATIONS: None. Address modification is not executed. Bit 29 is ignored.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: The RPD instruction itself does not affect any of the indicators. However, the execution of the repeated instructions may affect indicators. The repeat mode entered as a result of the instruction affects the Tally Runout indicator.

NOTES:

1. A repeat-double of instructions that have long execution times may cause a Lockup fault (LUF) if the time involved is greater than the lockup time interval, which may be 2, 4, 8, or 16 milliseconds.

2. The repeated instruction must be modified by an index register.

3. The following conditions cause an IPR fault to occur.

   - o If illegal repeats are used.
   - o If the repeated instruction uses X0/GX0.
   - o If R or RI modification is attempted with the repeated instruction with other than X1-X7/GX1-GX7.
   - o If the RPD instruction (or the XEC instruction accessing the RPD instruction) is not at an odd location.

RPD

If the exit was due to a fault, the Xn/GXn specified by the designator of each of the two repeated instructions may contain either:

a. The contents of the designated Xn/GXn when the fault occurred plus the DELTA associated with each instruction A and B = 1, or

b. The contents of the designated Xn/GXn when the fault occurred.

EXAMPLE:

```
1       8       16
        EAX6    FROM
        EAX7    TO
        RPD     100,2
        LDAQ    0,6
        STAQ    0,7
         .
         .
         .
        EVEN
FROM    BSS     200
TO      BSS     200
```

RPL	Repeat Link	500 (0)

FORMAT:

0 0	0 7	0 8	0 9	1 0	1 1	1 7	1 8	Op Code	2 6	2 7	2 8	2 9	3 0	3 5
TALLY		0	0	C	TERM. COND.			500(0)		0	1	0	000000	

OPERATING MODES: Executes in NS mode only

CODING FORMAT: RPL N,k1,k2,...,k7. (C = 1.) This format causes the instruction immediately following the RPL instruction to be repeated N times or until one of the conditions specified in k1,...,k7 is satisfied, or until the link address of zero is detected. The range of N is 0-255. If N = 0, the instruction will be iterated 256 times. If N is greater than 255, the instruction will cause an error flag (A) to be printed on the assembly listing. The fields k1, k2, ..., k7 may or may not be present. They represent conditions for termination which, when needed, are declared by the conditional transfer instructions TMI, TNC, TNZ, TOV, TPL, TRC, and TZE. These instructions affect the termination condition bits in position 11-17 of the Repeat instruction.

An octal number can be used rather than the transfer instructions to denote termination conditions. Thus, if the field for k1, k2, ..., k7 is found to be numeric, it will be interpreted as octal, and the low-order 7 bits will be ORed into bit positions 11-17 of the Repeat instruction. The variable field scan is terminated with the octal field.

RPLX (C = 0). This instruction operates just as the RPL instruction except that N and the conditions for termination are loaded by the user into index register zero.

EXPLANATION: The next instruction is executed either a specified number of times until a specified termination condition is met, or until the link address of zero is detected.

1. If C = 0, the tally and terminate conditions are those loaded from X0/GX0.

   NS Mode

   Tally, terminate condition = $C(X0)_{0-17}$

   ES Mode

   Tally, terminate condition = C(GX0)18-35

   $C(GX0)_{0-17}$ is unchanged

2. If C = 1, then bits 0-17 of the RPL instruction--> C(X0)/(GX0)

   NS Mode

   Bits 0-17 of the RPD instruction --> C(X0/GX0)

   ES Mode

   Bits 0-17 of the RPD instruction --> $(GX0)_{18-35}$

   00...0 --> $C(GX0)_{0-17}$.

3. The terminate condition(s) and tally from X0 control the repetition for the instruction following the RPL instruction. An initial tally of zero is interpreted as 256. A fault also causes an exit from the cycle.

4. The repetition cycle consists of the following steps:

   a. Execute the repeated instruction.

   b. $C(X0)_{0-7}$ -1 --> bits 0-7 of C(X0)

      or $C(GX0)_{18-25}$ -1 --> $C(GX0)_{18-35}$.

   c. If a terminate condition is met, set the Tally Runout indicator OFF and exit.

   d. If bits 0-7 of C(X0) or bits 18-25 of C(GX0) = 0, or the link address bits 0-17 of C(Y) = 0 and no terminate condition is met, set the Tally Runout indicator ON and exit.

   e. If conditions in c. or d. are not met, the effective address C(Y) is used as a link address to determine the C(Y) to be used in the next iteration. Go to a.

5. Many instructions cannot be repeat-linked. If an instruction cannot be repeated, an illegal repeat causes an IPR fault to occur. Refer to the individual instruction descriptions to determine whether or not a particular instruction can be repeated.

6. Address modification for the repeated instruction is as follows.

   Only address register (AR) modification and R modification specifying X1-X7/GX1-GX7 are permitted for repeated instructions.

   R modification is valid only for the first execution of the repeated instruction, AR modification is valid for all executions.

   The effective address is generated as follows.

   a. When AR modification is not indicated (bit 29 = 0)

      - For the first execution of the repeated instruction

        Y1 = y1 + C(R)

        Y1 --> C(R)

      - For each successive execution of the repeated instruction

        $Yn = C(Yn-1)_{0-17}$

        Yn --> C(R) (when Yn_{0-17} does not contain zeros)

b. When AR modification is indicated (bit 29 = 1)

- o For the first execution of the repeated instruction

  Y1 = (se)y + C(R) + C(ARm)

  (se)y + C(R) --> C(R)

  (se)y is the extended address with bit 3 of y.

  ARm is the address register m selected by instructions bits 0, 1, 2.

- o For each successive execution of the repeated instruction

  $Yn = C(Yn-1)_{0-17} + C(AR)$

  $C(Yn-1)_{0-17}$ --> C(R)

  when Yn_{0-17} does not contain zeros

The effective address Y is the address of the next list word. The lower portion of the list word contains the operand to be used for this execution of the repeated instruction.

The operand is handled in one of the following formats.

Bits 0-17: 00...0

Bits 18-35: $C(Y)_{18-35}$ for single-precision (1 word)

or as

Bits 0-17: 00...0

Bits 18-71: $C(Y)_{18-71}$ for double precision (2 words)

The upper 18 bits of the list word contain the link address; that is, the address of the next successive list word, and thus the effective address for the next successive execution of the repeated instruction.

RPL

7. Repeat Exit Conditions:

An exit is made from the repeat cycle if one of the terminate conditions exists or if tally = 0 or link address = 0 after the execution of the repeated instruction. Also, an exit is made when a fault occurs.

The program-controlled exit conditions are:

a. Tally = 0

b. Link Address = 0

c. Terminate Conditions:

The bit configuration in bit positions 11-17 of the RPL instruction defines the terminate conditions. If more than one condition is specified, the repeat terminates if any of the specified conditions is met.

The carry, negative, and zero indicators each use two bits, one for the OFF condition and one for ON. A zero in both positions for one indicator causes this indicator to be ignored as a terminate condition. A 1 in both positions causes an exit after the first execution of the repeated instruction.

Bit 17 = 0: Ignore all overflows. The respective overflow indicator is not set ON, and an Overflow fault does not occur.

Bit 17 = 1: Process overflows. If the overflow mask indicator is ON when an overflow occurs, exit from the repetition cycle. If the overflow mask indicator is OFF when an overflow occurs, then an Overflow fault occurs.

Bit 16 = 1: Terminate if carry indicator is OFF.

Bit 15 = 1: Terminate if carry indicator is ON.

Bit 14 = 1: Terminate if negative indicator is OFF.

Bit 13 = 1: Terminate if negative indicator is ON.

Bit 12 = 1: Terminate if zero indicator is OFF.

Bit 11 = 1: Terminate if zero indicator is ON.

d. Overflow Fault:

If bit 17 = 1 and an overflow occurs with the overflow mask indicator OFF, an Overflow fault occurs and an exit is made from the repetition cycle when the fault processor returns control.

A non-program-controlled exit from the repetition cycle occurs if any fault other than an Overflow occurs (Divide Check, Parity error on indirect word or operand fetch, etc.).

8. Status at termination of repeat

Bits 0-7 of C(X0) or bits 18-25 of C(GX0) contain the tally residue (i.e., the number of repeats remaining until a tally runout would have occurred). The terminate conditions in bits $X0_{11-17}/GX0_{29-35}$ remain unchanged.

The Xn/GXn specified by the designator of the repeated instruction contains the address of the list word that contains:

a. In its lower-half, the operand used in the last execution of the repeated instruction

b. In its upper-half, the address of the next list word.

9. When $X0_{0-7}/GX0_{18-25}$ contain zeros, or when the link address $(Y)_{0-17}$ contains zeros, and the terminate condition is not satisfied, the Tally runout indicator is set to ON; otherwise, it is set to OFF.

10. An exit will not occur if the effective address is 0 for the first execution of the linked instruction. This address specifies the location of the first word in the link table and is not interpreted as a link address.

ILLEGAL ADDRESS MODIFICATIONS: None. Address modification is not executed. Bits 29-35 are ignored.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: The RPL instruction itself does not affect any of the indicators. However, the execution of the repeated instruction may affect the indicators. The repeat mode entered as a result of the instruction affects the tally runout indicator.

NOTES:

1. The repeated instruction must be modified by an index register.

2. The following conditions cause an Illegal Procedure fault.

   - o If illegal repeats are used.
   - o If the repeated instruction uses X0/GX0.
   - o If other than AR or R modification is attempted with the repeated instruction.
   - o If R modification other than X1-X7/GX1-GX7 is attempted with the repeated instruction.

EXAMPLE:

```
1       8       16
        EAX7    A
        LDQ     =0777777,DU
        LDA     =3HIDD,DL
        RPL     5,TZE
        CMK     0,7
        TNZ     ERROR
        .
        .
        .
A       VFD     18/B,H18/IDA
        .
        .
B       VFD     18/C,H18/IDB
        .
        .
C       VFD     18/D,H18/IDC
        .
        .
D       VFD     18/E,H18/IDD
        .
        .
E       VFD     18/0,H18/IDE
```

RPT	Repeat	520 (0)

FORMAT:

0 0	0 7	0 8	0 9	1 0	1 1	1 7	1 8	Op Code	2 6	2 7	2 8	2 9	3 0	3 5
TALLY		0	0	C	TERM. COND.			520(0)			I		DELTA	

OPERATING MODES: Executes in NS mode only

CODING FORMAT: RPT N,I,k1,k2,...,k7. (Bit C=1.) The command generated by the assembler from this format will cause the instruction immediately following the RPT instruction to be iterated N times and that instruction's effective address to be incremented by the value I for each of N iterations. The range for N is 0-255. If N = 0, the instruction will be iterated 256 times. If N is greater than 256, the instruction will cause an error flag (A) to be printed on the assembly listing. The fields k1,k2,...k7 may or may not be present. They represent conditions for termination which, when needed, are declared by the conditional transfer instructions TMI, TNC, TNZ, TOV, TPL, TRC, and TZE. These instructions affect the termination condition bits in positions 11-17 of the Repeat instruction. See discussion of terminate conditions below.

In addition, an octal number can be used rather than the transfer instructions to denote termination conditions. Thus, if the field for k1,k2...,k7 is found to be numeric, it will be interpreted as octal and the low-order 7 bits will be ORed into bit positions 11-17 of the Repeat instruction. The variable-field scan will be terminated with the octal field.

RPTX ,I (Bit C = 0). This instruction operates just as the RPT instruction with the exception that N and the conditions for termination are loaded by the user into bit positions 0-7 and 11-17, respectively, of index register zero (instead of being embedded in the instruction).

EXPLANATION: The next instruction is executed either a specified number of times or until a specified termination condition is met.

1. If C = 0, the tally and terminate conditions are those loaded from X0/GX0.

   NS Mode

   Tally, terminate condition = $C(X0)_{0-17}$

   ES Mode

   Tally, terminate condition = $C(GX0)_{18-35}$

   $C(GX0)_{0-17}$ unchanged

2. If C = 1, then bits 0-17 of the RPT instruction are loaded into C(X0)/(GX0).

   NS Mode

   Bits 0-17 of the RPT instruction --> C(X0/GX0)

   ES Mode

   Bits 0-17 of the RPT instruction --> $(GX0)_{18-35}$
   00----0 --> $C(GX0)_{0-17}$

3. The terminate condition(s) and tally from X0 control the repetition for the instruction following the RPT instruction. An initial tally of zero is interpreted as 256. A fault also causes an exit from the cycle.

4. The repetition cycle consists of the following steps:

   a. Execute the repeat instruction.

   b. $C(X0)_{0-7}$ - 1 --> bits 0-7 of C(X0)

      or $C(GX0)_{18-25}$ - 1 --> $C(GX0)_{18-25}$

   c. If a terminate condition is met, set the tally runout indicator OFF and exit.

   d. If bits 0-7 of C(X0) or bits 18-25 of C(GX0) = 0, set the tally runout indicator ON and exit.

   e. If conditions in c. or d. are not met, go to a.

5. Many instructions cannot be repeated. For such instructions, an illegal repeat causes an IPR fault to occur. Refer to the individual instruction descriptions to determine whether or not a particular instruction can be repeated.

6. Address modification for the repeated instruction is as follows.

   For the repeated instruction, only the modifiers R and RI and only the designators specifying X1,...,X7/GX1,...,GX7 are permitted. Address register modification is also permitted.

   All other modifier designations result in an IPR fault.

   When the effective address for R modification is Y, and when the indirect word address for RI modification is YI, the address are determined as follows.

   When AR modification is not indicated (bit 29 = 0)

   a. For the first execution of the repeated instruction:

      $Y + C(R) \rightarrow Y_1$ or YI_1

      Y_1 or $YI_1 \rightarrow C(R)$

   b. For each successive execution of the repeated instruction

      $DELTA + C(R) \rightarrow Y_n$ or YI_n

      Y_n or $YI_n \rightarrow C(R)$

      DELTA is bits 30 to 35 of the RPT instruction.

RPT

When AR modification is indicated (bit 29 = 1)

a. For the first execution of the repeated instruction

(se)Y + C(R) + C(ARm) --> Y_1 or YI_1

(se)Y + C(R) --> C(R)

(se) is the extended address with bit 3 of y.
ARm is the address register m selected by instruction bits 0, 1, 2.

b. For any subsequent execution of the the repeated instruction

DELTA + C(R) + C(ARm) --> Y_n or YI_n

DELTA + C(R) --> C(R)

When RI modification is specified in the repeated instruction, indirect reference is performed only once for each repeat. The tag field of the indirect word is ignored and processed as R modification (R = N).

7. Repeat Exit Conditions:

An exit is made from the repeat cycle if one of the terminate conditions exists or if tally = 0 after the execution of the odd instruction of the repeated pair. Also, an exit is made when a fault occurs.

The program-controlled exit conditions are:

a. Tally = 0

RPT

b. Terminate Conditions:

The bit configuration in bit positions 11-17 of the RPT instruction defines the terminate conditions. If more than one condition is specified, the repeat terminates if any of the specified conditions is met.

The carry, negative, and zero indicators each use two bits, one for the OFF condition and one for ON. A zero in both positions for one indicator causes this indicator to be ignored as a terminate condition. A 1 in both positions causes an exit after the first execution of the repeated instruction pair.

Bit 17 = 0: Ignore all overflows. The respective overflow indicator is not set ON, and an Overflow fault does not occur.

Bit 17 = 1: Process overflows. If the overflow mask indicator is ON when an overflow occurs, exit from the repetition cycle. If the overflow mask indicator is OFF when an overflow occurs, an Overflow fault occurs.

Bit 16 = 1: Terminate if carry indicator is OFF.

Bit 15 = 1: Terminate if carry indicator is ON.

Bit 14 = 1: Terminate if negative indicator is OFF.

Bit 13 = 1: Terminate if negative indicator is ON.

Bit 12 = 1: Terminate if zero indicator is OFF.

Bit 11 = 1: Terminate if zero indicator is ON.

c. Overflow Fault:

If bit 17 = 1 and an overflow occurs with the Overflow Mask indicator OFF, an Overflow fault occurs and an exit is made from the repetition cycle when the fault processor returns control.

A non-program-controlled exit from the repetition cycle occurs if any fault other than Overflow occurs.

8. Status at termination of repeat

   Bits 0-7 of C(X0) or bits 18-25 of C(GX0) contain the tally residue (i.e., the number of repeats remaining until a tally runout would have occurred). The terminate conditions in bits 11-17 remain unchanged.

   If the exit was due to tally = 0 or a terminate condition, the Xn/GXn specified by the designator of the repeated instruction will contain:

   The contents of the designated Xn/GXn after the last execution of the repeated instruction plus the DELTA associated with each instruction.

   If the exit was due to a fault, the Xn/GXn specified by the designator of the repeated instruction may contain one of the following.

   - The contents of the designated Xn/GXn when the fault occurred plus the DELTA
   - The contents of the designated Xn/GXn when the fault occurred

9. When $X0_{0-7}/GX0_{18-25}$ contain zeros and the terminate condition is not satisfied, the tally runout indicator it set ON; otherwise, it is set OFF.

ILLEGAL ADDRESS MODIFICATIONS: None. Address modification is not executed. Bit 29 is ignored.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: The RPT instruction itself does not affect any of the indicators; however, the execution of the repeated instruction may affect indicators. The repeat mode entered as a result of the instruction affects the Tally Runout indicator.

NOTES:

1. The repeated instruction must be modified by an index register.

2. The follcwing conditions cause an IPR fault to occur.

   - o If illegal repeats are used.
   - o If the repeated instruction uses X0/GX0.
   - o If R or RI modification is attempted with the repeated instruction with other than X1-X7/GX1-GX7.
   - o If other than R or RI modification or AR modification are attempted with the repeated instruction.

EXAMPLE:

```
1       8       16
        LDA     KEY
        EAX4    TABLE
        RPT     64,1,TZE
        CMPA    0,4
        TZE     FOUND
        .
        .
        .
TABLE   BSS     64
KEY     BSS     1
```

RSCR	Read System Controller Register	413 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(AQ) --> C(SCU Register)

EXPLANATION: This instruction provides program access to all system controller registers. SCU selection is based upon the control SCU bit in the CPU mode register. Address development is followed, and is transferred to the SCU to select the general register. In Slave mode, the final address is forced to reference the calendar clock.

In VMS Privileged Master mode, if both SCU ports are enabled and the least-significant bit of the effective address (word address) is 1, the control SCU bit is temporarily changed to permit selection of the non-control SCU. (Reference Section 4 for CPU configuration register and ASR control.) The control SCU bit is then reset to its original value.

Real Memory Address:

0...21	Bits 22-24	25-27	Function
X...X	0	X	Not used
X...X	1	X	Configuration
X...X	2	X	Fault
X...X	3	X	History
X...X	4	X	Calendar Clock
X...X	5	X	Not used
X...X	6	X	Syndrome
X...X	7	X	Not used

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: None affected

NOTES:

1. A Command fault occurs if address bits 22-24 are 0, 5, or 7 (octal).

2. A Command fault occurs if execution is attempted in Slave or Master mode.

3. The SCU registers are defined in Section 4.

4. Bits 25-27 of the configuration register are the SCU port number. These bits must be zero in an SSCR instruction, in order that a subsequent RSCR instruction returns the port number; otherwise, the OR of bits 25-27 and the port number are returned.

5. An IPR fault occurs if illegal address modification or illegal repeats are executed.

RSW	Read Processor Model Characteristics	231 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(model char.) --> C(A)

0...0 --> $C(A)_{0-3}$

Processor type --> $C(A)_{4-6}$ (DPS 8000 type = 101)

Test Mode Register Bit 13 (Transfer Trace Mode) --> $C(A)_{30}$
1 = enable

Performance submodel type --> $C(A)_{31-32}$

CPU Number --> $C(A)_{33-35}$

EXPLANATION: This instruction reads system model characteristics previously set by the firmware and loads them into the A register.

The submodel field is interpreted as follows:

$C(A)_{31-32}$	Performance
00	1,5
01	2.3
10	3.0
11	Undefined

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, RI, IR, IT

ILLEGAL REPEATS: None

INDICATORS: None affected

NOTES:

1. Address development occurs but has no effect on the execution of this instruction.
2. Additional model characteristics may be defined by the firmware.
3. An IPR fault occurs if illegal address modification is executed.

S4BD
S4BDX

S4BD S4BDX	Subtract 4-Bit Displacement from Address Register	522 (1)

FORMAT: Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

```
1       8       16
                  {S4BD }
                  {S4BDX} word displacement,R,AR
```

OPERATING MODES: Any

EXPLANATION: Description is the same as for A4BD except that y and C(DR) are added and the sum is subtracted from the content of AR<u>n</u>.

When the mnemonic is coded with an X (S4BDX), bit 29 is forced to 0. If bit 29 is 0, the content of AR<u>n</u> is assumed as 0.

ILLEGAL ADDRESS MODIFICATIONS: If DU, DL, or IC are specified in DR.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES: Applies to NS mode only

```
1       8       16              32
        EAX3    10
        S4BDX   2,3,4           AR4 octal contents  -  7 7 7 7 7 4 6 0
        S4BD    0,3,4           AR4 octal contents  -  7 7 7 7 7 3 4 0

        EAX6    7
        S4BDX   3,6,2           AR2 octal contents  -  7 7 7 7 7 4 0 5
        S4BD    0,6,2           AR2 octal contents  -  7 7 7 7 7 3 2 0
```

S6BD
S6BDX

S6BD S6BDX	Subtract 6-Bit Displacement from Address Register	521 (1)

FORMAT: Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

```
1       8       16
_______________________________

        {S6BD }
        {S6BDX} word displacement,R,AR
```

OPERATING MODES: Any

EXPLANATION: Description is the same as for A6BD except that y and C(DR) are added and the sum is subtracted from the content of ARn.

When the mnemonic is coded with an X (S6BDX), bit 29 is forced to zero. If bit 29 is 0, the content of ARn is assumed as 0.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, or IC specified in DR.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES: Applies to NS mode only

```
1       8       16              32
_____________________________________________

        EAX5    14
        S6BDX   0,5,2           AR2 octal contents - 7 7 7 7 7 5 4 6
        S6BD    2,5,2           AR2 octal contents - 7 7 7 7 7 1 2 3

        EAX6    5
        S6BDX   1,6,7           AR7 octal contents - 7 7 7 7 7 6 0 5
        S6BD    0,6,7           AR7 octal contents - 7 7 7 7 7 5 2 3
```

S9BD
S9BDX

S9BD S9BDX	Subtract 9-Bit Displacement from Address Register	520 (1)

FORMAT: Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

```
1       8       16
                        {S9BD }
                        {S9BDX} word displacement,R,AR
```

OPERATING MODES: Any

SUMMARY: Description is the same as for A9BD except that y and C(DR) are added and the sum is subtracted from the content of ARn.

When the mnemonic is coded with an X (S9BDX), bit 29 is forced to zero. If bit 29 is 0, the content of ARn is assumed as 0.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, or IC specified in DR.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

EXAMPLES: Applies to NS mode only

```
1       8       16              32
        EAX7    9
        S9BDX   1,7,5           AR5 octal contents - 7 7 7 7 7 4 6 0
        S9BD    1,,5            AR5 octal contents - 7 7 7 7 7 3 6 0

        EAX2    7
        S9BDX   2,2,6           AR6 octal contents - 7 7 7 7 7 4 2 0
        S9BD    0,2,6           AR6 octal contents - 7 7 7 7 7 2 4 0
```

SARn	Store Address Register n	74n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        SARn    LOCSYM,R,AM
```

OPERATING MODES: Any

SUMMARY: NS Mode

For n=0,1,..,7 as determined by op code

C(ARn) --> $C(Y)_{0-23}$; $C(Y)_{24-35}$, C(ARn) unchanged

ES Mode

For n=0,1,..,7 as determined by op code

C(ARn) --> C(Y), C(ARn) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES: Applies to NS mode only

```
1       8       16              32
        SAR5    ADDRWS          0 0 1 7 5 0 2 7  AR5 contents
         .
         .
         .
ADDRWS  BSS     1               0 0 1 7 5 0 2 7 x x x x  memory after
```

SAREG	Store Address Registers	443 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        SAREG   LOCSYM,R,AR
```

OPERATING MODES: Any

SUMMARY: NS Mode

$C(AR0,AR1,...,AR7) \rightarrow C(Y,Y+1,...,Y+7)_{0-23}$

$Zeros \rightarrow C(Y,Y+1,...,Y+7)_{24-35}$

ES Mode

$C(AR0,AR1,...,AR7) \rightarrow C(Y,Y+1,...,Y+7)$

EXPLANATION: The lower 3 bits of Y are assumed as 000 and the 8 words beginning from the 8-word boundary are accessed for storage. No check is performed to determine whether the lower 3 bits of Y are actually 000.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

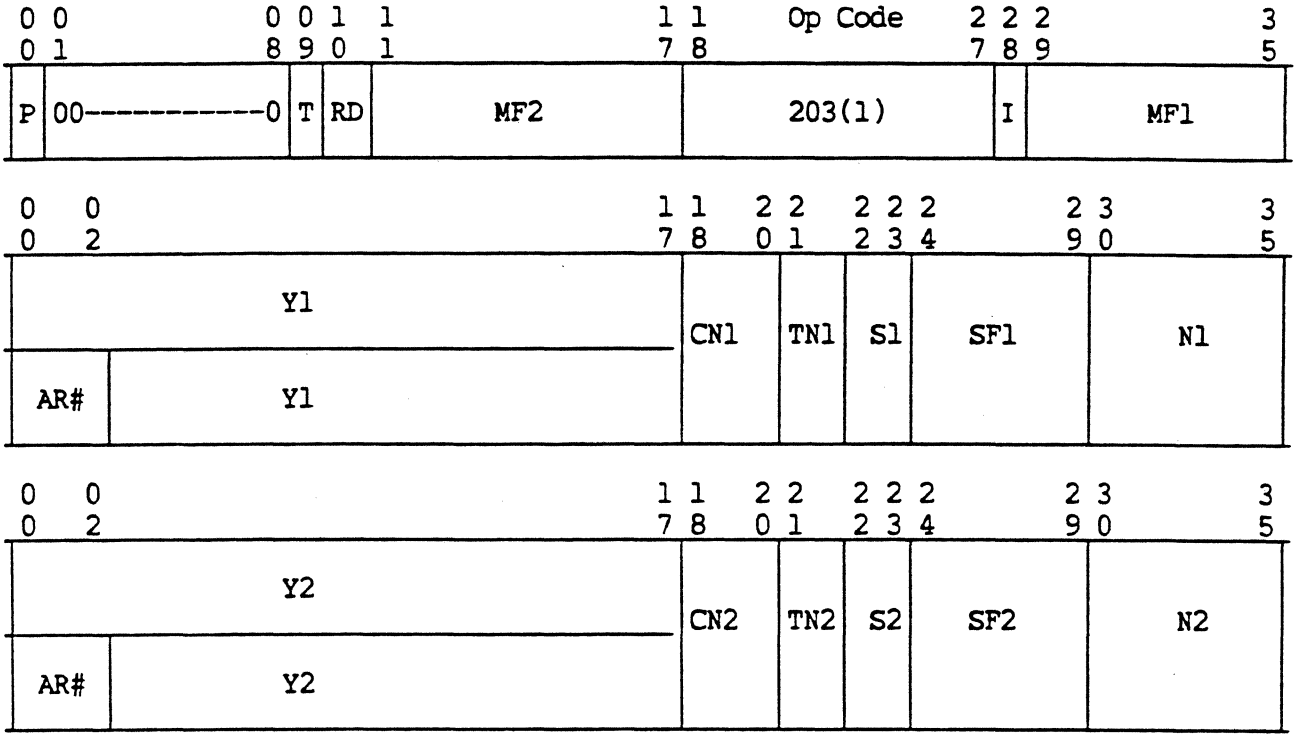
NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

```
1       8       16              32
        SAREG   REGWS
         .
         .
         .
        EIGHT
REGWS   BSS     8
```

SB2D	Subtract Using Two Decimal Operands	203 (1)

FORMAT:



CODING FORMAT: The SB2D instruction is coded as follows:

```
1       8       16
_______________________

SB2D    (MF1),(MF2),RD,P,T
NDSCn   LOCSYM,CN,N,S,SF,AM
NDSCn   LOCSYM,CN,N,S,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) - C(string 1) --> C(string 2)

EXPLANATION: Same as SB3D except that the difference is stored using YC2, TN2, S2, and, if S2 indicates a scaled format, SF2.

The zero indicator is set when the decimal number is zero; it does not indicate that all bits are zeros.

Refer to AD3D, for a description of justifying the scaling factors.

Independent of the data type being used (either packed decimal or 9-bit numeric; floating-point or scaled) significant digits in the result may be lost if:

1. The difference between the scaling factors (exponents) of the source operands is large enough to cause the expected length of the intermediate result to exceed 63 digits after decimal-point alignment of source operands, followed by subtraction.

2. The result field as defined by the result descriptor is not large enough to contain the calculated result after it has been aligned.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero - If result equals zero, then ON; otherwise, OFF

Negative - If result is negative, then ON; otherwise, OFF

Truncation - If, in the preparation of the final result, one or more least significant digits (zero or nonzero) are lost and rounding is not specified, then ON; otherwise (i.e., no least significant digits lost or rounding is specified), OFF

Exponent Overflow - If exponent of floating-point result is > 127, then ON; otherwise, unchanged

Exponent
Underflow - If exponent of floating-point result is < -128, then ON; otherwise, unchanged

Overflow - If fixed-point integer, or internal register overflow, then ON; otherwise, unchanged

NOTES:

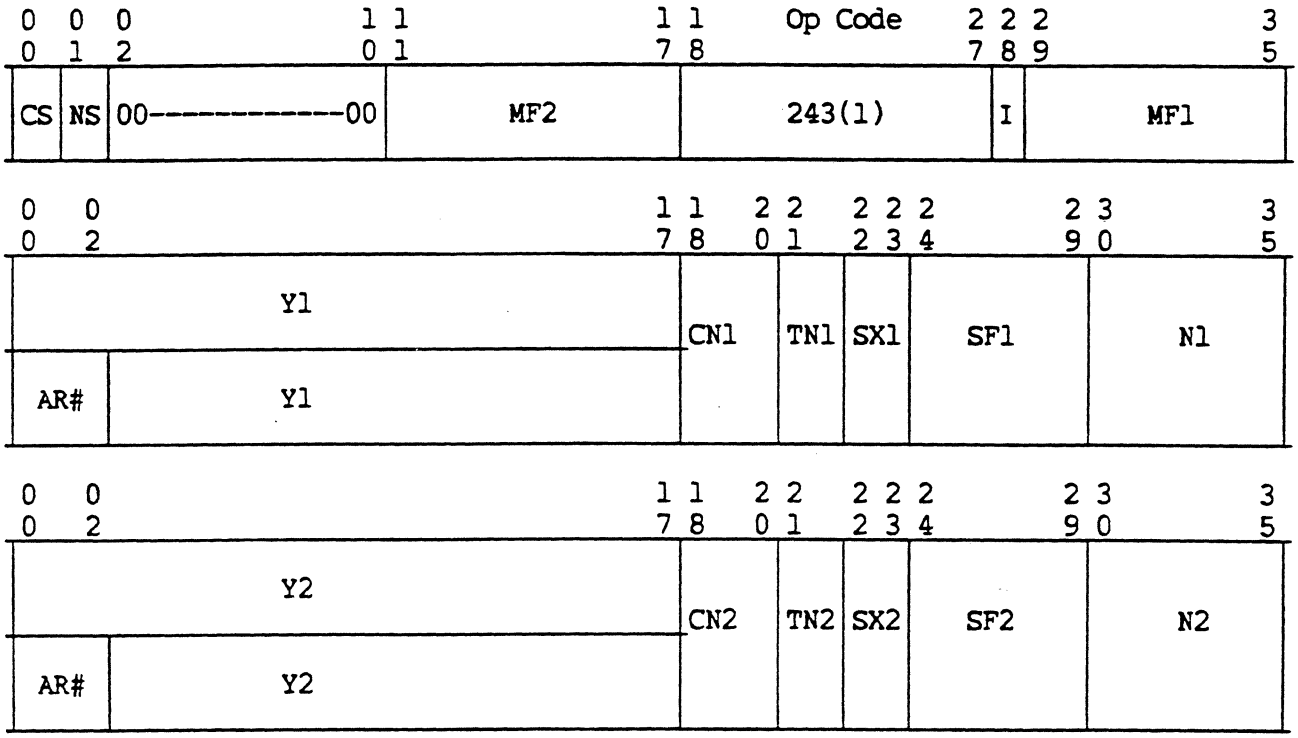
1. Truncation fault same as for AD3D.
2. Illegal Procedure fault same as for MVN.
3. If an illegal digit or sign is detected, part or all of the receiving field may be changed before the IPR fault occurs.

EXAMPLES: Applies to NS mode only

```
1       8       16              32
        SB2D    ,,1             with rounding option
        NDSC4   FLD1,0,4,2,-3   subtrahend operand descriptor
        NDSC9   FLD2,0,8        minuend operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    4P125+          1 2 5 +
FLD2    EDEC    8A+6543.21      + 6 5 4 3 2 1 -2
        USE                     + 6 5 4 3 0 9 -2  (Result)

        SB2D    ,,,1            with truncation enable option
        NDSC4   FLD1,0,8,3,-4   subtrahend operand descriptor
        NDSC9   FLD2,0,8,3,-2   minuend operand descriptor
        USE     CONST.          memory contents

FLD1    EDEC    8P12345678      12345678
FLD2    EDEC    8A87654321      87654321
        USE                     87530864  (Result)
*INSTRUCTION FAULT? YES     WHAT KIND?     truncation fault
```

SB2DX	Subtract Using Two Decimal Operands Extended	243 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
SB2DX  (MF1),(MF2),RD,CS,T,NS
NDSCn  LOCSYM,CN,N,SX,SF,AM
NDSCn  LOCSYM,CN,N,SX,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string2) - C(string1) --> C(string2)

EXPLANATION: Same as for SB3DX except that the difference is stored using YC2, TN2, SX2 and, if SX2 indicates a scaled format, SF2.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or MF2

ILLEGAL REPEATS: RPT, RPD, RPL

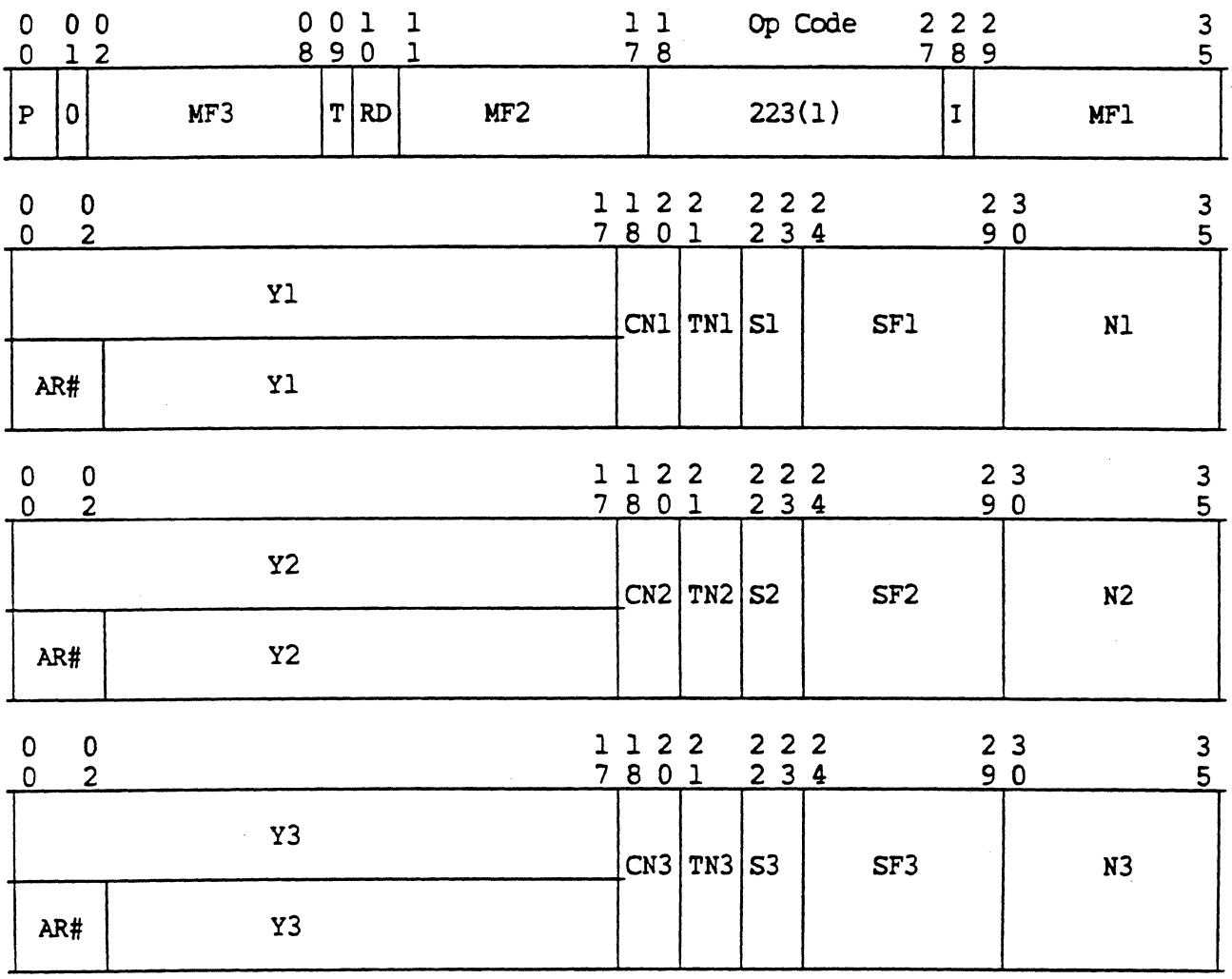
INDICATORS: Same as for AD3DX

NOTES:

1. All notes for AD3DX apply to SB2DX.
2. See MVNX for information about coding of overpunched signs.

SB3D	Subtract Using Three Decimal Operands	223 (1)

FORMAT:



CODING FORMAT: The SB3D instruction is coded as follows:

```
1       8       16
_______________________

        SB3D    (MF1),(MF2),(MF3),RD,P,T
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
        NDSCn   LOCSYM,CN,N,S,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) - C(string 1) --> C(string 3)

The decimal number of data type TN1, sign and decimal type S1, and starting location YC1, is subtracted from the decimal number of data type TN2, sign and decimal type S2, and starting location YC2. The difference is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type S3.

If S3 indicates a fixed-point format, the results are stored using scale factor SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most-significant-digit overflow or least-significant-digit truncation to occur.

If S3 indicates a floating-point format, the result is right-justified to preserve the most significant nonzero digits even if this causes least-significant truncation.

If P=1, positive signed 4-bit results are stored using octal 13 as the plus sign. If P=0, positive signed 4-bit results are stored with octal 14 as the plus sign. If RD is a 1, rounding takes place prior to storage.

Provided that strings 1, 2, and 3 are not overlapped, the contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Same as for SB2D

NOTES:

1. Truncation fault same as for AD3D.

2. Illegal Procedure fault same as for MVN.

3. The zero indicator is set when the decimal number is zero.

4. Independent of the data type being used (either packed decimal or 9-bit numeric; floating-point or scaled) significant digits in the result may be lost if:

   a. The difference between the scaling factors (exponents) of the source operands is large enough to cause the expected length of the intermediate result to exceed 63 digits after decimal-point alignment of source operands, followed by subtraction

   b. The result field as defined by the result descriptor is not large enough to contain the calculated result after it has been aligned

5. If an illegal digit or sign is detected, part or all of the receiving field may be changed before the IPR fault occurs.

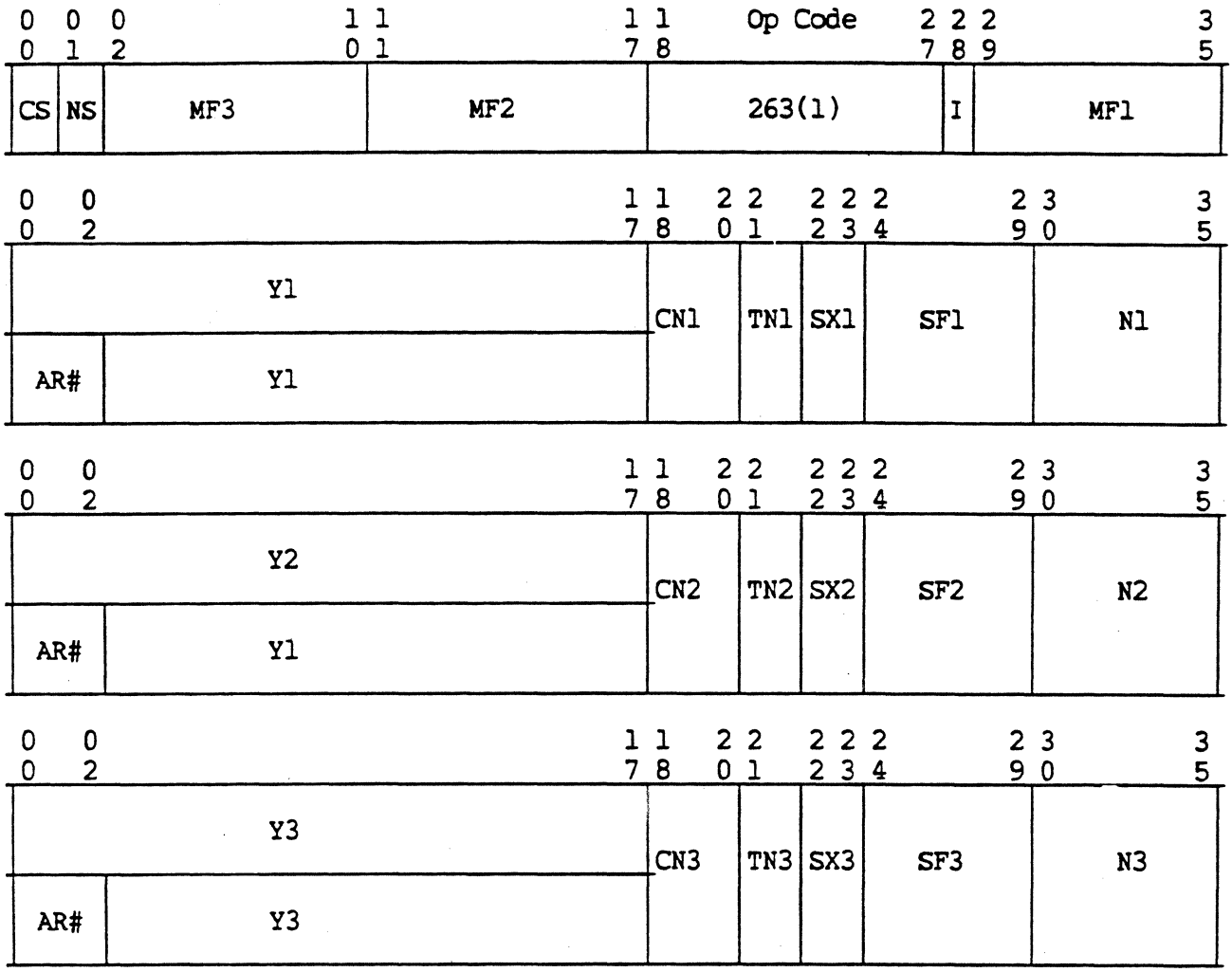
EXAMPLES: Applies to NS mode only

```
1       8       16              32
        SB3D    ,,,1            with rounding option
        NDSC4   FLD1,0,4,2      subtrahend operand descriptor
        NDSC4   FLD2,0,4,1      minuend operand descriptor
        NDSC9   FLD3,3,5        operand descriptor for result field
        USE     CONST.          memory contents
FLD1    EDEC    4P123-          123-
FLD2    EDEC    4P-123          -123
FLD3    BSS     2               X X X + 0 0 0 +127      (Result)
        USE                     zero indicator ON
```

```
1       8       16              32
        SB3D                    with truncation enable option
        NDSC9   FLD1,0,8        subtrahend operand descriptor
        NDSC9   FLD2,0,8        minuend operand descriptor
        NDSC4   FLD3,0,8,1,-2   result operand descriptor
        USE     CONST.          memory contents
FLD1    EDEC    8A-123456E-3    - 1 2 3 4 5 6 -3
FLD2    EDEC    8A-987654E-3    - 9 8 7 6 5 4 -3
FLD3    BSS     1               -0086419   (Result)
        USE                     indicators on? -  negative and truncation
```

SB3DX	Subtract Using Three Decimal Operands Extended	263 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
SB3DX  (MF1),(MF2),(MF3),RD,CS,T,NS
NDSCn  LOCSYM,CN,N,SX,SF,AM
NDSCn  LOCSYM,CN,N,SX,SF,AM
NDSCn  LOCSYM,CN,N,SX,SF,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 2) - C(string 1) --> C(string 3)

EXPLANATION: The decimal number of data type TN1, sign and decimal type SX1, and starting location YC1, is subtracted from the decimal number of data type TN2, sign and decimal type SX2, and starting location YC2. The difference is stored starting in location YC3 as a decimal number of data type TN3 and a sign and decimal type SX3.

If SX3 indicates a fixed-point format, the difference is stored using scale factor SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most-significant-digit overflow or least-significant-digit truncation to occur.

If SX3 indicates a floating-point format, the result is right-justified to preserve the most-significant-nonzero digits even if this causes least-significant truncation. The character set is defined by CS. Placement of overpunched sign in the output is controlled by NS. (Refer to definition of NS in introductory pages of this section.)

If RD = 1, rounding takes place prior to storage.

Provided strings 1, 2, and 3 are not overlapped, the contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, or MF3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Same as for AD3D

NOTES:

1. All notes for AD3D apply to SB3DX.
2. See MVNX for information about coding of overpunched signs.

SBA	Subtract from A-Register	175 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(A) - C(Y) --> C(A); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero	-	If C(A) = 0, then ON; otherwise, OFF
Negative	-	If $C(A)_0$ = 1, then ON; otherwise, OFF
Overflow	-	If range of A is exceeded, then ON
Carry	-	If a carry out of bit 0 of C(A) is generated, then ON; otherwise, OFF

SBAQ	Subtract from AQ-Register	177 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(AQ) - C(Y-pair) --> C(AQ); C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of AQ is exceeded, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

SBAR	Store Base Address Register	550 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(BAR) --> $C(Y)_{0-17}$

$C(Y)_{18-35}$ unchanges

EXPLANATION: The relationship between C(Y) and the BAR follows.

Bits	Interpretation
0-7	Base Address/1024
8	Not used
9-16	(Unrelocated Address Limit)/1024
17	Not used

The base address is a zero modulo 1024 word address that is the first valid address allocated to the slave program. The unrelocated address limit is a zero modulo 1024 word address that is the first invalid address relative, relative to the base address, beyond the memory space allocated to the slave program. (The unrelocated address limit/1024 is also the quantity of 1024-word blocks allocated to the slave program.)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are executed.

SBD
SBDX

SBD SBDX	Subtract Bit Displacement from Address Register	523 (1)

FORMAT: Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

```
1       8       16
                {SDB }
                {SBDX}   word displacement,R,AR
```

OPERATING MODES: Any

EXPLANATION: Description is the same as for ABD except that y and C(DR) are added and the sum is subtracted from the AR.

When the mnemonic is coded with an X (SBDX), bit 29 is forced to zero. If bit 29 is 0, the content of ARn is assumed as 0.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, and IC specified in DR.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES: Applies to NS mode only

```
1       8       16              32
        EAX1    48
        SBDX    2,1,6           AR6 octal contents  -  7 7 7 7 7 4 4 6
        SBD     0,1,6           AR6 octal contents  -  7 7 7 7 7 3 2 3

        EAX2    75
        SBDX    1,2,3           AR2 octal contents  -  7 7 7 7 7 4 6 6
        SBD     0,2,3           AR2 octal contents  -  7 7 7 7 7 2 6 3
```

SBLA	Subtract Logical from A-Register	135 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(A) - C(Y) --> C(A); C(Y) unchanged

EXPLANATION: This instruction is identical to SBA except that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(A) = 0, then ON; otherwise, OFF

Negative - If $C(A)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(A) is generated, then ON; otherwise, OFF. When the Carry indicator is OFF, the range of A has been exceeded.

SBLAQ	Subtract Logical from AQ-Register	137 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(AQ) - C(Y-pair) --> C(AQ); C(Y-pair) unchanged

EXPLANATION: This instruction is identical to SBAQ except that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF. When the carry indicator is OFF, the range of AQ has been exceeded.

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

SBLQ	Subtract Logical from Q-Register	136 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Q) - C(Y) --> C(Q); C(Y) unchanged

EXPLANATION: This instruction is identical to SBQ except that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

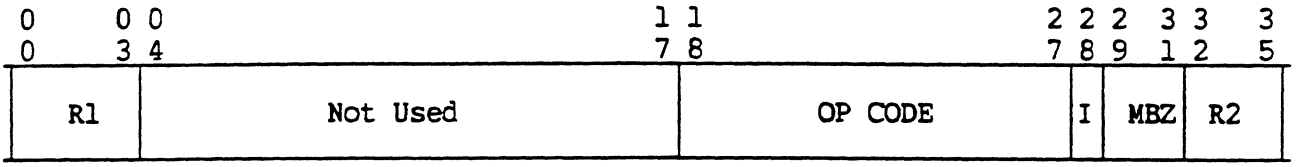
Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If $C(Q)_0$ = 1, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(Q) is generated, then ON; otherwise, OFF. When the carry indicator is OFF, the range of Q has been exceeded.

SBLR	Subtract Logical Register from Register	437 (1)

FORMAT:



CODING FORMAT:

```
1       8       16
        SBLR    R1,,R2
```

OPERATING MODES: Executes in ES mode only.

SUMMARY: R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) - C(R2) --> C(R1)

C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS:

Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(R1) is generated, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

SBLXn	Subtract Logical from Index Register n	12N (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,..., or 7 as determined by op code

$C(Xn) - C(Y)_{0-17} \rightarrow C(Xn)$; C(Y) unchanged

ES Mode

For n = 0,1,..., or 7 as determined by op code

C(GXn) - C(Y) --> C(GXn); C(Y) unchanged

EXPLANATION: This instruction is identical to SBXn except that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of SBLX0

INDICATORS:

Zero - If C(Xn/GXn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn/GXn)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(Xn/GXn) is generated, then ON; otherwise, OFF

NOTES:

1. If DL modification is specified in the NS mode, all data is processed as 0.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SBQ	Subtract from Q-Register	176 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Q) - C(Y) --> C(Q); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(Q) = 0, then ON; otherwise, OFF

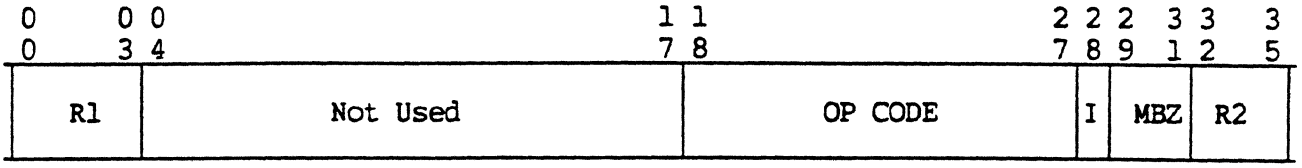
Negative - If $C(Q)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of Q is exceeded, then ON

Carry - If a carry out of bit 0 of C(Q) is generated, then ON; otherwise, OFF

SBRR	Subtract Register from Register	436 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
        SBRR   R1,,R2
```

OPERATING MODES: Executes in ES mode only.

SUMMARY: R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) - C(R2) --> C(R1)

C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification is not executed.

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: Execution in NS mode

INDICATORS:

Zero - If C(R1) = 0, then ON; otherwise, OFF

Negative - If $C(R1)_0 = 1$, then ON; otherwise, OFF

Overflow - If the range of R1 is exceeded, ON

Carry - If a carry out of bit 0 of C(R1) is generated, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

SBXn	Subtract from Index Register n	16n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0, 1, ...,7 as determined by op code

$C(Xn) - C(Y)_{0-17} \rightarrow C(Xn)$; C(Y) unchanged

ES Mode

For n = 0, 1, ...,7 as determined by op code

C(GXn) - C(Y) --> C(GXn); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of SBX0

INDICATORS:

Zero - If C(Xn/GXn) = 0, then ON; otherwise, OFF

Negative - If $C(XnGXn)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of Xn/GXn is exceeded, then ON

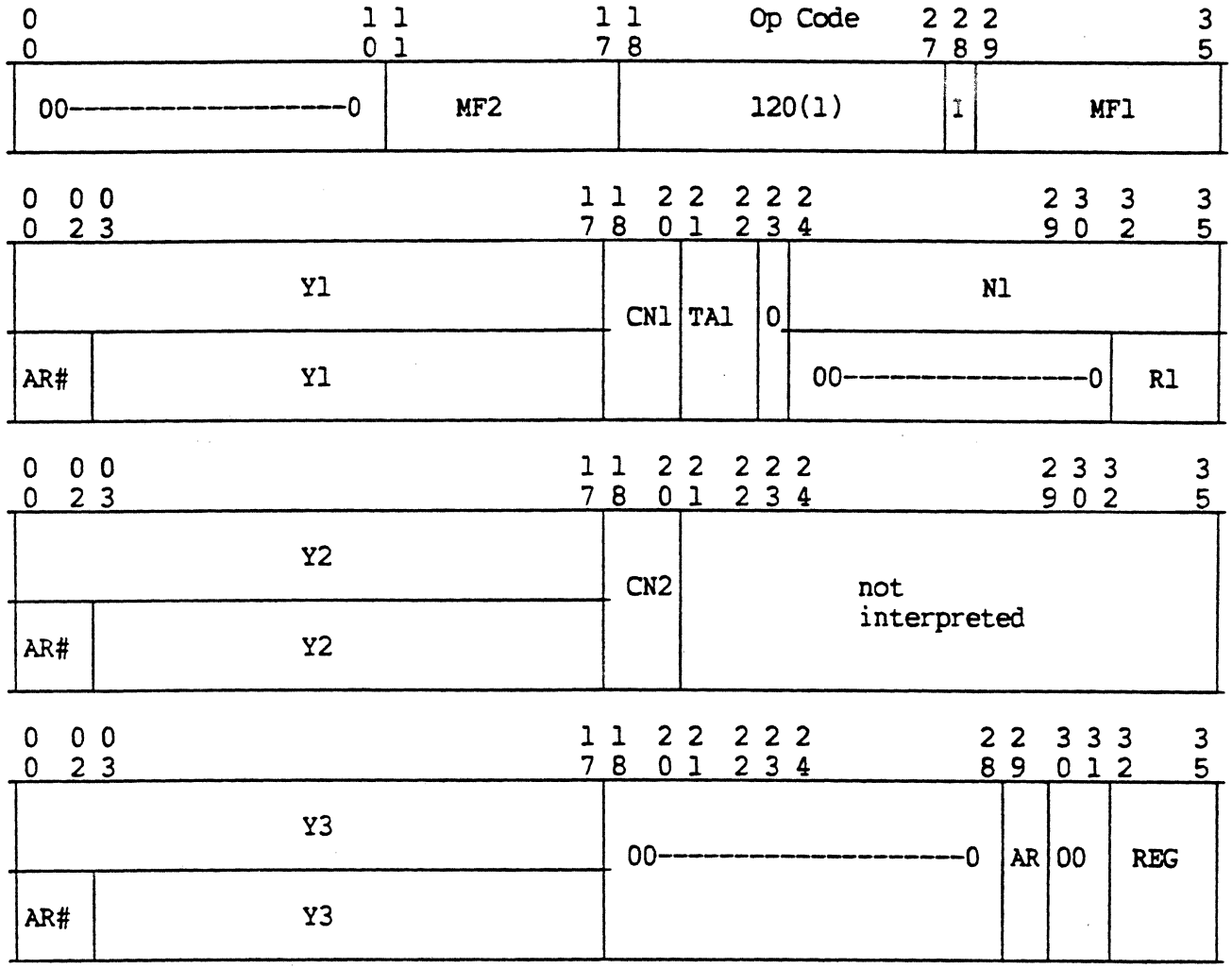
Carry - If a carry out of bit 0 cf C(XnGXn) is generated, then ON; otherwise, OFF

NOTES:

1. If DL modification is specified in the NS mode, all data is processed as 0.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SCD	Scan Characters Double	120 (1)

FORMAT:



CODING FORMAT: The SCD instruction is coded as follows:

```
1       8       16
_______________________

        SCD     (MF1),(MF2)
        ADSCn   LOCSYM,CN,N,AM
        ADSCn   LOCSYM,CN,,AM
        ARG     LOCSYM,RM,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

EXPLANATION: When N1 = 0 or 1, starting at location YC1, L1-1 concatenated pairs of type TA1 characters are compared with the two assumed type TA1 characters that are either stored in location YC2 and YC2 + 1 or contained in bits 0-7, bits 0-11, or; when the REG field of MF2 specifies DU modification, bits 0-17 of the address field of operand descriptor 2.

The compare continues until an identical match is found or until the L1-1 tally is exhausted. A count of compares is kept and for each unsuccessful match the count is incremented by 1. When a match is found or the tally is exhausted, the compare count is stored in bits 12-35 of Y3 and bits 0-11 of Y3 are zeroed.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or the Y3 REG field; DL for MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Tally - If the tally (L1-1) is exhausted without a successful match, then ON; otherwise, OFF

NOTES:

1. The RL bit in the MF2 field is not used.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

```
1       8       16              32
        SCD                     with no options
        ADSC6   FLD1,,6         scanned string operand descriptor
        ADSC6   FLD2,3          character pair operand descriptor
        ZERO    FLD3            FLD3 operand descriptor pointer
        TTF     HAVE1           match found - tally runout OFF
        USE     CONST.          characters compared
FLD1    BCI     1,123456        123456
FLD2    BCI     1,654321        32
FLD3    BSS     1               unmatched count - 5
        USE                     Result - no match found
```

```
1       8       16              32
        SCD                     with no options
        ADSC6   DATA,,24        24 characters fetched from lower DATA
        ADSC6   COMPmm2         in units of 2 chars. and compared with
                                HH.
        ARG     COUNT           when HH found in DATA, count stored as
*                               binary number before HH detection and
*                               instruction terminated.
          .       .
          .       .
          .       .
DATA    BCI     2,AABBCCDDEEFF
        BCI     2,GGHHIIJJKKLL
COUNT   BSS     1               COUNT countains decimal 14
COMP    BCI     1,HH
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16              32
        EAX5    5               load 5 into X5
        EAX7    7               load 7 into X7
        EAX4    FLD1            load FLD1 address into X4
        AWDX    0,4,4           put FLD1 address into AR4
        SCD     (1,1,,5),(,,,DU) - with address modification
        ADSC9   0,0,X7,4        FLD1 operand pointer (FLD1+1,1,7)
FLD2    VFD     A18/45          FLD2 operand
        ARG     FLD3            pointer to count FLD3
        TTN     *+2             no match found
        NULL                    match found
        USE     CONST.          characters compared
FLD1    EDEC    12A1234567      000001234567
FLD3    DEC     0               unmatched count - 3
        USE                     Result - match found on 4th pair
```

SCDR	Scan Characters Double in Reverse	121 (1)

FORMAT: Same as Scan Characters Double (SCD) format

CODING FORMAT: The SCDR instruction is coded as follows:

```
1       8       16
        SCDR    (MF1),(MF2)
        ADSCn   LOCSYM,CN,N,AM
        ADSCn   LOCSYM,CN,,AM
        ARG     LOCSYM,RM,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

EXPLANATION: Same as for SCD except that start is at location YC1 + (L1-1) and pairs are scanned in reverse to location YC1.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or the Y3 REG field; DL for MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Tally - If the tally (L1-1) is exhausted without a successful match, then ON; otherwise, OFF

NOTES:

1. The RL bit in the MF2 field is not used.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

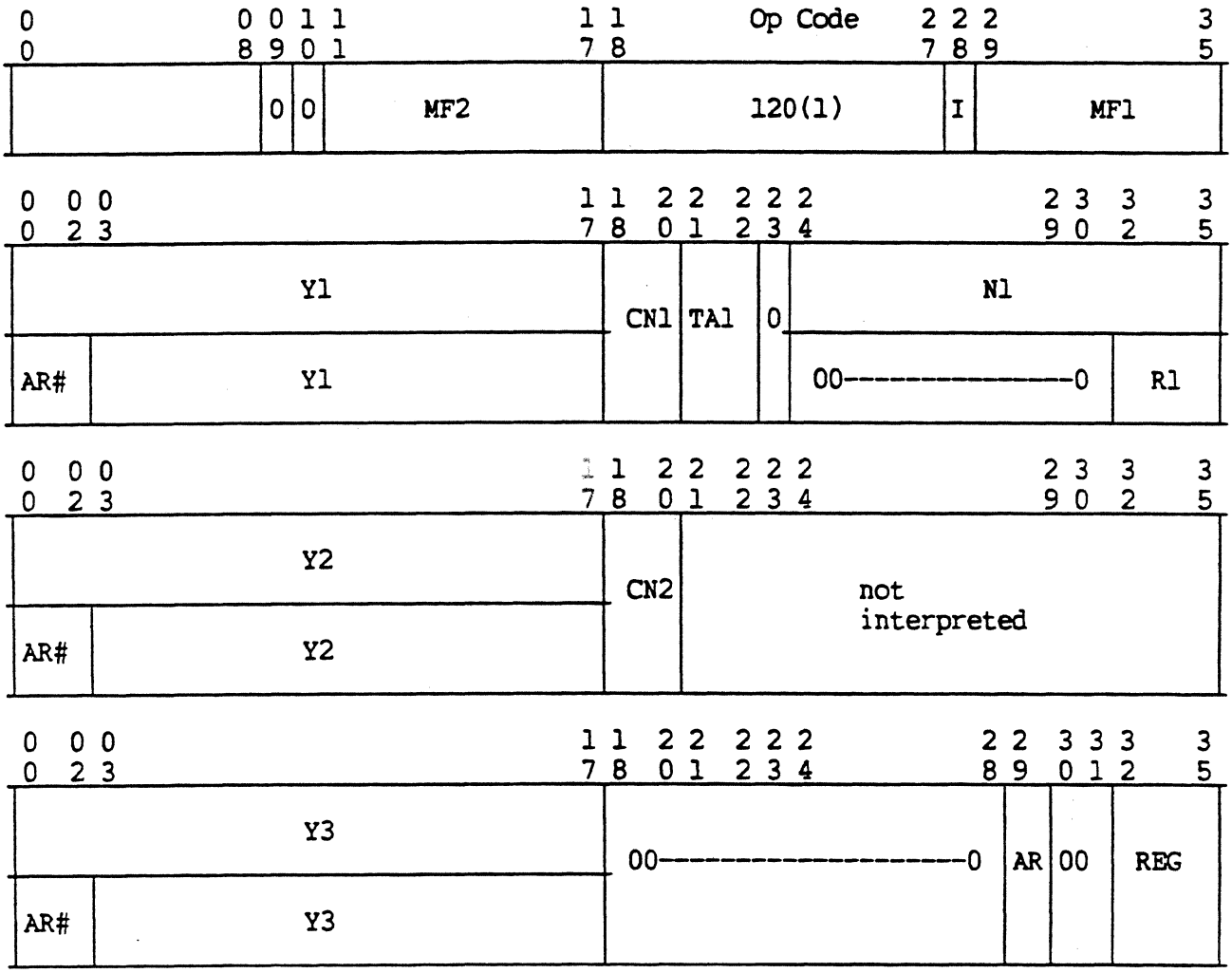
```
1       8       16              32
        SCDR    ,(,,,DU)        DU modification of FLD2 operand descriptor
        ADSC9   FLD1,0,8        scanned string operand descriptor
        VFD     U18/AB          FLD2 character pair - A B
        ARG     FLD3            pointer count word
        TTF     HAVE1           match found - tally runout OFF
        USE     CONST.          characters compared
FLD1    UASCI   2,ABCDE         A,B,C,D,E,Ø,Ø,Ø
FLD3    BSS     1               unmatched count - 6
        USE                     Result - match found on 7th pair
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16              32
K0      EQU     0
K7      EQU     7
        EAX2    1
        EAX3    FLD1            load FLD1 address into X3
        AWDX    0,3,4           put FLD1 address into AR4
        SCDR    (1,,,2),(,,,DU) - with address modification
        ADSC4   0,K0,K7,4       FLD1 operand descriptor (FLD 1,1,7)
        EDEC    2PL23           FLD2 operand descriptor pointer
        ARG     FLD3            pointer to count word
        TTN     OOPS            no match - tally runout ON
        NULL                    match found
        USE     CONST.          characters compared
FLD1    EDEC    8P123456        0123456 VS 23
FLD3    BSS     1               unmatched count - 3
        USE                     Result - match found on 4th pair
```

SCM	Scan with Mask	124 (1)

FORMAT:



CODING FORMAT: The SCM instruction is coded as follows:

```
1       8       16
_______________________

        SCM     (MF1),(MF2),MASK
        ADSCn   LOCSYM,CN,N,AM
        ADSCn   LOCSYM,CN,,AM
        ARG     LOCSYM,RM,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

EXPLANATION: Starting at location YC1, the L1 type TA1 characters are masked and compared with the assumed type TA1 character contained either in location YC2 or in bits 0-8 or 0-5 of the address field of operand descriptor 2 (when the REG field of MF2 specifies DU modification). The mask is right-justified in bit positions 0-8 of the instruction word. Each bit position of the mask that is a 1 prevents that bit position in the two characters from entering into the compare.

The masked compare operation continues until either a match is found or the tally (L1) is exhausted. For each unsuccessful match, a count is incremented by 1. When a match is found or when the L1 tally runs out, this count is stored right-justified in bits 12-35 of location Y3 and bits 0-11 of Y3 are zeroed. The contents of location YC2 and the source string remain unchanged. The RL bit of the MF2 field is not used.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or Y3 REG field; DL for MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Tally - If the tally (L1) is exhausted without a successful match, then ON; otherwise, OFF

NOTES:

1. If L1 = 0, zero is stored in Y3 (bits 12-35) and the tally indicator is affected.
2. If L1 ≠ 0 and a match is found in the first character, zero is stored in Y3 (bits 12-35) and the tally indicator is set to OFF.
3. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

```
1       8       16              32
        SCM     ,,760           mask to eliminate zone bits
        ADSC9   FLD1,0,4        character string operand descriptor
        ADSC9   FLD2,3          compare character operand descriptor
        ARG     FLD3            pointer to unmatched count word
        TTF     GOT.IT          match found
        NULL                    no match - tally runout ON
        USE     CONST.          octal representation of scanned characters
FLD1    ASCII   1,ABCD          141  142  143  144  (before masking)
                                001  002  003  004  (after masking)
                                octal representation of compare character
FLD2    ASCII   1,0004          064 (before masking)
FLD3    BSS     1               004 (after masking)
        USE                     unmatched compare count - 3
                                Result - match found on 4th character

        SCM     ,(,,,DU)        DU type REG modifier on FLD2
        ADSC4   FLD1,3,5        character string operand descriptor
        EDEC    8PL-1           FLD2's compare character -
        ARG     FLD3            pointer to unmatched count word
        TTF     GOT.IT          match found
        NULL                    no match - tally runout ON
        USE     CONST.          character scanned
FLD1    EDEC    8P-1234         0,1,2,3,4
FLD3    BSS     1               unmatched compare count - 5
        USE                     Result - no match found
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16              32
        EAX1    1               load FLD2 character modifier into X1
        EAX2    2               load FLD1 character modifier into X2
        EAX4    FLD1            load FLD1 address into X4
        AWDX    0,4,4           put FLD1 address into AR4
        SCM     (1,1,1,2),(1,,1,1),010  with all options
        ARG     INDSC1          pointer to FLD1 indirect descriptor
        ARG     INDSC2          pointer to FLD2 indirect descriptor
        ARG     FLD3            pointer to unmatched count word
        TTN     0Y              no match - tally runout ON
        USE     CONST.          character compared
FLD1    EDEC    8PL4321         2 1
FLD2    EDEC    4P0987          1
FLD3    BSS     1               unmatched compare count - 1
INDSC1  ADSC4   0,,X2,4         FLD1 operand descriptor (FLD1,2,2)
INDSC2  ADSC9   FLD2,0          FLD2 operand descriptor (FLD2,1)
        USE                     Result - match found on 2nd character
```

SCMR	Scan with Mask in Reverse	125 (1)

FORMAT: Same as Scan with Mask (SCM) format

CODING FORMAT: The SCMR instruction is coded as follows:

```
1       8       16
SCMR    (MF1),(MF2),MASK
ADSCn   LOCSYM,CN,N,AM
ADSCn   LOCSYM,CN,,AM
ARG     LOCSYM,RM,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

EXPLANATION: Same as SCM except starts at location YC1 + (L1-1) and progresses toward location YC1.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or the Y3 REG; DL for MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Tally - If the tally (L1) is exhausted without a successful match, then ON; otherwise, OFF

NOTES:

1. If L1 = 0, zero is stored in Y3 (bits 12-35) and the tally indicator is affected.
2. If L1 > 0 and a match is found in the first character, zero is stored in Y3 (bits 12-35) and the tally indicator is set to OFF.
3. The RL bit of the MF2 field is not used.
4. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

```
1       8       16              32
        SCMR    ,(,,,DU),760    DU type register modification with mask
        ADSC4   FLD1,0,6        character string operand descriptor
        EDEC    1P4             FLD2's compare character - 4
        ARG     FLD3            pointer to unmatched count word
        TTF     *+2             match found
        NULL                    no match - tally runout ON
        USE     CONST.          characters scanned
FLD1    EDEC    8PL654321-      6,5,4,3,2,1
FLD3    DEC     0               unmatched count - 3
        USE                     Result - match found on 4th character
```

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16              32
        EAX6    6               load FLD1 length into X6
        EAX2    2               load character modifier into X2
        EAX4    FLD1            load FLD1 address into X4
        AWDX    0,4,4           put FLD1 address into AR4
        SCMR    (1,1,1,2),,760  with all options
        ARG     FLD3+1          pointer to FLD1 indirect descriptor
        ADSC4   FLD2,0          pointer to compare character
        ARG     FLD3            pointer to unmatched count word
        TTN     OUCH            no match - tally runout ON
        TRA     WHEW            match found
        USE     CONST.          characters compared
FLD1    EDEC    8P0123456-      2,3,4,5,6,-
FLD3    DEC     0               unmatched compare count - 4
        ADSC4   0,,X6,4         FLD1 operand descriptor(FLD 1,2,6)
FLD2    EDEC    4PL3            FLD2 compare character 3
        USE                     Result - match found on 5th compare
```

SCPR	Store Central Processor Register	452 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(CPU Reg.) --> C(Y-pair), or
C(Y-Block 4), or
C(A), or
C(Y, Y+1,...Y+7)

EXPLANATION: This instruction selects CPU registers based upon the instruction's tag field, and stores them in memory or loads them into the A register

The tags and register/operand stored are as follows:

Octal Tag	Register/Operand	C(Y-Pair) Bits
01	Fault Register	0-35
	0....0	36-71
03	Extended Fault Register	0-7
	0....0	8-71
06	CPU Mode Register	0-35
	0....0	36-53
	Cache Mode Register	54-61
	0....0	62-69
	Lockup Fault Register	70-71
10	Reserve Memory Base	0-35
	0....0	36-71
11	Port Configuration Register	0-17
	0....0	18-71
12	Address Trap Register	0-30
	0....0	31-71

Octal Tag	Register/Operand	C(Y-Pair) Bits
	0....0	0-32
13	CPU Number Register	33-35
	0....0	36-71
14	Virtual Address Trap Register	0-35
	0....0	36-71

		C(Y-Block) 4 Bits
20	History Register	0-35
	0....0	36-58
	History Register	59-71
	0....0	72-79
	History Register	80-107
	0....0	108-143

		C(Y,Y+1,...Y+7) Bits
07	Connect Table,	0-143
	Secondary Connect Table	144-287

The following tags load the contents of the cache directory, PTWAM directory, and PTWAM registers into the A-register. The entry location is specified by the Y address field in the instruction.

Tag	Entry Select Column	Entry Level	Entry	C(A) Bits
15	Y_{3-14}	Y_2	Cache Directory	0-35
16	Y_{11-16}	Y_{17}	PTWAM Register	0-35
17	Y_{11-16}	Y_{17}	PTWAM Directory	0-35

ILLEGAL ADDRESS MODIFICATIONS: Tag field defines the operation.

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: None affected

SCPR

NOTES:

1. A Command fault occurs if execution is attempted in Slave or Master mode.

2. For SCPR tags 15, 16, and 17, if bit 29 is ON, C(AR) is added to the Y field and the sum forms the entry select value. The full virtual address development is not used.

3. The address trap register values are read from scratch pad locations 66, 67 rather than from the register itself.

4. An IPR fault occurs if illegal tag fields or illegal repeats are executed.

SDRn	Save Descriptor Register n	11n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(DRn) --> Argument Stack (AS)

Argument stack bound --> HWMR (Refer to Explanation item 4.)

SEGIDn is set to indicate the stored segment descriptor.

EXPLANATION: This instruction stores the descriptor from DRn in the next available location of the argument stack, and adjusts the argument stack bound and high water mark register (HWMR). The y field of this instruction is not interpreted by the hardware. No address bound checks are made. The argument segment is the operand segment.

The instructions are executed as follows.

1. The following checks are performed.

   a. The ASR (Argument Stack Register) flag bit 28 is checked. If it is zero, the argument segment is not present, a Missing Segment fault occurs, and the instruction is terminated.

   b. If the ASR bound + 8 $\geq$ 8192 bytes, a BND fault is generated.

2. If the conditions described under (1) are satisfied, execution continues. It generates the effective byte address indicating the next available double-word location on the AS. The ASR flag bit 27 is then checked.

   a. If the ASR flag bit 27 = 0, the argument segment is empty. The ASR base indicates the first double-word location.

   b. If the ASR flag bit 27 = 1, ASR bound + base + 1 is executed to generate a virtual address.

   NOTE: The descriptor is stored relative to the argmuent stack bound. The HWMR does not influence this storage location. (Refer to the description of the CLIMB instruction for more information on the HWMR.)

3. After the DRn content has been stored in AS, the following operations are executed and the instruction is completed.

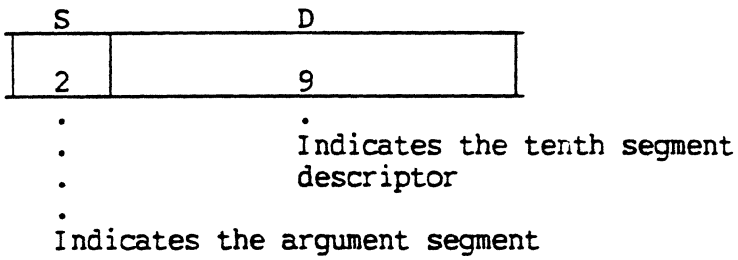
   a. The ASR flag bit 27 is checked.

      If ASR bit 27 = 1, 8 is added to the ASR bound field. It indicates that the new segment has been stored and the segment size has increased.

      If ASR bit 27 = 0, the argument segment indicates that it was empty when the instruction was begun. The bound field is then set to seven bytes to indicate that a segment descriptor has been stored. The ASR flag bit 27 is set to 1 to indicate that this segment is no longer empty.

   b. SEGIDn is set to indicate the location in which the segment descriptor is stored.

      For example, if the ASR bound field is 117 (octal) bytes (= 80 bytes = 20 words = 10 double-words) after 8 is added, SEGIDn is set as follows.



4. The HWMR is set to indicate the maximum ASR bound following any sequence of SDRn and PAS instructions.

   If the new ASR bound > C(HWMR), then the new ASR bound --> C(HWMR).

ILLEGAL ADDRESS MODIFICATIONS DU, DL, RI, IR, IT

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. A Missing Working Space, Missing Segment, or Missing Page fault may occur.

2. If a save is attempted to a nonhousekeeping page, a Security Fault, Class 1 occurs.

3. An BND fault occurs if the ASR bound + 1 byte $\geq$ 8192 bytes (before the ASR is updated).

4. A Security Fault, Class 2 fault occurs if a working space violation is attempted, or if the specified page does not have write permission. The descriptor itself is not required to have either write or store permission.

5. An Illegal Procedure fault occurs if illegal address modification or illegal repeats are used.

SIW	Set Interrupt Word Pair	451 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(AQ) --> C(Interrupt Queue)$_{0-71}$

EXPLANATION: A double-word write occurs to the designated control SCU. The SCU stores the double word in the level interrupt queue and informs all of the receiving ports. The SCU looks at bits 27-30 of the data to determine the interrupt queue level. The eight queues are circular, first-in/first-out, with queue lengths of 256 word pairs per port. If the queue level number exceeds 256, a bit is set in the SCU fault register.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: None affected

NOTES:

1. Prior to executing this instruction, the SCU must be "selected" by using the LCPR instruction to set or reset bit 22 in the CPU mode register.

2. An IPR fault occurs if illegal address modification or an illegal repeat is used.

3. An IPR fault occurs if execution is attempted in Slave or Master mode.

SMID	Set Memory ID Register	272 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(AQ) --> C(Memory ID Register)

EXPLANATION: This instruction sets the memory ID registers. The physical memory unit that is selected by the address is dependent upon the SCU's physical ID or logical ID based on the setting of the SCU configuration register.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: None Affected

NOTES:

1. SCU selection is based upon the control SCU bit in the CPU mode register.
2. An IPR fault occurs if illegal address modification or an illegal repeat is executed.
3. An IPR fault occurs if execution is attempted in Slave or Master mode.

SMR	Set Memory Register	271 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(AQ) inverted --> C(Memory Status Register)

EXPLANATION: This instruction provides a means of setting the memory status registers. SCU selection is based upon the control SCU bit in the CPU mode register.

Address development is followed and transferred to the SCU to select the memory unit. The physical memory unit that is selected by the address is dependent upon the SCU's physical ID or logical ID based on the setting of the SCU configuration register.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if illegal address modification or an illegal repeat is used.
2. An IPR fault occurs if execution is attempted in Slave or Master mode.
3. The contents of the AQ register are inverted (one's complement).

SPCF	Set Pointer Compare Flags Off	251 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: 1 --> SD Compare Flagn

where n = 0, 1, ... 7

EXPLANATION: This instruction provides a means to turn segment descriptor (SD) compare flags OFF, and to inhibit the compare.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None

NOTE: Disabled by GCOS

SPDBR	Store Page Table Directory Base Register	151 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(PDBR) --> $C(Y)_{0-18}$ (Mod 512)

00...0 --> $C(Y)_{19-35}$

C(PDBR) unchanged

EXPLANATION: The PDBR content is stored in bit 0-18 of location Y. Zero is stored in $C(Y)_{19-35}$. The PDBR content remains unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.
2. A Command fault occurs if execution of this instruction is attempted in Slave or Master Mode.

SPL	Store Pointers and Lengths	447 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        SPL     LOCSYM,R,AR
```

OPERATING MODES: Any

SUMMARY: C(Pointer and Length storage) --> C(Y), C(Y+1), ... C(Y+5)
C(LOR) --> C(Y+6), C(Y+7)

EXPLANATION: The pointers and lengths storage are used by hardware to store control information when an interruptible multiword instruction is interrupted during execution. These registers enable hardware to resume processing an interrupted instruction after a return from servicing the interrupt.

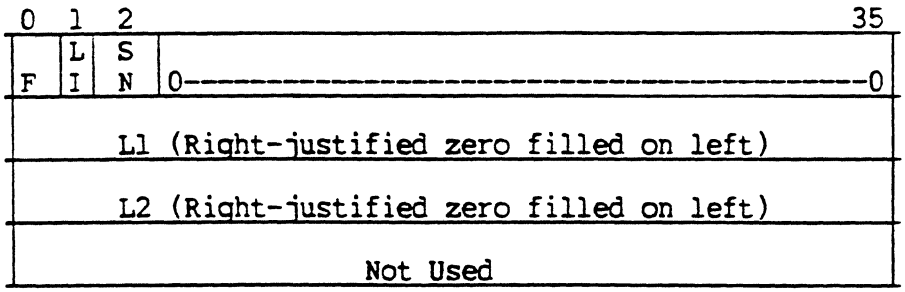
Y must be a multiple of 8. However, a fault does not occur when the lower 3 bits of Y are not 000. For purposes of execution, the hardware forces these bits to 000 (modulo 8).

The format of the eight words is the same as words 48 through 55 of the Safe Store Stack format (see Figures 8-7 and 8-8 under CLIMB). The contents of the first four words depend upon whether the multiword instruction is alphanumeric or bit string.

For an SPL execution, the eight words are stored into scratch pad memory and the first flag is set or reset.

The format of the first four words follows.

Alphnumeric Instructions

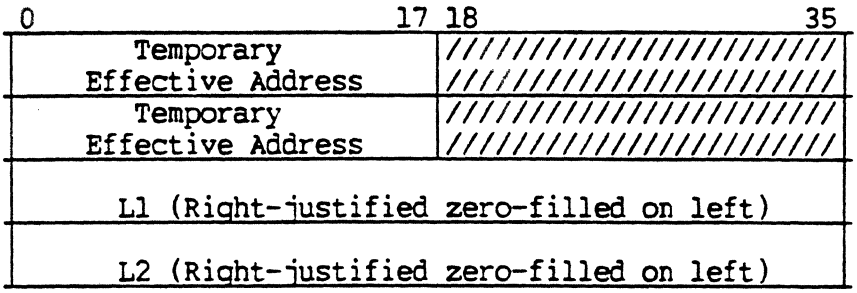


where

F	First Flag	If = 1, indicates the start of a multiword instruction execution for which the data from the instruction operands is used. If = 0, if bit 30 in the indicator register is = 1, and, if the next instruction is an EIS instruction, the P&L data stored in scratch pad memory is used. (Refer to Indicator Register, Section 4.)
L	Length Indicator	If = 0, only the length in L1 is valid If = 1, only the length in L2 is valid The firmware uses this bit to determine whether L1 or L2 contains the valid length. The length of L1 and L2 varies depending upon whether NS or ES mode are being used. For NS mode alphanumeric, the length is 21 bits for 4- and 6-bit characters and 20 bits for 9-bit characters. For ES mode the maximum length is 36 bits.

SN	Sign Negative	This indicator is used only if the interrupted instruction is an MLR in which a 6- or 4-bit move is being done. (Refer to Explanation under description of MRL for use of overpunch sign on 6-4 moves.)

Bit String Instructions



The first effective address relates to L1; the second effective address relates to L2.

The length of L1 and L2 varies, depending upon whether NS or ES mode are being used. For NS mode, the length is 24 bits for bit strings. For ES mode the maximum length is 36 bits.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, RI, IR, IT

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTIONS: XEC, XED

INDICATORS: Multiword Instruction Interrupt indicator (bit 30), reset to OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modifications, illegal repeats, or illegal executions are used.

2. The content of the pointer and length storage is changed if RPT, RPD, RPL, XEC, or XED or indirect modification (IT) are executed.

3. The SPL instruction is normally only used by routines that process interrupts.

4. After an interrupt, the SPL must be executed before any multiword instruction to avoid destruction of the pointer and length information.

SREG	Store Registers	753 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

The registers are stored as follows:

C(X0) --> $C(Y)_{0-17}$
C(X1) --> $C(Y)_{18-35}$
C(X2) --> $C(Y+1)_{0-17}$
C(X3) --> $C(Y+1)_{18-35}$
C(X4) --> $C(Y+2)_{0-17}$
C(X5) --> $C(Y+2)_{18-35}$
C(X6) --> $C(Y+3)_{0-17}$
C(X7) --> $C(Y+3)_{18-35}$
C(A) --> $C(Y+4)_{0-35}$
C(Q) --> $C(Y+5)_{0-35}$
C(E) --> $C(Y+6)_{0-7}$; 0...0 --> $C(Y+6)_{8-35}$
C(TR) --> $C(Y+7)_{0-26}$; 0...0 --> $C(Y+7)_{27-35}$

ES Mode

The registers are stored as follows:

C(GX0) --> C(Y)
C(GX1) --> C(Y+1)
C(GX2) --> C(Y+2)
C(GX3) --> C(Y+3)
C(GX4) --> C(Y+4)
C(GX5) --> C(Y+5)
C(GX6) --> C(Y+6)
C(GX7) --> C(Y+7)
C(A) --> C(Y+8)
C(Q) --> C(Y+9)
C(E) --> $C(Y+10)_{0-7}$; 0...0 --> $C(Y+10)_{8-35}$
C(TR) --> C(Y+11); 0...0 --> $C(Y+11)_{27-35}$

In both NS and ES modes the register content remains unchanged.

SREG

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Location Y must be forced to a multiple of 8 by entering an 8 in column 7 of the statement that defines Y, or by means of the EIGHT pseudo-operation.

2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SSA	Subtract Stored from A-Register	155 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(A) - C(Y) —> C(Y); C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of C(Y) is exceeded, then ON

Carry - If a carry out of bit 0 of C(Y) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SSCR	Set System Controller Register	057 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(AQ) --> C(SCU Register)

EXPLANATION: This instruction provides program access to all system controller registers. SCU selection is based upon the control SCU bit in the CPU mode register. Address development is followed, and is transferred to the SCU to select the general register.

In VMS Privileged Master mode, if both SCU ports are enabled and the least-significant bit of the effective address (word address) is 1, the control SCU bit is temporarily changed to permit selection of the non-control SCU. (Reference Section 4 for CPU configuration register and ASR control.) The control SCU bit is then reset to its original value.

Real Memory:

0...21	Bits 22-24	25-27	Function
X...X	0	X	Not used
X...X	1	X	Configuration
X...X	2	X	Not used
X...X	3	X	Not used
X...X	4	X	Calendar Clock
X...X	5	X	Not used
X...X	6	X	Not used
X...X	7	X	Not used

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: None affected

SSCR

NOTES:

1. A Command fault occurs if address bits 22-24 are 0, 2, 3, 5, 6, or 7 (octal).

2. A Command fault occurs if execution is attempted in Slave or Master mode.

3. The SCU registers are defined in Section 4.

4. Bits 25-27 of the configuration register are the SCU port number. These bits must be zero in an SSCR instruction, in order that a subsequent RSCR instruction returns the port number; otherwise bits 25-27 are OR'ed with the port number returned.

5. An IPR fault occurs if illegal address modification or illegal repeats are executed.

SSQ	Subtract Stored from Q-Register	156 (0)

FORMAT: Single-word instruction format (see Figure 8-1)
OPERATING MODES: Any

SUMMARY: C(Q) - C(Y) --> C(Y); C(Q) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS:

Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of C(Y) is exceeded, then ON

Carry - If a carry out of bit 0 of C(Y) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SSXn	Subtract Stored from Index Register n	14n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,...,7 as determined by op code

$C(Xn) - C(Y)_{0-17} \rightarrow C(Y)_{0-17}$; C(Xn) unchanged

ES Mode

For n = 0,1,...,7 as determined by op code

C(GXn) - C(Y) --> C(Y); C(GXn) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, or RPL of SSX0

INDICATORS: NS Mode

Zero - If $C(Y)_{0-17} = 0$, then ON; otherwise, OFF

Negative - If $C(Y)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of C(Y) is exceeded, then ON

Carry - If a carry out of bit 0 of C(Y) is generated, then ON; otherwise, OFF

ES Mode

Zero - If C(Y) = 0, then ON; otherwise, OFF

Negative - If $C(Y)_0 = 1$, then ON; otherwise, OFF

Overflow - If range of C(Y) is exceeded, then ON

Carry - If a carry out of bit 0 of C(Y) is generated, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STA	Store A-Register	755 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(A) --> C(Y); C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL

ILLEGAL REPEATS: RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STAC	Store A Conditional	354 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: If C(Y) = 0, C(A) --> C(Y)

EXPLANATION: This instruction issues a read-lock, write-unlock sequence to memory. Cache is bypassed; if a cache hit occurs and the conditional test is satisfied, the cache block is updated.

If write does not occur, the next command to memory from the same processor port performs unlock.

Execution of STAC is delayed until all outstanding stores to memory from the processor have been completed.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If initial C(Y) = 0, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STACQ	Store A Conditional on Q	654 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: If C(Y) = C(Q), C(A) --> C(Y)

If C(Y) ≠ C(Q), C(Y) is unchanged

EXPLANATION: This instruction issues a read-lock, write-unlock sequence. Cache is bypassed; if a cache hit occurs and the conditional test is satisfied, the cache block is updated.

If write does not occur, the next command to memory from the same processor port performs unlock.

Execution of STACQ is delayed until all outstanding stores to memory from the processor have been completed.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If initial C(Y) = C(Q), then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STAQ	Store AQ-Register	757 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(AQ) --> C(Y-pair); C(AQ) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STAS	Store Argument Stack Register	750 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(ASR) --> C(Y-pair); C(ASR) unchanged

EXPLANATION: The execution of this instruction causes the current contents of the argument stack register (ASR) to be stored in even and odd memory locations Y and Y+1. The contents of the ASR remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

```
1       8       16
        STAS    SVASR
        SDR     P0
        STP     P0,SVP0
        SDR     P1
        STP     P1,SVP1
        .         .
        .         .
        .         .
        LDP     P0,SVP0
        LDP     P1,SVP1
        PAS     SVASR
```

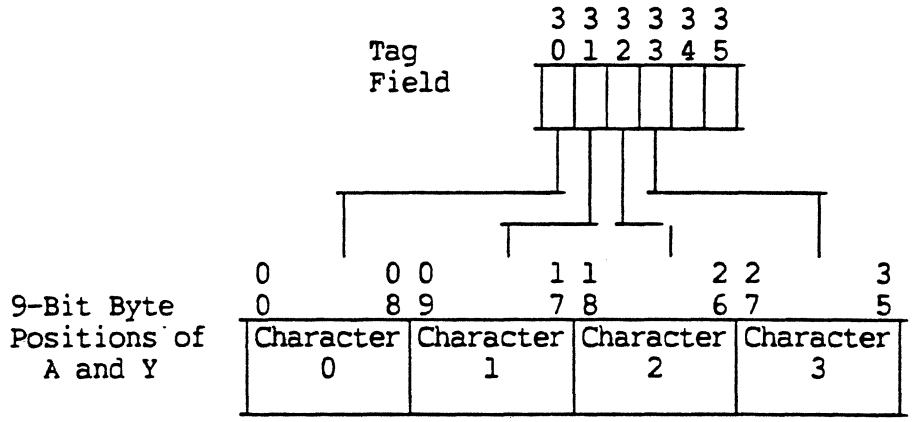
STBA	Store 9-bit Bytes of A-Register	551 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: 9-bit bytes of C(A) --> corresponding characters of C(Y); the byte positions affected are specified in the tag field; C(A) is unchanged.

EXPLANATION: Binary ones in the tag field specify the byte positions of A and Y affected as indicated in the diagram below. The tag field is entered as one 2-digit octal number. Bit positions 34 and 35 are ignored.



ILLEGAL ADDRESS MODIFICATIONS: The tag field cannot be used for address modification. AR modification is permitted.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if an illegal repeat is used.

EXAMPLE: The instruction

```
STBA    LOC,04
```

moves byte 3 of C(A) to the corresponding byte position of C(LOC) (04 octal = 000100 binary). All other byte positions of C(LOC) are unaffected.

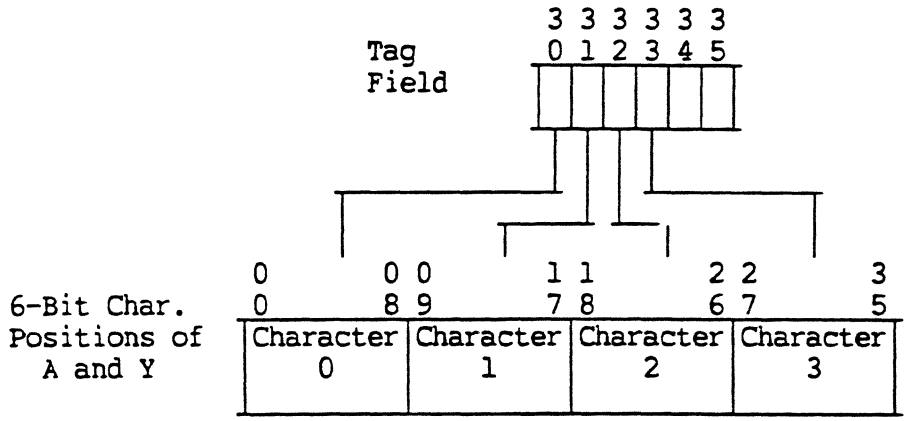
STBQ	Store 9-bit Bytes of Q-Register	552 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: 9-bit bytes of C(Q) --> corresponding bytes of C(Y); the byte positions affected are specified in the tag field; C(Q) is unchanged

EXPLANATION: Binary ones in the tag field specify the byte positions of Q and Y affected as indicated in the diagram below. The tag field is entered as one 2-digit octal number. Bit positions 34 and 35 are ignored.



ILLEGAL ADDRESS MODIFICATIONS: The TAG field cannot be used for address modification. AR modification is permitted.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if an illegal repeat is used.

EXAMPLE: The instruction STBQ LOC,04 moves byte 3 of C(Q) to the corresponding byte position of C(LOC) (04 octal = 000100 binary). All other byte positions of C(LOC) are unaffected.

STC1	Store Instruction Counter Plus 1	554 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(IC) + 1 --> C(Y); C(IR) --> $C(Y)_{18-32}$; 000 --> $C(Y)_{33-35}$; C(IC), C(IR) unchanged

EXPLANATION: The relation between bit positions of C(Y) and the indicators is as follows:

Bit Position	Indicator
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent overflow
23	Exponent underflow
24	Overflow mask
25	Tally runout
26	Parity error
27	Parity mask
28	Master mode
29	Truncation
30	Multiword instruction interrupt
31	Exponent underflow mask
32	Hexadecimal exponent mode
33-35	000

The ON state corresponds to a 1 bit; the OFF state corresponds to a 0 bit. Bit 25 of C(Y) will contain the state of the Tally Runout indicator prior to address modification of the STC1 instruction (for tally operations).

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STC2	Store Instruction Counter Plus 2	750 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(IC) + 2 --> $C(Y)_{0-17}$; $C(Y)_{18-35}$, C(IC) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

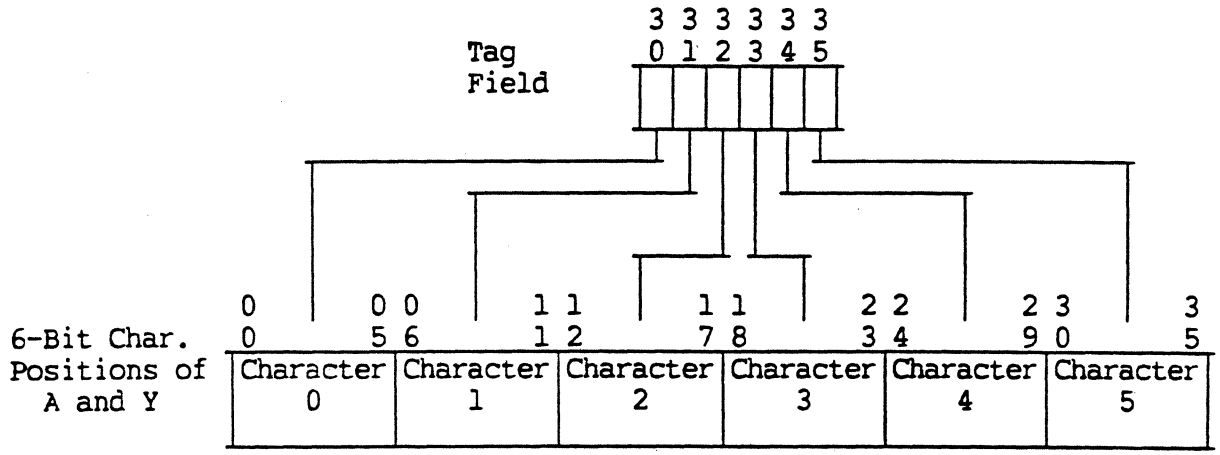
STCA	Store 6-bit Characters of A-Register	751 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: 6-bit characters of C(A) --> corresponding characters of C(Y); the character positions affected are specified in the tag field; C(A) is unchanged

EXPLANATION: Binary (1) bits in the tag field specify the affected A and Y character locations as follows. The TAG field is entered as one 2-digit octal number. (See Example below.)



The CPU reads one word from memory, embeds a character specified in the CPU into the word, and writes this word back in memory. Therefore, while the CPU reads a word and writes it, the word's content can be lost if another CPU writes the same word. To prevent multiprocessor contention, gating is necessary.

ILLEGAL ADDRESS MODIFICATIONS: No modification except AR allowed.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The tag field cannot be used for address modification. AR modification is permitted.

2. An Illegal Procedure fault occurs if illegal repeats are used.

EXAMPLE: The instruction STCA LOC,07 moves characters 3, 4, and 5 of C(A) to corresponding character positions of C(LOC) (07 octal = 000111 binary). Character positions 0, 1, and 2 of C(LOC) are unaffected.

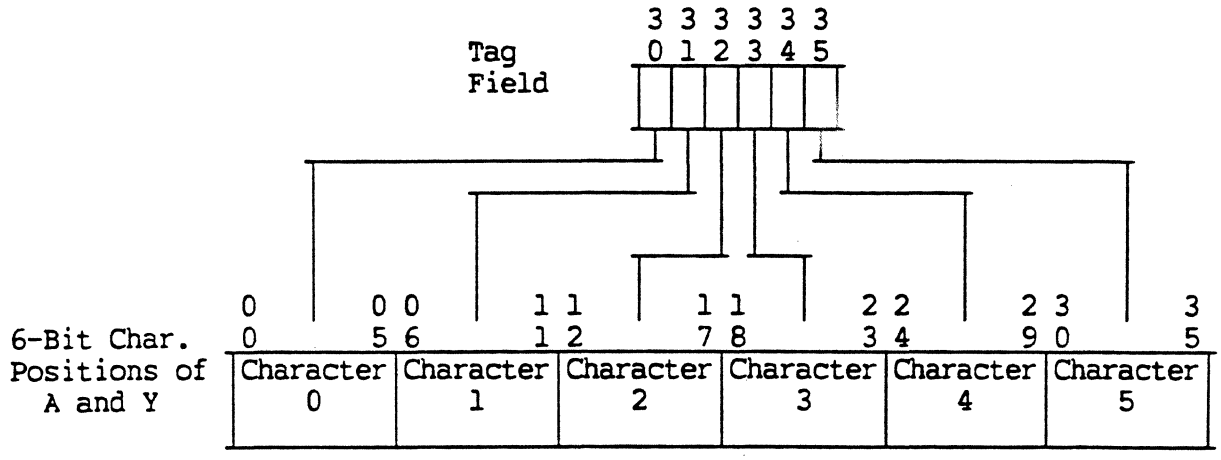
STCQ	Store 6-bit Characters of Q-Register	752 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: 6-bit characters of C(Q) --> corresponding characters of C(Y); the character positions affected are specified in the tag field.

EXPLANATION: Binary (1) bits in the tag field specify the affected Q and Y character locations as follows. The tag field is entered as one 2-digit octal number. (See Example below.)



The CPU reads one word from memory, embeds a character specified in the CPU into the word, and writes this word back in memory. Therefore, while the CPU reads a word and writes it, it is possible that the word's content can be lost if another CPU writes the same word. To prevent multiprocessor contention, gating is necessary.

ILLEGAL ADDRESS MODIFICATIONS: No modification except AR allowed.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The tag field cannot be used for address modification. AR modification is permitted.

2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE: The instruction STCQ LOC,07 moves characters 3, 4, and 5 of C(Q) to corresponding character positions of C(LOC) (07 octal = 000111 binary). Character positions 0, 1, and 2 of C(LOC) are unaffected.

STDn	Store Descriptor Register n	05n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(DRn) --> C(Y),C(Y+1); C(DRn) unchanged

EXPLANATION: This instruction stores the DRn content in an even/odd location of the segment descriptor segment or the operand segment.

If instruction bit 29 = 0 then C(DRn) --> C(Y-pair) in the instruction segment.

If instruction bit 29 = 1 and DRm descriptor type T = 1,3 (m is selected by instruction bits 0,1,2) then C(DRn) --> C(Y-pair) of descriptor segment.

NOTE: DRn store permission is required.

If instruction bit 29 = 1 and DRm descriptor type T = 0, 2, 4, 6, 12, 14 then C(DRn) --> C(Y-pair) in the operand segment.

NOTE: DRn store permission is not required.

To summarize the differences in processing performed due to the differing types of segment descriptors:

- If the DRn segment descriptor is stored in a segment descriptor segment (T = 1 or 3), the page must be a housekeeping page (PTW bit 32 must = 1). When all other conditions (e.g., write permission) are satisfied, the segment descriptor is stored, irrespective of the CPU mode.
- If an attempt is made to store in the operand segment, the write operation for the housekeeping page is dependent upon the CPU mode as the store flag is not examined by hardware.

ILLEGAL ADDRESS MODIFICATIONS: If the DRm type T = 1 or 3, only R type modification is permitted. An IPR fault occurs if DU, DL, RI, IR, or IT is specified.

If the DRm type T = 0, 2, 4, 6, 12, or 14, an IPR fault occurs when DU, DL, SC, SCR, or CI is specified.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An Illegal Procedure fault occurs when illegal address modification or an illegal repeat is used.
2. If DRn does not have store permission (bit 18 for T = 8, 9, 11; bit 22 for all other types), an SCL2 fault occurs.
3. If DRm page is not housekeeping, an SCL1 fault occurs.
4. If DRm segment or page does not have write permission, an SCL2 fault occurs.
5. If processor is in Master or Slave mode and DRm page is housekeeping, an SCL1 fault occurs.
6. If DRm segment or page does not have write permission, an SCL2 fault occurs.
7. If instruction bit 29 = 1 and DRm descriptor type T = 5 or 7-11, 13, 15, an IPR fault occurs.

STDSA	Store Data Stack Address Register	150 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(DSAR) --> $C(Y)_{0-16}$

00...0 --> $C(Y)_{17-35}$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if illegal address modifications or illegal repeats are used.
2. A Command fault occurs if this instruction is executed in Slave or Master mode.

EXAMPLE:

```
1     78       16
        STDSD   SVREG
        STDSA   SVREG+2
        LDX0    SVREG+2
        ADLX0   NWPS,DU
        CMPX0   SVREG
        TPNZ    NOGOOD
        LDD     P.DS,DSVEC
        .
        .
        .
SVREG 8BSS      8
DSVEC  FVEC     NWDS,(ALL)
```

STDSD	Store Data Stack Descriptor Register	551 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: C(DSDR) --> C(Y-pair); C(DSDR) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if illegal address modifications or illegal repeats are used.
2. A Command fault occurs if this instruction executed in Slave or Master mode.

STE	Store Exponent Register	456 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(E) --> $C(Y)_{0-7}$; 00....0 --> $C(Y)_{8-17}$;

$C(Y)_{18-35}$, C(E) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification or illegal repeats are used.

STI	Store Indicator Register	754 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(IR) --> $C(Y)_{18-32}$

00...0--> $C(Y)_{33-35}$;

$C(Y)_{0-17}$, C(IR) unchanged

EXPLANATION: The content of the indicator register is stored in $C(Y)_{18-32}$ after address modification. The value stored in $C(Y)_{25}$ is the Tally Runout status before address modification. The relation between bit positions of C(Y) and indicators is as follows:

Bit Location	Indicator
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent overflow
23	Exponent underflow
24	Overflow mask
25	Tally runout
26	Parity error
27	Parity mask
28	Master mode
29	Truncation
30	Multiword instruction interrupt
31	Reserved for exponent underflow mask
32	Hexadecimal exponent mode
33-35	000

The ON state corresponds to a 1 bit; the OFF state to a 0 bit.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STO	Store Option Register	152 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(DSCF) --> bit 18 of C(Y)

C(SSBF) --> bit 19 of C(Y)

00...0 --> remaining 34 bits of C(Y)

EXPLANATION: This instruction stores the two flag bits of the option register in memory.

DSCF Data stack clear flag
0 = do not clear
1 = clear

SSBF Safe store bypass flag
0 = bypass safe store during ICLIMB
1 = perform safe store during ICLIMB

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLES:

```
1       8       16                  32
ORNCHE  BOOL    4000                *CRCF bit of option register
MPOR    EQU     *
        LDO     .SORSV,,P.SSA
        STO     .CRORR,PN,P.CR      *set with CRCF ON
        STO     .CRORS,PN,P.CR
        LDA     ORNCHE,DL
        ERSA    .CRORS,PN,P.CR      *reset CRCF to OFF
        TRA     X.RED+1
        .
        .

*SAVE VIRTUAL UNIT REGISTERS
STREG   NULL
        STWS    REG+12
        STWS    REG+13
        SPDBR   REG+40
        STO     REG+41
        SZN     SSFALT+.WICI        safe store frame saved?
```

STPn	Store Pointer n	45n (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

$C(ARn) \rightarrow C(Y)_{0-23}$

$C(SEGIDn) \rightarrow C(Y)_{24-35}$

ES Mode

$C(ARn) \rightarrow C(Y)$

$C(SEGIDn) \rightarrow C(Y+1)_{0-11}$

$00...0 \rightarrow C(Y+1)_{12-35}$

EXPLANATION: These instructions store the address register (ARn) and the associated segment identity register, (SEGIDn), in memory. The contents of the registers remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLE:

```
1       8       16                  32
NEPR    EPPR    P0,FANY             error handler
          .
          .
        STP     P0,.SVFLT,,P.SSA    store pointer 0
        LDP     P0,.PS,DL           old argument segment
        LDP     P1,.SSR,DL          safe store
        LDD     P0,0,,P0            get argument 0
        LDD     P1,.WLSR,,P1        get original linkage segment
        LDA     0,,P0               get EPPA pointer
        CNAA    =020160,DL          test null descriptor
        TZE     FANY
```

STPDW	Store PTWAM Directory Word	155 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master Mode

SUMMARY: $C(PTWAM\ Directory)_n$ --> $C(Y)_{00-29}$
00...0 --> $C(Y)_{30-35}$

where: $n = Y_{11-17}$ Y11-16 specifies row
Y17 specifies column of associative memory

EXPLANATION: The contents of the PTWAM directory word n are stored in memory location Y bits 00-29; zeros are stored in bits 30-35. Bits 00-26 represent the combination of working space number and virtual address that is stored in the directory word for future association. Bits 28 and 29 specify the round robin counter for the row in which this directory word is stored in the AM. Bit 27 = 1 specifies that the row in which this directory word is stored is full.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The PTWAM is 64 rows by 2 columns. Bits 25-30 of the virtual address select a row. Thus, the two entries in each row have the same six least-significant bits.

2. This instruction functions whether the PTWAM is ON or OFF. (Refer to the CAMP instruction.)

3. The STPDW instruction inhibits the CPU from carrying out the execute interrupt procedure when the STPDW instruction is executed from an odd memory location, even though the interrupt condition is present and waiting for execution.

4. An IPR fault occurs if illegal address modifications or illegal repeats are used.

5. A Command fault occurs if this instruction is executed in Slave or Master mode.

STPS	Store Parameter Segment Register	751 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(PSR) --> C(Y, Y+1)

EXPLANATION: This instruction stores the current contents of the parameter segment register (PSR) in even and odd memory locations Y and Y+1. The contents of the PSR remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An IPR fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE: (PMME processing)

```
1       8       16              32
        STPS    .STEMP,,P.SSA   STASH PSR
        LDA     .STEMP,,P.SSA
        CANA    .FBT27,DL       ANY PARAMETERS?
        TZE     NOPARM          NO,XFER
        LDP     P1,.PS          0,DL+YES, GET FIRST
```

STPTW	Store PTWAM Register	157 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master Mode

SUMMARY: $C(PTWAM)_n$ --> $C(Y)_{00-35}$

where: $n = Y_{11-17}$ Y11-16 specifies row
Y17 specifies column of associative memory

EXPLANATION: The contents of the PTWAM word n are stored in memory location Y. The absolute memory address (mod 1024) of the referenced page is stored in bits 4-17. Bits 0-3 and 18-29 are stored as zeros. Bits 30-35 are the hardware control field bits in the PTW (bits 30 and 35 are stored as ones).

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The PTWAM is 64 rows by 2 columns. Bits 25-30 of the virtual address select a row. Thus, the two entries in each row have the same six least-significant bits.

2. This instruction functions whether the PTWAM is ON or OFF. (Refer to the CAMP instruction.)

3. The STPTW instruction inhibits the CPU from carrying out the execute interrupt procedure when the STPTW instruction is executed from an odd memory location, even though the interrupt condition is present and waiting for execution.

4. An IPR fault occurs if illegal address modifications or illegal repeats are used.

5. A Command fault occurs if this instruction is executed in Slave or Master mode.

STQ	Store Q-Register	756 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(Q) --> C(Y); C(Q) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL

ILLEGAL REPEATS: RPL

INDICATORS: None affected

NOTE: An IPR fault occurs if illegal address modifications or an illegal repeats are used.

STSS	Store Safe Store Register	753 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master mode

SUMMARY: $C(SSR)_{0-35}$ --> $C(Y)_{0-35}$

$C(SSR)_{36-69}$ --> $C(Y+1)_{0-33}$

The following value is stored in $C(Y+1)_{34,35}$ in accordance with the SCR value.

If C(SCR) = 00/01/11

11 --> $C(Y+1)_{34,35}$ (64-word frame)

If C(SCR) = 10

10 --> $C(Y+1)_{34,35}$ (80-word frame)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An Illegal Procedure fault occurs when illegal address modification or an illegal repeat is used.
2. A Command fault occurs if the processor is in Slave or Master mode and this instruction is executed.

EXAMPLES:

```
1       8       16                  32
SOVTE   NULL
        LDP     P0,SD.PSH,DL        copy push segment descriptor to P0
        LDP     P0,.CTYP,DL         change push descriptor type
        STSS    .SSSR,,P.SSA        store SSR
        LDA     .SSSR+1,,P.SSA      SSR base
        ADA     1K*4,DL             + 1K words
        ORA     =O7777,DL           adjust page bound
        STA     .SVFLT+1,,P.SSA     save it
        SBA     192*4,DL
        EAX2    1,3
        LDQ     PH.SS,,P0           original SSR bound + base
        QRL     16
        ADQ     PH.SS+1,,P0         get max virtual address for safe store
        CMPQ    .SVFLT+1,,P.SSA
        EAX2    0
        SBA     .SSSR+1,,P.SSA      get new bound
        ALS     16
        STA     .SVFLT+1,,P.SSA     store new bound
        LDP     P1,SD.DGS,DL        load DGS segment descriptor
        LDP     P0,SD.DGS,DL
        LDP     P0,.CTYP,DL         change type GDS descriptor
        LXL0    POINT,7
        LDAQ    0,0,P0
        STAQ    .SSSR,,P.SSA        store current contents
        STSS    0,0,P0              store SSR to generate page load segment
        LDA     0,0,P0
        ANA     =O177777,DL
        ORA     .SVFLT+1,,P.SSA     set new bound
        STA     0,0,P0
        LDD     P2,0,0,P1           load new safe store descriptor
```

STT	Store Timer Register	454 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(TR) --> $C(Y)_{0-26}$

00...0 --> $C(Y)_{27-35}$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Bit 26 has a significance of 1/512 millisecond.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STTA	Store Test Address Registers	553 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master Mode

SUMMARY: C(Test Register 0,1) --> C(Y-pair)

EXPLANATION: Contents of test registers 0 and 1 are stored in even/odd memory locations Y and Y+1. Contents of test registers remain unchanged.

This instruction inhibits the processor from carrying out the execute interrupt procedure when the STTA is executed from an odd memory location, even though the interrupt condition is present and waiting for execution.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An Illegal Procedure Fault occurs if illegal address modification or illegal repeats are executed.
2. A Command fault occurs if execution is attempted in Master or Slave mode.

STTD	Store Test Descriptor Registers	550 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master Mode

SUMMARY: C(Test Register 0,1) --> C(Y-pair)

EXPLANATION: Contents of test registers 2 and 3 are stored in even/odd memory locations Y and Y+1. Contents of test registers remain unchanged.

This instruction inhibits the processor from carrying out the execute interrupt procedure when the STTD is executed from an odd memory location, even though the interrupt condition is present and waiting for execution.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An Illegal Procedure Fault occurs if illegal address modification or illegal repeats are executed.
2. A Command fault occurs if execution is attempted in Master or Slave mode.

STWS	Store Working Space Registers	752 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Privileged Master Mode

SUMMARY: When EA_{17} (NS Mode) or EA_{33} (ES Mode) = 0

C(WSR0) --> $C(Y)_{0-8}$

C(WSR1) --> $C(Y)_{9-17}$

C(WSR2) --> $C(Y)_{18-26}$

C(WSR3) --> $C(Y)_{27-35}$

When EA_{17} (NS Mode) or EA_{33} (ES Mode)= 1

C(WSR4) --> $C(Y)_{0-8}$

C(WSR5) --> $C(Y)_{9-17}$

C(WSR6) --> $C(Y)_{18-26}$

C(WSR7) --> $C(Y)_{27-35}$

EXPLANATION: The contents of WSR0 to WSR3, or WSR4 to WSR7 are stored in memory location Y, in accordance with the setting of the EA_{17}/EA_{33} value.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

2. A Command fault occurs if the processor is in Slave or Master mode and this instruction is executed.

EXAMPLE:

```
1       8       16              32
TODES   NULL
        STWS    WSR             store WSR 0-3
        STWS    WSR+1           store WSR 4-7, store contents
         .
         .
        EVEN
WSR     BSS     2
```

STX*n*	Store Index Register *n* in Upper	74*n* (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For n = 0,1,...,7 as determined by op code

$C(Xn) \rightarrow C(Y)_{0-17}$

$C(Y)_{18-35}$ unchanged

ES Mode

For n = 0,1,...,7 as determined by op code

$C(GXn) \rightarrow C(Y)$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, or RPL of STX0

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STZ	Store Zero	450 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: 00...0 --> C(Y)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL

ILLEGAL REPEATS: RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SWCA	Subtract with Carry from A-Register	171 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: If carry indicator is ON

C(A) - C(Y) --> C(A)

C(Y) unchanged

If carry indicator is OFF

C(A) - C(Y) - 00...1 --> C(A)

C(Y) unchanged

EXPLANATION: This instruction is identical to SBA except that, when the carry indicator is OFF at the beginning of the instruction, a positive 1 is subtracted from the least-significant position.

This instruction is intended for use with multiword-precision arithmetic. Thus, the summary above can be reworded as follows:

If carry indicator is ON, then C(A) + one's complement of C(Y) + 00...1 --> C(A)

If carry indicator is OFF, then C(A) + one's complement of C(Y) --> C(A)

The positive 1 is added when ON represents the carry from the next less-significant part of the multiword subtraction.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

SWCA

INDICATORS:	Zero	- If C(A) = 0, then ON; otherwise, OFF
	Negative	- If $C(A)_0$ = 1, then ON; otherwise, OFF
	Overflow	- If range of A is exceeded, then ON
	Carry	- If a carry out of bit 0 of C(A) is generated, then ON; otherwise, OFF

SWCQ	Subtract with Carry from Q-Register	172 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: If carry indicator is ON

C(Q) - C(Y) --> C(Q)

C(Y) unchanged

If carry indicator is OFF

C(Q) - C(Y) - 0...1 --> C(Q)

C(Y) unchanged

EXPLANATION: This instruction is identical to SBQ except that, when the carry indicator is OFF at the beginning of the instruction, a positive 1 is subtracted from the least-significant position.

This instruction is intended for multiword-precision arithmetic. Thus, the summary above can be reworded as follows:

If carry indicator is ON, then C(Q) + one's complement of C(Y) + 00...1 --> C(Q)

If carry indicator is OFF, then C(Q) + one's complement of C(Y) --> C(Q)

The positive 1 is added when ON represents the carry from the next less-significant part of the multiword subtraction.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS:

Zero	-	If C(Q) = 0, then ON; otherwise, OFF
Negative	-	If $C(Q)_0$ = 1, then ON; otherwise, OFF
Overflow	-	If range of Q is exceeded, then ON
Carry	-	If a carry out of bit 0 of C(Q) is generated, then ON; otherwise, OFF

EXAMPLE: (Triple-precision binary fixed-point subtraction)

```
1       8       16              32
        STI     C               set overflow mask ON
        LDA     =1B24,DL
        ORSA    C
        LDI     C
        LDQ     A+2             subtract low-order bits
        SBLQ    B+2
        STQ     C+2
        LDQ     A+1             subtract intermediate bits
        SWCQ    B+1
        STQ     C+1
        STI     C               set overflow and overflow mask OFF
        LDA     =0733777,DL
        ANSA    C
        LDI     C
        LDQ     A               subtract high-order bits
        SWCQ    B
        STQ     C

A       DEC     9,8,7
B       DEC     6,5,4
C       BSS     3
```

SWD
SWDX

SWD SWDX	Subtract Word Displacement from Address Register	527 (1)

FORMAT: Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT: 1 8 16

```
{SWD }   word displacement,R,AR
{SWDX}
```

When the mnemonic is coded with X (AWDX), bit 29 is forced to zero.

OPERATING MODES: Any

SUMMARY: If bit 29 = 1: $C(ARn)_{0-17} - (y + C(DR)) \rightarrow ARn_{0-17}$

If bit 29 = 0: $- y + C(DR) \rightarrow ARn_{0-17}$

In either case, 00...0 $\rightarrow ARn_{18-23}$

EXPLANATION: The y field (with bit 3 extended) is added to the contents of the register specified by the code in the DR field. Then, if bit 29 = 0, this value replaces bits 0-17 of the AR specified by bits 0-2 of the y field. If bit 29 = 1, this value is subtracted from bits 0-17 of the specified AR and the result is stored in bits 0-17 of the specified AR. In either case, bits 18-23 of the specified AR are zeroed.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, or IC specified in DR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SWD
SWDX

EXAMPLE: Applies to NS mode only

```
1       8       16              32
        EAX5    2
        SWDX    2,5,4           AR4 octal contents  -  7 7 7 7 7 4 0 0
        SWD     0,5,4           AR4 octal contents  -  7 7 7 7 7 2 0 0

        EAX4    1
        SWDX    4,4,7           AR7 octal contents  -  7 7 7 7 7 3 0 0
        SWD     1,4,7           AR7 octal contents  -  7 7 7 7 7 1 0 0
```

SXLn	Store Index Register n in Lower	44n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: NS Mode

For N=0,1,...,7 as determined by op code

$C(Xn) \rightarrow C(Y)_{18-35}$

$C(Y)_{0-17}$ unchanged

ES Mode

For N=0,1,...,7 as determined by op code

$C(GXn_{18-35}) \rightarrow C(Y)_{18-35}$

$C(Y)_{0-17}$ unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, or RPL of SXL0

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SYNC	Gate Synchronize	014 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: This instruction operates as a NOP; no operation takes place.

ILLEGAL ADDRESS MODIFICATION: Address modifications are performed, but have no effect on the operation.

ILLEGAL REPEATS: RPD, RPL, RPT

INDICATORS: Address modifications cause defined changes to address and tally. The tally runout indicator may be set ON as a result.

NOTE: An IPR fault occurs if an illegal repeat is executed.

SZN	Set Zero and Negative Indicators from Storage	234 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: C(Y) is tested and the indicators are set in accordance with the result

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

Negative - If $C(Z)_0$ = 1, then ON; otherwise, OFF

Zero	Negative	Relationship
0	0	Number C(Y) > 0
1	0	Number C(Y) = 0
0	1	Number C(Y) < 0

SZNC	Set Zero and Negative Indicators from Storage and Clear	214 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

EXPLANATION: This instruction provides test-and-set operation, required for setting and releasing locks, or for closing and opening gates. C(Y) is tested and the indicators are set in accordance with the result. C(Y) is then zeroed.

This instruction is used for a gating operation in multiple CPU systems. Execution of the next instruction is delayed until the cache-flush request applied to all CPUs has completed.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS: Zero - If C(Z) = 0, then ON; otherwise, OFF

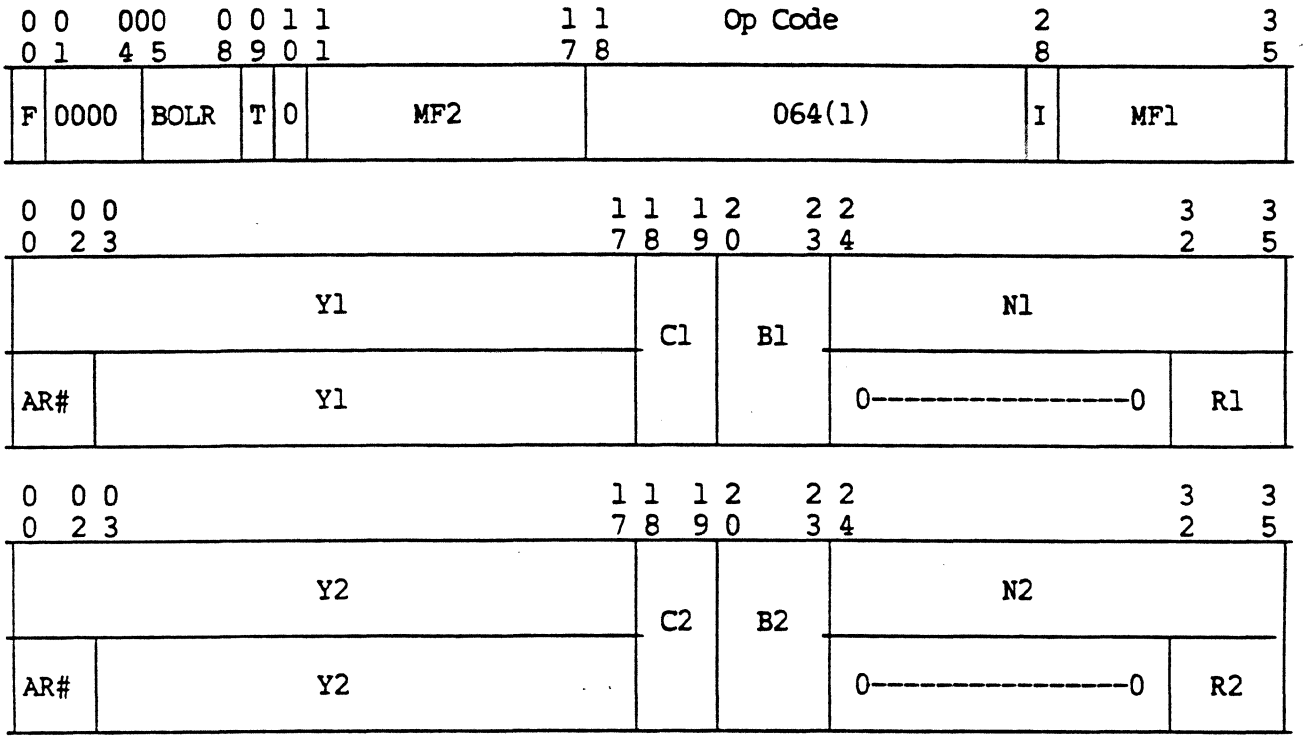
Negative - If $C(Z)_0$ = 1, then ON; otherwise, OFF

Zero	Negative	Relationship
0	0	Number C(Y) > 0
1	0	Number C(Y) = 0
0	1	Number C(Y) < 0

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

SZTL	Set Zero and Truncation Indicators with Bit Strings Left	064 (1)

FORMAT:



CODING FORMAT:

```
1       8       16
        SZTL    (MF1),(MF2),BOLR,F,T
        BDSC    LOCSYM,N,C,B,AM
        BDSC    LOCSYM,N,C,B,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) : (BOLR) : C(string 2)

EXPLANATION: The string of bits starting at location YCB1 is evaluated, bit by bit, with the string starting at location YCB2 until either the resultant bit from the BOLR field is a 1 or until L2 is exhausted. If L1 is greater than L2, the Truncation indicator is set.

If L1 is less than L2, the fill bit (F) is used as the L2-L1 least-significant bits of string 1. The contents of both strings remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero - If all the resultant bits generated are zero, then ON; otherwise, OFF

Truncation - If L1 is > L2, then ON; otherwise, OFF

NOTES:

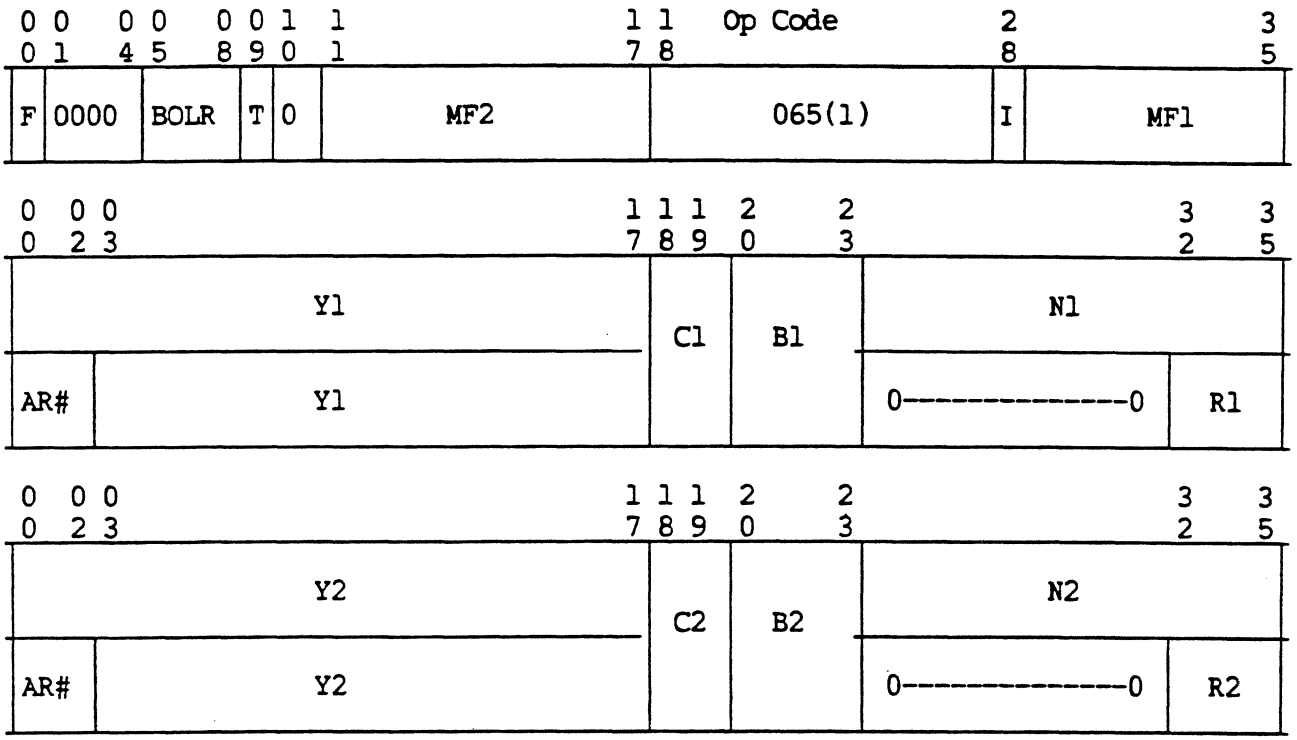
1. An Illegal Procedure fault occurs when illegal address modification or illegal repeats are used.
2. An IPR fault does not occur even when $L_1 = 0$ or $L_2 = 0$. In this case, the zero and truncation indicators are affected.

EXAMPLES:

```
1       8       16              32
        SZTL    ,,6             exclusive OR operation
        BDSC    FLD1,36,0,0     FLD1 operand descriptor
        BDSC    FLD2,35,0,1     FLD2 operand descriptor
        TZE     ALLOFF          zero indicator ON
        TRTN    TRUNC           truncation indicator ON
        USE     CONST.          memory contents in octal
FLD1    DEC     -1              777777777777
FLD2    DEC     -1              777777777777
        USE                     indicators set? - zero and truncation

        LDI     0,DL
        LDX7    -1,DU           load negative value into X7
        STI     FLD1            store processor indicators
        SZTL    ,,1             AND operation
        BDSC    FLD1,1,2,1      FLD1 operand descriptor
        BDSC    FLD2,1,2,1      FLD2 operand descriptor
        TNZ     19ON            not zero - negative indicator ON
        USE     CONST.          memory contents in octal
FLD1    BSS     1               x x x x x x 2 0 0 0 0 0
FLD2    DEC     1B19            0 0 0 0 0 0 2 0 0 0 0 0
        USE                     indicators set? -  none
```

SZTR	Set Zero and Truncation Indicators with Bit Strings Right	065 (1)

FORMAT:



CODING FORMAT: 1 8 16

```
SZTR    (MF1),(MF2),BOLR,F,T
BDSC    LOCSYM,N,C,B,AM
BDSC    LOCSYM,N,C,B,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

SUMMARY: C(string 1) : (BOLR) : C(string 2)

EXPLANATION: Same as for SZTL except that starting locations are YCB1 + (L1-1) and YCB2 + (L2-1) and the evaluation is from right to left (least-significant bit to most significant bit). Any fill (used in comparison) is of most-significant bits.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Same as for SZTL

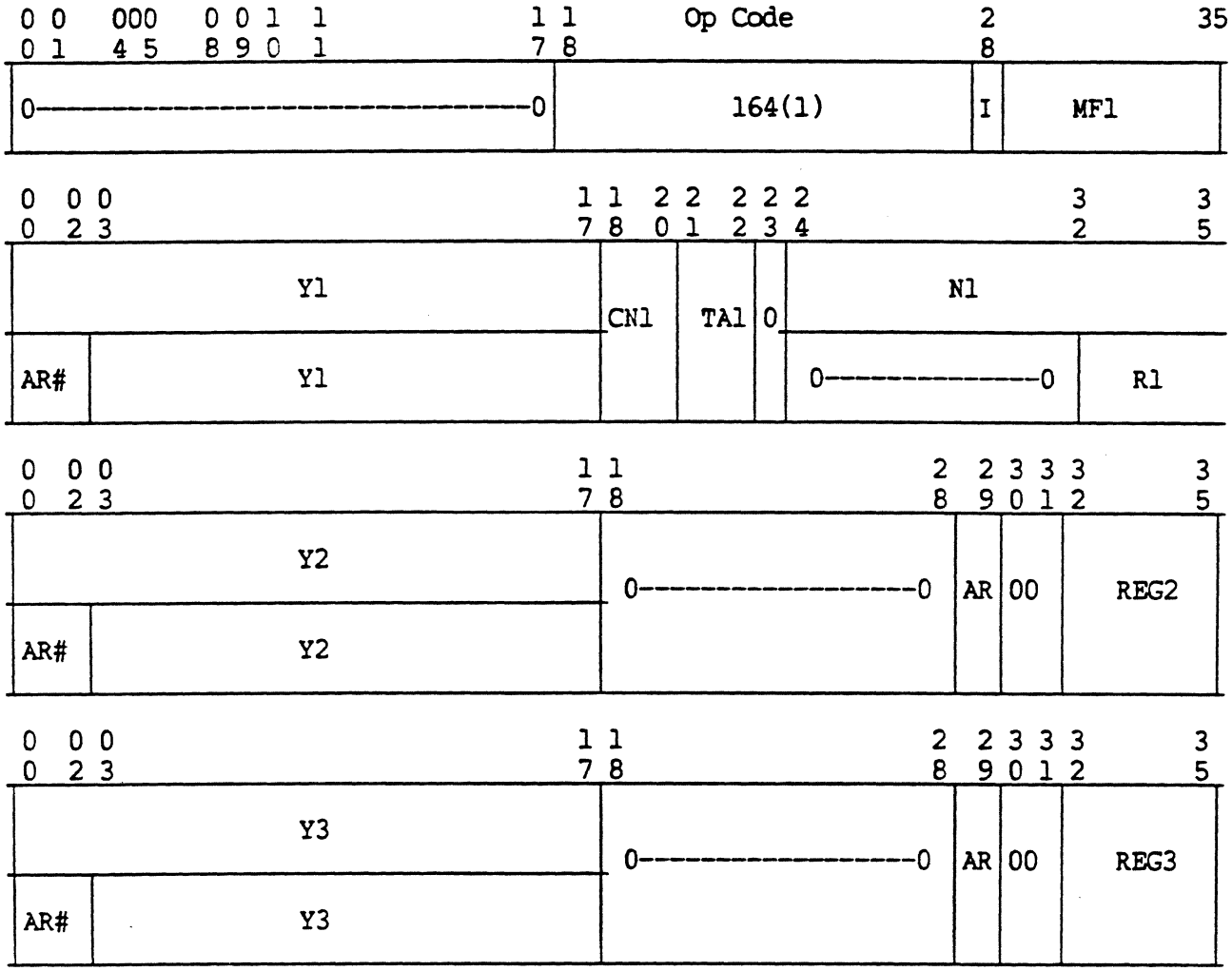
NOTE: Notes for SZTR are the same as for SZTL.

EXAMPLES:

```
1       8       16              32
        SZTR    ,,3,1           evaluate FLD1 as is (move)
        BDSC    FLD1,1,2,1      FLD1 operand descriptor (bit 19)
        BDSC    0,1             FLD2 operand descriptor
        TNZ     19ON
        USE     CONST.          memory contents in octal
FLD1    DEC     1B19            0 0 0 0 0 0 2 0 0 0 0 0
        USE                     indicators set? - none

        LDI     0,DL            clear processor indicators
        LDX7    0,DU            load zeros into X7
        STI     FLD1            store processor indicators
        SZTR    ,,14            invert
        BDSC    FLD1,1,2,0      FLD1 operand descriptor (bit 18)
        BDSC    0,1             FLD2 operand descriptor
        TZE     18ON            zero indicator ON
        USE     CONST.          memory contents in octal
FLD1    BSS     1               x x x x x x 4 0 0 0 0 0
        USE                     indicators set? - zero
```

TCT	Test Character and Translate	164 (1)

FORMAT:



CODING FORMAT:

```
1       8       16
        TCT     (MF1)
        ADSCn   LOCSYM,CN,N,AM
        ARG     LOCSYM,RM,AM
        ARG     LOCSYM,RM,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

EXPLANATION: Starting at location YC1, each type TA1 character is used as an index to a table of 9-bit characters that starts at location Y2. If the table entry is zero, a counter is incremented by 1.

The operation terminates if a nonzero table entry is found or if the tally (L1) is exhausted. At the conclusion of the instruction, the counter contents are stored right-justified in bits 12-35 of Y3. The last accessed table entry is placed in bits 0-8 of Y3. Zeros are placed in bits 9-11 of Y3. Except in cases of string overlap, the contents of the source field and the table remain unchanged. (Refer to Explanation under MVT.)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, REG2, REG3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Tally - If the tally (L1) is exhausted and table entry is zero, then ON; otherwise, OFF

NOTES:

1. If N1=0, zero is stored in Y3 (bits 12-35) and the tally indicator is affected.
2. If N1>0 and a match is found in the first character, zero is stored in Y3 (bits 12-35) and the tally indicator is not affected.
3. An Illegal procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

```
1       8       16                  32
        TCT                         no modification
        ADSC6   FLD1,0,12           indexing string operand descriptor
        ARG     TABLE               pointer to table
        ARG     FLD3                pointer to character and count word
        TTF     FOUND               nonzero character found
        USE     CONST.              memory contents
FLD1    BCI     2, 1234567890#      20010203040506071011 0013 (octal)
FLD3    BSS     1                   character and count - 020000000013
         .
         .
                                                    Octal
                0  1  2  3   4  5  6  7             Index
TABLE   OCT     000000000000,000000000000           0X
        OCT     000000020020,020020020020           1X
        OCT     000000000000                        2X
        USE                  Result - nonzero character found
```

NOTE: The highest possible value in FLD1 is an octal 20, a "blank".

EXAMPLE WITH ADDRESS MODIFICATION:

```
1       8       16                  32

X6      BOOL    16
        EAX2    2                   put 2 into X2
        EAX3    FLD1                put FLD1 address into X3
        EAX6    6                   put FLD1 length into X6
        AWDX    0,3,7               put FLD1 address into AR7
        TCT     (1,1,1,2)           with all modification options
        ARG     INDSCR              pointer indirect operand descriptor
        ARG     TABLE               pointer to table
        ARG     FLD3                pointer to FLD3
        TTF     *+2                 nonzero found
        NULL                        tally runout ON
        USE     CONST.              memory contents

FLD1    ASCII   2,  1234;5          040040061062063064073065  (octal)
FLD3    BSS     1                   character and count 040000000004
INDSCR  ADSC9   0,0,X6,7            indexing FLD1 operand descriptor
                                    (FLD1,2,6)
TABLE   BSS     12                  generate 60 (octal) table characters
        OCT     000000000000,000000000000 (060-067)
        OCT     000000000040              (070-073)
        USE                         Result - nonzero found
```

NOTE: The highest possible value in FLD1 is an octal 073, a ";".

TCTR	Test Character and Translate in Reverse	165 (1)

FORMAT: Same as Test Character and Translate (TCT) format

CODING FORMAT:

```
1       8       16
        TCTR    (MF1)
        ADSCn   LOCSYM,CN,N,AM
        ARG     LOCSYM,RM,AM
        ARG     LOCSYM,RM,AM
```

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES: Any

EXPLANATION: Same as TCT except start at location YC1 + (L1-1) and progress toward YC1.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, REG2, REG3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Tally - If the tally (L1) is exhausted and table entry is zero, then ON; otherwise, OFF

NOTE: Notes for TCTR are the same as for TCT.

EXAMPLE:

```
1       8       16              32
        TCTR                    no modification
        ADSC4   FLD1,6,10       indexing string operand descriptor
        ARG     TABLE           pointer to table
        ARG     FLD3            pointer to character and count word
        TTF     *+2             nonzero found
        NULL                    nonzero not found - tally runout ON
        USE     CONST.          memory contents
FLD1    EDEC    16P1234567890   0000001234567890
FLD3    BSS     1               character and count 000000000012  (octal)

TABLE   OCT     0,0
        OCT     000000014014,000000014014
*Highest possible value (in 4-bit field) in FLD1 is octal 17
        USE                     Result - no illegal character found
```

TEO	Transfer on Exponent Overflow	614 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TEO     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If exponent overflow indicator ON, then Y --> C(IC)

If exponent overflow indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn --> C(SEGID(IS))

ES Mode

If exponent overflow indicator ON, then Y_{16-33} --> C(IC)

If exponent overflow indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Exponent Overflow - Set OFF

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TEU	Transfer on Exponent Underflow	615 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TEU     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If exponent underflow indicator ON, then Y --> C(IC)

If exponent underflow indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If exponent underflow indicator ON, then Y_{16-33} --> C(IC)

If exponent underflow indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Exponent Underflow - Set OFF

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TMI	Transfer on Minus	604 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TMI     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If negative indicator ON, then Y --> C(IC)

If negative indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If negative indicator ON, then Y_{16-33} --> C(IC)

If negative indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TMOZ	Transfer on Minus or Zero	604 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TMOZ    LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMMARY:

NS Mode

If negative indicator ON or Zero indicator ON, then

Y --> C(IC)

If negative indicator ON or Zero indicator ON; and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If negative indicator ON or Zero indicator ON, then

Y_{16-33} --> C(IC)

If negative indicator ON or Zero indicator ON; and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.

- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

```
1       8       16                  32
        LCQ     2,DL
        TMOZ    NOPLUS              transfer on minus or zero
        NULL                        plus routine

*DID TRANSFER OCCUR?    YES     TO WHAT LOCATION?    NOPLUS
```

TNC	Transfer on No Carry	602 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TNC     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If carry indicator OFF, then Y --> C(IC)

If carry indicator OFF and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If carry indicator OFF, then Y_{16-33} --> C(IC)

If carry indicator OFF and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TNZ	Transfer on Nonzero	601 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TNZ     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If zero indicator OFF, then Y --> C(IC)

If zero indicator OFF and instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If zero indicator OFF, then Y_{16-33} --> C(IC)

If zero indicator OFF and instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TOV	Transfer on Overflow	617 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TOV     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If overflow indicator ON, then Y --> C(IC)

If overflow indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If overflow indicator ON, then Y_{16-33} --> C(IC)

If overflow indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Overflow - Set OFF

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TPL	Transfer on Plus	605 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TPL     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If negative indicator OFF, then Y --> C(IC)

If negative indicator OFF and instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If negative indicator OFF, then Y_{16-33} --> C(IC)

If negative indicator OFF and instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TPNZ	Transfer on Plus and Nonzero	605 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TPNZ    LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If negative indicator OFF and Zero indicator OFF, then

Y --> C(IC)

If negative indicator OFF and Zero indicator OFF and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If negative indicator OFF and Zero indicator OFF, then

Y_{16-33} --> C(IC)

If negative indicator OFF and Zero indicator OFF and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.

TPNZ

- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TPNZ

EXAMPLES:

```
1       8       16                  32
-------------------------------------------------------------------

        EAX5    6                   load address modifier into X5
        EAX6    PLUSRT              load transfer address into X6
        AWDX    0,6,6               put transfer address into AR6
        LDA     5,DL                load +5 into A-register
        TPNZ    0,5,6               transfer on plus and nonzero
        NULL                        zero and negative routine

*DID TRANSFER OCCUR?     YES     TO WHAT LOCATION?     PLUSRT+6

        EAX2    3                   load address modifier into X2
        LDX7    4,DU                load +4 into X7
        TPNZ    TRANS,2             transfer on plus and nonzero
        NULL                        zero and negative routine

*DID TRANSFER OCCUR?     YES     TO WHAT  LOCATION?     TRANS+3
```

TRA	Transfer Unconditionally	710 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TRA     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

Y --> C(IC)

If instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

Y_{16-33} --> C(IC)

If instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With unconditional transfer of control instructions, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not affected. An IPR fault does not occur.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer in this case is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TRC	Transfer on Carry	603 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TRC     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If carry indicator ON, then Y --> C(IC)

If carry indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If carry indicator ON, then Y_{16-33} --> C(IC)

If carry indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TRCTn	Transfer on Count n	54n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TRCTn   LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

For n = 0,1...,7 as determined by op code

If zero indicator OFF and negative indicator ON

then C(Xn) - 1 --> C(Xn)

If C(Xn) ≠ 0, Y --> C(IC)

If zero indicator OFF and negative indicator ON and instruction bit 29=1 then

m = Y_{0-2}

C(DRm) --> C(ISR); C(SEGIDm) --> C(SEGID(IS))

ES Mode

For n = 0,1...,7 as determined by op code

If zero indicator OFF and negative indicator ON

then C(GXn) - 1 --> C(Xn)

IF C(GXn) ≠ 0, Y_{16-33} --> C(IC)

If zero indicator OFF and negative indicator ON and instruction bit 29=1 then

m = Y_{0-2}

C(DRm) --> C(ISR); C(SEGIDm) --> C(SEGID(IS))

EXPLANATION: A 1 is subtracted from the content of Xn/GXn and the result is loaded into Xn/GXn. Unless the content of the result in Xn/GXn is zero, control is transferred to the location specified by the y field. If the result is 0, the next instruction is executed.

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRm selected with bits 0, 1, 2, and the corresponding SEGIDm, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRm (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRm, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

ILLEGAL EXECUTES: XEC, XED

INDICATORS:

Zero - If C(Xn/GXn) = 0, then ON; otherwise, OFF

Negative - If C(Xn/GXn) = 1, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications, illegal repeats, or illegal executes are used.

EXAMPLE:

```
1       8       16              32
        LDX0    10,DU
          .
          .
A       LDA
          .
          .
        TRTC0   A
```

TRTF	Transfer on Truncation Indicator OFF	601 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TRTF    LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If truncation indicator OFF, then Y --> C(IC)

If truncation indicator OFF and instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If truncation indicator OFF, then Y_{16-33}IC)

If truncation indicator OFF and instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

```
         1        8        16                  32
                  MLR                          move alphanumeric left to right
                  ADSC9    FLD1,0,4            sending operand descriptor
                  ADSC4    FLD2,0,4            receiving operand descriptor
                  TRTF     NTRUNC              truncation indicator OFF
                  NULL
*
          *Did transfer to NTRUNC occur?        YES
*
          *State of truncation indicator after? OFF
```

TRTN	Transfer on Truncation Indicator ON	600 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TRTN    LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If truncation indicator ON, then Y --> C(IC)

If truncation indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If truncation indicator ON, then Y_{16-33} --> C(IC)

If truncation indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Truncation - If ON, it is turned OFF

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

```
1       8       16              32
        MLR                     move alphanumeric left to right
        ADSC4   FLD1,0,8        sending operand descriptor
        ADSC6   FLD2,0,6        receiving operand descriptor
        TRTN    TRUNC           truncation indicator ON
        TRA     TRUNC+6         truncation indicator OFF

*To where was transfer?            TRUNC

*State of truncation indicator after?     OFF

        MLR                     move alphanumeric left to right
        ADSC9   FLD1,0,8        sending operand descriptor
        ADSC4   FLD2,0,4        receiving operand descriptor
        TRTN    TRUNC           truncation indicator ON
        NULL                    no truncation routine

*Did transfer of control occur?     yes     where to?     TRUNC

*State of truncation indicator after?     OFF
```

TSS	Transfer After Setting Slave	715 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TSS     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

Y --> C(IC)

If instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

Y_{16-33} --> C(IC)

If instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: All outstanding memory requests are checked for completion before the Master Mode indicator is reset on the TSS instruction.

With unconditional transfer of control instructions, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not affected. An IPR fault does not occur even when bit 29 of the TSS instruction word is 0.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Master Mode - Set OFF

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TSS

6. For a fault that occurs as a result of execution of a TSS instruction in Master mode, the state of bit 28 (Master Mode indicator) in the copy of the indicator register stored in the safe store frame is as follows:

   - If IPR or Fault Tag fault, caused by the tag field in the instruction or indirect word, then IR28 = 1.
   - If Bound fault, caused by attempt to access an indirect word, then IR28 = 1.
   - If Bound fault, caused by attempt to access the target location then IR28 = 1.

7. Use of the TSS instruction does not change the contents of the DR or AR registers which may have been set by previous Master Mode Entry (MME/PMME) and/or user code.

TSXn	Transfer and Set Index Register n	70n (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TSXn    LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

For n = 0,1,...,7 as determined by op code

C(IC) + 0...01 --> C(Xn); Y --> C(IC)

If instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

For n = 0,1,...,7 as determined by op code

00...0--> $C(GXn)_{0-17}$

C(IC) + 0...01 --> $C(GXn)_{18-35}$;

(no transfer of a carry from bit 18 of GXn to high-order bit)

Y_{16-33} --> C(IC)

If instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With unconditional transfer of control instructions, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not affected. An IPR fault does not occur.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TTF	Transfer on Tally Runout Indicator OFF	607 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TTF     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If tally runout indicator OFF, then Y --> C(IC)

If tally runout indicator OFF and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If tally runout indicator OFF, then Y_{16-33} --> C(IC)

If tally runout indicator OFF and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TTN	Transfer on Tally Runout Indicator ON	606 (1)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT: The TTN instruction is coded as follows:

```
1       8       16
        TTN     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If tally runout indicator ON, then Y --> C(IC)

If tally runout indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If tally runout indicator ON, then Y_{16-33} --> C(IC)

If tally runout indicator ON and instruction bit 29=1 then

$n = Y_{0-2}$

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

```
1       8       16              32
        TCT                     test character and translate
        ADSC6   FLD1,0,12       indexing string operand descriptor
        ARG     TABLE           pointer to table
        ARG     FLD3            operand pointer to count word
        TTN     NMATCH          tally runout ON - nonzero entry
        NULL                    tally runout OFF
        USE     CONST.

TABLE   OCT     ,,20020,020020020020,0
FLD1    BCI     2, 1234567890#
FLD3    BSS     1
        USE

*Did transfer occur?     no

        TCT                     test character and translate
        ADSC4   FLD1,0,8        indexing string operand descriptor
        ARG     TABLE           pointer to table
        ARG     FLD3            pointer to character and count word
        TTN     CHAROK          tally runout ON
        TRA     ERROR           tally runout OFF
        USE     CONST.
TABLE   OCT     ,,14014,14014
FLD1    OCT     022064126317
        USE

*To what location was transfer made?     ERROR
```

TZE	Transfer on Zero	600 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

CODING FORMAT:

```
1       8       16
        TZE     LOCSYM,RM,AM
```

OPERATING MODES: Any

SUMMARY:

NS Mode

If zero indicator ON, then Y --> C(IC)

If zero indicator ON and instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

ES Mode

If zero indicator ON, then Y_{16-33} --> C(IC)

If zero indicator ON and instruction bit 29=1 then

n = Y_{0-2}

C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))

EXPLANATION: With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DRn selected with bits 0, 1, 2, and the corresponding SEGIDn, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DRn (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DRn, an IPR fault occurs. The ISR bit can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes; or has a bound that is not 31 modulo 32 bytes.

2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.

3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.

4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

UFA	Unnormalized Floating Add	435 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) + C(Y)] not normalized --> C(EAQ)

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	-	If C(AQ) = 0, then ON; otherwise OFF
Negative	-	If $C(AQ)_0$ = 1, then ON; otherwise OFF
Exponent Overflow	-	If exponent is > +127, then ON
Exponent Underflow	-	If exponent is < -128, then ON
Carry	-	If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32=1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

UFA

EXAMPLE: (Convert from floating to fixed)

```
1       8       16                32
FIXIT   MACRO
        INE     #1,'.EAQ.',1
        FLD     #1
        FCMP    -0110400,DU       2**35
        TMI     2,IC
        NOP     ,F
        FCMP    =0107000,DU       -2**35
        TMI     02,IC
        UFA     =71B25,DU
        INE     #2,'.QR.',1
        STQ     #2
        ENDM    FIXIT
        FIXIT   X,I               I=X
```

UFM	Unnormalized Floating Multiply	421 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) * C(Y)] not normalized --> C(EAQ)

EXPLANATION: This multiplication is executed like the FMP instruction except that the final normalization is performed only if both factor mantissas are = - 1.00...0. The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent is < -128, then ON

NOTES:

1. When indicator bit 32=1, the floating-point alignment and normalization is hexadecimal. Otherwise, the floating-point alignment and normalization are binary.

2. An Illegal Procedure fault occurs if illegal address modification is used.

UFS	Unnormalized Floating Subtract	535 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: [C(EAQ) - C(Y)] not normalized --> C(EAQ)

EXPLANATION: The two's complement of the subtrahend is first taken and the smaller value is then right-shifted to equalize it. The shifted-out portion is truncated and addition is executed.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

Exponent Overflow - If exponent is > +127, then ON

Exponent Underflow - If exponent is < -128, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32=1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary.

2. An Illegal Procedure fault occurs if illegal address modification is used.

UFTR	Unnormalized Floating Truncate Fraction	434 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: C(EAQ) fraction-truncated --> C(EAQ)

EXPLANATION: This instruction truncates the fraction part of the floating-point data of C(EAQ) to obtain an integer. The result is unnormalized and stored into C(EAQ). A proper truncation to an integer is such that truncating the fraction parts of two numbers with the same absolute and different sign and adding the results produces 0.

ILLEGAL ADDRESS MODIFICATIONS: None. The address modification does not affect instruction operations, but the modification is executed.

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if an illegal repeat is used.

XEC	Execute	716 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Any

SUMMARY: Obtain and execute the instruction stored at memory location Y.

EXPLANATION: The next instruction to be executed is obtained from C(IC)+1. This is the instruction located in memory immediately following the location containing the XEC instruction. This does not apply if the execution of the instruction obtained from location Y changes the content of the IC.

To execute a repeat instruction with the XEC instruction, the XEC must reside at an odd location. The instructions to be repeated using the RPT, RPD, or RPL instructions must immediately follow the XEC instruction.

With the exceptions noted in Note 1, an XEC instruction may point to a multiword instruction. However, the descriptors for the multiword instruction must be stored immediately following the XEC instruction. The next instruction to be executed is obtained from C(IC)+n+1, where n is the number of descriptors for the multiword instruction.

If IC modification is used with the instruction being executed, the value of IC will be the same as the location of the XEC instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: The XEC instruction itself does not affect any indicator. However, the execution of the instruction from Y may affect indicators.

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or illegal repeats are used when the XEC instruction is executing an SPL, LPL, CLIMB, or TRCTn

2. An Illegal Procedure fault occurs if a CLIMB is executed via an XEC instruction.

EXAMPLE:

```
1       8       16              32
        REM                     X7 has value 0 or 1
        REM                     X6 has value 1, 2, 3, 4 or 5
        XEC     DOIT,7          add or subtract
        USE     SMARTS
DOIT    ADQ     FF
        SBQ     FF
        USE
        XEC     BRANCH-1,6      5-way branch
        USE     YERHED
BRANCH  NOP
        AOS     FLAG2
        TRA     .S3
        TRA     .S4
        TRA     WRAPUP
        USE
```

XED	Execute Double	717 (0)

FORMAT: Single-word instruction format (see Figure 8-1)

OPERATING MODES: Executes in NS mode only

SUMMARY: Obtain and execute the two instructions stored at the memory Y-pair locations (must be even and next odd location).

EXPLANATION: The first instruction obtained from Y-pair must not alter the memory location from which the second instruction is obtained, and must not be another XED instruction.

If the first instruction obtained from Y-pair alters the contents of the instruction counter, this transfer of control is effective immediately, and the second instruction of the pair is not executed.

After execution of the two instructions obtained from the Y-pair, the next instruction to be executed is obtained from C(IC)+1. This location immediately follows the XED instruction. This does not apply if the execution of the two instructions obtained from the y-pair alters the content of the IC.

To Execute Double (XED) the RPD instruction , the RPD must be the second instruction at an odd-numbered address. When RPD is at the odd-numbered address of the pair, the XED instruction must be at an odd location. In this case, the repeated instructions are those that immediately follow the XED instruction. If RPD is specified within a sequence of XEDs, the original and all subsequent XEDs in the sequence must be in odd locations.

When repeat instructions RPT or RPL are executed with an XED instruction and the first instruction specified by the XED resides an an even-numbered location, the repeated instruction is that immediately following the RPT or RPL. When the RPT or RPL instruction resides at an odd-numbered address, the repeated instruction is that immediately following the XED instruction.

With the exceptions noted in Note 1, multiword instructions are executed with the XED instruction. The multiword instruction (second instruction) must be located at an odd-numbered address. If it is not, an IPR fault occurs. The data descriptors for this multiword instruction immediately follow the XED instruction.

If IC modification is used with either of the instructions being executed, the value of IC will be the same as the location of the XED instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

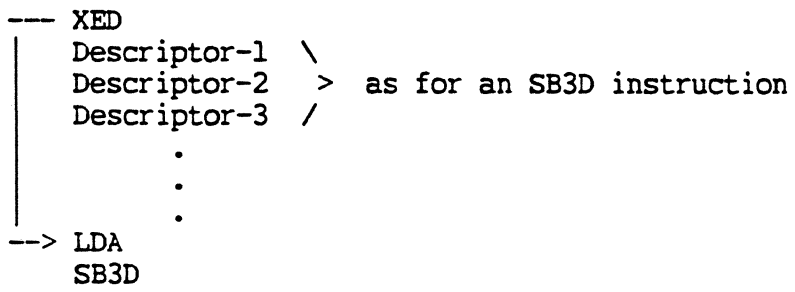
ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: The XED instruction itself does not affect any indicator. However, the execution of the two instructions from Y-pair may affect indicators.

NOTES:

1. An IPR fault occurs if XED instruction is used with SPL, LPL, CLIMB, or TRCTn.

2. When multiword instructions other than those indicated in Note 1 are executed with an XED instruction, the multiword instruction must be located at an odd-numbered address (second instruction). If it is not, an IPR fault occurs. The data descriptors for this multiword instruction are those immediately following the XED instruction as indicated below:



3. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

4. An Illegal Procedure fault occurs if execution is attempted in ES mode.

EXAMPLES:

```
1       8       16              32
                                
        REM                     X7 0 = 0,2,4, or 6
        XED     ENTRY,7
        .
        .
        .
        EVEN
ENTRY   NULL
        STC1    SAVE1
        TRA     FIRST
        STC1    SAVE2
        TRA     SECOND
        STC1    SAVE3
        TRA     THIRD
        STC1    SAVE4
        TRA     FOURTH
```

APPENDIX A

OPERATION CODE MAPS

The operation code maps for the processor are shown in in Tables A-1 and A-2. The operation codes are separated into sections: the first section lists operation codes with bit 27 = 0 and the second section with bit 27 = 1.

Table A-1. Operation Code Map (Bit 27 = 0)

Upper 6 bits \ Lower 3 bits	0	1	2	3	4	5	6	7
00		MME	DRL					
01		NOP	PULS1	PULS2	SYNC	CIOC	LCON	
02	ADLX0	ADLX1	ADLX2	ADLX3	ADLX4	ADLX5	ADLX6	ADLX7
03			LDQC	ADL	LDAC	ADLA	ADLQ	ADLAQ
04	ASX0	ASX1	ASX2	ASX3	ASX4	ASX5	ASX6	ASX7
05					AOS	ASA	ASQ	SSCR
06	ADX0	ADX1	ADX2	ADX3	ADX4	ADX5	ADX6	ADX7
07		AWCA	AWCQ	LREG		ADA	ADQ	ADAQ
10	CMPX0	CMPX1	CMPX2	CMPX3	CMPX4	CMPX5	CMPX6	CMPX7
11		CWL				CMPA	CMPQ	CMPAQ
12	SBLX0	SBLX1	SBLX2	SBLX3	SBLX4	SBLX5	SBLX6	SBLX7
13						SBLA	SBLQ	SBLAQ
14	SSX0	SSX1	SSX2	SSX3	SSX4	SSX5	SSX6	SSX7
15						SSA	SSQ	
16	SBX0	SBX1	SBX2	SBX3	SBX4	SBX5	SBX6	SBX7
17		SWCA	SWCQ			SBA	SBQ	SBAQ
20	CNAX0	CNAX1	CNAX2	CNAX3	CNAX4	CNAX5	CNAX6	CNAX7
21		CMK			SZNC	CNAA	CNAQ	CNAAQ
22	LDX0	LDX1	LDX2	LDX3	LDX4	LDX5	LDX6	LDX7
23		RSW	WRES	RIMR	SZN	LDA	LDQ	LDAQ
24	ORSX0	ORSX1	ORSX2	ORSX3	ORSX4	ORSX5	ORSX6	ORSX7
25	RCW					ORSA	ORSQ	
26	ORX0	ORX1	ORX2	ORX3	ORX4	ORX5	ORX6	ORX7
27	RMR	SMR	SMID	RMID		ORA	ORQ	ORAQ
30	CANX0	CANX1	CANX2	CANX3	CANX4	CANX5	CANX6	CANX7
31						CANA	CANQ	CANAQ
32	LCX0	LCX1	LCX2	LCX3	LCX4	LCX5	LCX6	LCX7
33						LCA	LCQ	LCAQ
34	ANSX0	ANSX1	ANSX2	ANSX3	ANSX4	ANSX5	ANSX6	ANSX7
35					STAC	ANSA	ANSQ	
36	ANX0	ANX1	ANX2	ANX3	ANX4	ANX5	ANX6	ANX7
37						ANA	ANQ	ANAQ

Table A-1 (cont). Operation Code Map (Bit 27 = 0)

Upper 6 bits \ Lower 3 bits	0	1	2	3	4	5	6	7
40		MPF	MPY			CMG		
41		LDE	RIW	RSCR		ADE		
42		UFM	SCPR	DUFM		FCMG		DFCMG
43	FSZN	FLD	QFLD	DFLD	UFTR	UFA		DUFA
44	SXL0	SXL1	SXL2	SXL3	SXL4	SXL5	SXL6	SXL7
45	STZ	SIW	SFR	QFST	STT	FST	STE	DFST
46	QSMP	FMP	QFMP	DFMP		FSBI	QFSTR	DFSBI
47	FSTR	FRD	DFSTR	DFRD	FTR	FAD	QFAD	DFAD
50	RPL					BCD	DIV	DVF
51				FNEG		FCMP		DFCMP
52	RPT					FDI		DFDI
53	FLP	NEG	DFLP	NEGL		UFS		DUFS
54	TRCT0	TRCT1	TRCT2	TRCT3	TRCT4	TRCT5	TRCT6	TRCT7
55	SBAR	STBA	STBQ	LIMR	STC1			
56	RPD					FDV		DFDV
57				FNO		FSB	QFSB	DFSB
60	TZE	TNZ	TNC	TRC	TMI	TPL		TTF
61		RPAT			TEO	TEU	DIS	TOV
62	EAX0	EAX1	EAX2	EAX3	EAX4	EAX5	EAX6	EAX7
63	RET				LDI	EAA	EAQ	LDT
64	ERSX0	ERSX1	ERSX2	ERSX3	ERSX4	ERSX5	ERSX6	ERSX7
65					STACQ	ERSA	ERSQ	
66	ERX0	ERX1	ERX2	ERX3	ERX4	ERX5	ERX6	ERX7
67					LCPR	ERA	ERQ	ERAQ
70	TSX0	TSX1	TSX2	TSX3	TSX4	TSX5	TSX6	TSX7
71	TRA		LRMB			TSS	XEC	XED
72	LXL0	LXL1	LXL2	LXL3	LXL4	LXL5	LXL6	LXL7
73		ARS	QRS	LRS		ALS	QLS	LLS
74	STX0	STX1	STX2	STX3	STX4	STX5	STX6	STX7
75	STC2	STCA	STCQ	SREG	STI	STA	STQ	STAQ
76								
77		ARL	QRL	LRL	GTB	ALR	QLR	LLR

Table A-2. Operation Code Map (Bit 27 = 1)

Upper 6 bits \ Lower 3 bits	0	1	2	3	4	5	6	7
00					MVNEX			
01		CCAC						
02	MVE				MVNE			
03								
04	MPX0	MPX1	MPX2	MPX3	MPX4	MPX5	MPX6	MPX7
05	STD0	STD1	STD2	STD3	STD4	STD5	STD6	STD7
06	CSL	CSR			SZTL	SZTR	CMPB	
07								
10	MLR	MRL					CMPC	
11	SDR0	SDR1	SDR2	SDR3	SDR4	SDR5	SDR6	SDR7
12	SCD	SCDR			SCM	SCMR		
13								
14	GSTD0		GSTD2		GSTD4		GSTD6	
15	STDSA	SPDBR	STO					
16	MVT				TCT	TCTR	CMPCT	
17	LDDSA	LPDBR	LDO				PAS	
20			AD2D	SB2D			MP2D	DV2D
21								
22			AD3D	SB3D			MP3D	DV3D
23								
24			AD2DX	SB2DX			MP2DX	DV2DX
25		SPCF						
26			AD3DX	SB3DX			MP3DX	DV3DX
27								
30	MVN	BTD		CMPN		DTB		
31								
32	GLDD0		GLDD2		GLDD4		GLDD6	
33								
34	MVNX			CMPNX				
35								
36		MTR				MTM		
37								

Table A-2 (cont). Operation Code Map (Bit 27 = 1)

Upper 6 bits \ Lower 3 bits	0	1	2	3	4	5	6	7
40								
41			EPAT					
42								
43	LDRR	LDCR	LDPR	LDDR	ADRR	ADLR	SBRR	SBLR
44				SAREG				SPL
45	STP0	STP1	STP2	STP3	STP4	STP5	STP6	STP7
46	GRS	GRL	GLS	LAREG	GLRS	GLRL	GLLS	LPL
47	LDP0	LDP1	LDP2	LDP3	LDP4	LDP5	LDP6	LDP7
50	A9BD	A6BD	A4DB	ABD				AWD
51								
52	S9BD	S6BD	S4BD	SBD				SWD
53	MPRR	MPRS	CAMP	DVRR	CMRR	ANRR	ORRR	ERRR
54	ARA0	ARA1	ARA2	ARA3	ARA4	ARA5	ARA6	ARA7
55	STTD	STDSD		STTA				
56	AAR0	AAR1	AAR2	AAR3	AAR4	AAR5	AAR6	AAR7
57		LDDSD						
60	TRTN	TRTF			TMOZ	TPNZ	TTN	
61	LDEA0	LDEA1	LDEA2	LDEA3	LDEA4	LDEA5	LDEA6	LDEA7
62								
63	EPPR0	EPPR1	EPPR2	EPPR3	EPPR4	EPPR5	EPPR6	EPPR7
64	ARN0	ARN1	ARN2	ARN3	ARN4	ARN5	ARN6	ARN7
65								
66	NAR0	NAR1	NAR2	NAR3	NAR4	NAR5	NAR6	NRA7
67	LDD0	LDD1	LDD2	LDD3	LDD4	LDD5	LDD6	LDD7
70								
71				CLIMB				
72								
73								
74	SAR0	SAR1	SAR2	SAR3	SAR4	SAR5	SAR6	SAR7
75	STAS	STPS	STWS	STSS				
76	LAR0	LAR1	LAR2	LAR3	LAR4	LAR5	LAR6	LAR7
77	LDAS	LDPS	LDWS	LDSS				

APPENDIX B

OBSOLETE INSTRUCTION CODES

This appendix lists instruction mnemonic codes which have either (1) been obsoleted by new instructions or (2) been deleted from the DPS 8000 instruction repertoire.

All hardware instructions pointed out with an (*) are not supported by the GMAP software. Use of these opcodes are only valid in ES mode.

1. GMAP instructions which replace former instructions yielding the same opcode.

Valid DPS 8000 Instruction	Former Usage
* ANRR 535(1) AND Register to Register	CAMS1 Clear Associative Memory Segment
CAMP 532(1) Clear Associative Mem. Paged	CAMP2 Clear Associative Memory Pages
* CMRR 534(1) Compare Register to Register	CAMS0 Clear Associative Memory Segment
* DVRR 533(1) Divide Register by Register	CAMP3 Clear Associative Memory Pages
SSCR 057(0) Set System Controller Reg.	LCCL Load Calendar Clock
LIMR 553(0) Load Interrupt Mask Register	SMCM Set Memory Continuous Mask Reg.
LCPR 674(0) Load Central Processor Reg.	LLUF Load Lockup Fault Register
* MPPR 530(1) Multiply Reg. Pair by Reg	CAMP0 Clear Associative Memory Pages
* MPRS 531(1) Multiply Reg. by Reg	CAMP1 Clear Associative Memory Pages
RSCR 413(0) Read System Controller Reg	RCCL Read Calendar Clock
RIMR 233(0) Read Interrupt Mask Reg.	RMCM Read Memory Controller Mask Reg.

Valid DPS 8000 Instruction	Former Usage
RSW 231(0) Read Processor Model Characteristics	RRES Read Reserved Memory
SCPR 452(0) Store CPU Register	SFR Store Fault Registers
SIW 451(0) Set Interrupt Word Pair	SMIC Set Memory Controller Interrupt Cells

2. The GMAP instructions in the following list are not valid for DPS 8000 and if executed result in an IPR fault.

ABSA 212(0)	CCAC0 376(1)	CCAC1 377(1)	LBAR 230(0)
LBER 572(0)	LDAB 374(1)	LDAT 336(1)	LDCB 375(1)
LDFB 314(1)	LDHB 334(1)	LDHC 337(1)	LFR 316(1)
LGCOS 356(1)	LHFER 317(1)	LHPT 335(1)	LHTR 315(1)
LMBA 570(0)	LMBB 571(0)	LMSD 354(1)	LVMS 355(1)
MLDA 235(1)	MLDAQ 237(1)	MLDQ 236(1)	MMF 364(1)
MRF 360(1)	MSTA 755(1)	MSTAQ 757(1)	MSTQ 756(1)
SBER 157(0)	SMBA 555(0)	SMBB 556(0)	STAC 354(0)
STBZ 157(0)	STTA 553(1)	STTD 550(1)	TTES 521(0)
TTEZ 524(0)	TTTL 522(0)	TTTU 523(0)	

UNIFIED CHARACTER SET - ASCII SEQUENCE

NOTE	SYMBOL	NAME	ASCII CODE 16 8	EBCDIC CODE 16 8		GBCD CODE 16 8		HBCD CODE 16 8	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
3	NUL	Null	00 000	00 000		1F 37		1E 36	12-0-1-8-9		
3	SOH	Start of Heading	01 001	01 001		1F 37		1E 36	12-1-9		
3	STX	Start of Text	02 002	02 002		1F 37		1E 36	12-2-9		
3	ETX	End of Text	03 003	03 003		1F 37		1E 36	12-3-9		
3	EOT	End of Transmission	04 004	37 067		1F 37		1E 36	7-9		
3	ENQ	Enquiry	05 005	2D 055		1F 37		1E 36	0-5-8-9		
3	ACK	Acknowledge	06 006	2E 056		1F 37		1E 36	0-6-8-9		
3	BEL	Bell (Audible Signal)	07 007	2F 057		1F 37		1E 36	0-7-8-9		
3	BS	Backspace	08 010	16 026		1F 37		1E 36	11-8-9		
3	HT	Horizontal Tab (Punch Card Skip)	09 011	05 005		1F 37		1E 36	12-5-9		
3	LF	Line Feed	0A 012	25 045		1F 37		1E 36	0-5-9		
3	VT	Vertical Tabulation	0B 013	0B 013		1F 37		1E 36	12-3-8-9		
3	FF	Form Feed	0C 014	0C 014		1F 37		1E 36	12-4-8-9		
3	CR	Carriage Return	0D 015	0D 015		1F 37		1E 36	12-5-8-9		
3	SO	Shift Out	0E 016	0E 016		1F 37		1E 36	12-6-8-9		
3	SI	Shift In	0F 017	0F 017		1F 37		1E 36	12-7-8-9		
3	DLE	Data Link Escape	10 020	10 020		1F 37		1E 36	12-11-1-8-9		
3	DC1	Device Control 1	11 021	11 021		1F 37		1E 36	11-1-9		
3	DC2	Device Control 2	12 022	12 022		1F 37		1E 36	11-2-9		
3,5	DC3	Device Control 3	13 023	13 023		1F 37		1E 36	11-3-9		
3	DC4	Device Control 4 (Stop)	14 024	3C 074		1F 37		1E 36	4-8-9		
3	NAK	Negative Acknowledge	15 025	3D 075		1F 37		1E 36	5-8-9		
3	SYN	Synchronous Idle	16 026	32 062		1F 37		1E 36	2-9		
3	ETB	End of Transmission Block	17 027	26 046		1F 37		1E 36	0-6-9		
3	CAN	Cancel	18 030	18 030		1F 37		1E 36	11-8-9		
3	EM	End of Medium	19 031	19 031		1F 37		1E 36	11-1-8-9		
3	SUB	Substitute	1A 032	3F 077		1F 37		1E 36	7-8-9		
3	ESC	Escape	1B 033	27 047		1F 37		1E 36	0-7-9		
3	IFS	File Separator	1C 034	1C 034		1F 37		1E 36	11-4-8-9		
3	IGS	Group Separator	1D 035	1D 035		1F 37		1E 36	11-5-8-9		
3	IRS	Record Separator	1E 036	1E 036		1F 37		1E 36	11-6-8-9		
3	IUS	Unit Separator	1F 037	1F 037		1F 37		1E 36	11-7-8-9		
		Space, Blank	20 040	40 100		10 20		0D 15	Blank	Blank	Blank
	!	Exclamation Point (ASCII)	21 041	4F 117	CI	3F 77	R	3D 75	12-7-8	0-7-8	0-5-8
	"	Double Quote	22 042	7F 177		3E 76		2D 55	7-8	0-8-8	11-5-8
2,9	#	Number Sign	23 043	7B 173		0B 13		2A 52	3-8	3-8	11-2-8
2,9	$	Dollar Sign	24 044	5B 133		2B 53		2B 53	11-3-8	11-3-8	11-3-8
	%	Percent Sign	25 045	6C 154		3C 74		1D 35	0-4-8	0-4-8	12-5-8
	&	Ampersand	26 046	50 120		1A 32		0F 17	12	12	7-8
2	'	Apostrophe	27 047	7D 175		2F 57		0A 12	5-8	11-7-8	2-8
	(	Left Parenthesis	28 050	4D 115		1D 35		3C 74	12-5-8	12-5-8	0-4-8
	)	Right Parenthesis	29 051	5D 135		2D 55		1C 34	11-5-8	11-5-8	12-4-8
	*	Asterisk	2A 052	5C 134		2C 54		2C 54	11-4-8	11-4-8	11-4-8
7	+	Plus	2B 053	4E 116		30 60		10 20	12-6-8	12-0	(12-0)(12)
	,	Comma	2C 054	6B 153		3B 73		3B 73	0-3-8	0-3-8	0-3-8
7	-	Hyphen, Minus	2D 055	60 140		2A 52		20 40	11	11	(11-0)(11)
	.	Period	2E 056	4B 113		1B 33		1B 33	12-3-8	12-3-8	12-3-8
	/	Slash	2F 057	61 141		31 61		31 61	0-1	0-1	0-1

NOTE	SYMBOL	NAME	ASCII CODE 16	8	EBCDIC CODE 16	8	GBCD CODE 16	8	HBCD CODE 16	8	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
	0		30	060	F0	360	00	00	00	00	0	0	0
	1		31	061	F1	361	01	01	01	01	1	1	1
	2		32	062	F2	362	02	02	02	02	2	2	2
	3		33	063	F3	363	03	03	03	03	3	3	3
	4		34	064	F4	364	04	04	04	04	4	4	4
	5		35	065	F5	365	05	05	05	05	5	5	5
	6		36	066	F6	366	06	06	06	06	6	6	6
	7		37	067	F7	367	07	07	07	07	7	7	7
	8		38	070	F8	370	08	10	08	10	8	8	8
	9		39	071	F9	371	09	11	09	11	9	9	9
	:	Colon	3A	072	7A	172	0D	15	0C	14	2-8	5-8	4-8
	;	Semi-Colon	3B	073	5E	136	2E	56	1A	32	11-6-8	11-6-8	12-2-8
	<	Less Than	3C	074	4C	114	1E	36	30	60	12-4-8	12-6-8	5-8
	=	Equal	3D	075	7E	176	3D	75	0B	13	6-8	0-5-8	3-8
	>	Greater Than	3E	076	6E	156	0E	16	0E	16	0-6-8	6-8	6-8
7	?	Question Mark	3F	077	6F	157	0F	17	1F	37	0-7-8	7-8	(12)(12-0)
2,9	@	At Sign	40	100	7C	174	0C	14	3A	72	4-8	4-8	0-2-8
	A		41	101	C1	301	11	21	11	21	12-1	12-1	12-1
	B		42	102	C2	302	12	22	12	22	12-2	12-2	12-2
	C		43	103	C3	303	13	23	13	23	12-3	12-3	12-3
	D		44	104	C4	304	14	24	14	24	12-4	12-4	12-4
	E		45	105	C5	305	15	25	15	25	12-5	12-5	12-5
	F		46	106	C6	306	16	26	16	26	12-6	12-6	12-6
	G		47	107	C7	307	17	27	17	27	12-7	12-7	12-7
	H		48	110	C8	310	18	30	18	30	12-8	12-8	12-8
	I		49	111	C9	311	19	31	19	31	12-9	12-9	12-9
	J		4A	112	D1	321	21	41	21	41	11-1	11-1	11-1
	K		4B	113	D2	322	22	42	22	42	11-2	11-2	11-2
	L		4C	114	D3	323	23	43	23	43	11-3	11-3	11-3
	M		4D	115	D4	324	24	44	24	44	11-4	11-4	11-4
	N		4E	116	D5	325	25	45	25	45	11-5	11-5	11-5
	O		4F	117	D6	326	26	46	26	46	11-6	11-6	11-6
	P		50	120	D7	327	27	47	27	47	11-7	11-7	11-7
	Q		51	121	D8	330	28	50	28	50	11-8	11-8	11-8
	R		52	122	D9	331	29	51	29	51	11-9	11-9	11-9
	S		53	123	E2	342	32	62	32	62	0-2	0-2	0-2
	T		54	124	E3	343	33	63	33	63	0-3	0-3	0-3
	U		55	125	E4	344	34	64	34	64	0-4	0-4	0-4
	V		56	126	E5	345	35	65	35	65	0-5	0-5	0-5
	W		57	127	E6	346	36	66	36	66	0-6	0-6	0-6
	X		58	130	E7	347	37	67	37	67	0-7	0-7	0-7
	Y		59	131	E8	350	38	70	38	70	0-8	0-8	0-8
	Z		5A	132	E9	351	39	71	39	71	0-9	0-9	0-9
2	[	Left Bracket	[5B	133	¢ 4A	112	[0A	12	¢ 3F	77	12-2-8	2-8	0-7-8
2	\	Reverse Slant	\ 5C	134	\ E0	340	\ 1F	37	! 2E	56	0-2-8	12-7-8	11-0-8
247	]	Right Bracket	] 5D	135	! 5A	132	] 1C	34	| 2F	57	11-2-8	12-4-8	(11)(11-0)
2	^	Circumflex Accent	^ 5E	136		5F 137	↑ 20	40		1E 36	11-7-8	11-0	12-0-8
	_	Underline	_ 5F	137	_ 6D	155	← 3A	72		3E 76	0-5-8	0-2-8	0-0-8

NOTE	SYMBOL	NAME	ASCII CODE 16	8	EBCDIC CODE 16	8	GBCD CODE 16	8	HBCD CODE 16	8	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
3	`	Grave Accent	60	140	79	171	1F	37	1E	36	1-8	12-7-8	12-8-8
1,8	a		61	141	81	201	11	21	11	21	12-0-1	12-1	12-1
1,8	b		62	142	82	202	12	22	12	22	12-0-2	12-2	12-2
1,8	c		63	143	83	203	13	23	13	23	12-0-3	12-3	12-3
1,8	d		64	144	84	204	14	24	14	24	12-0-4	12-4	12-4
1,8	e		65	145	85	205	15	25	15	25	12-0-5	12-5	12-5
1,8	f		66	146	86	206	16	26	16	26	12-0-6	12-6	12-6
1,8	g		67	147	87	207	17	27	17	27	12-0-7	12-7	12-7
1,8	h		68	150	88	210	18	30	18	30	12-0-8	12-8	12-8
1,8	i		69	151	89	211	19	31	19	31	12-0-9	12-9	12-9
1,8	j		6A	152	91	221	21	41	21	41	12-11-1	11-1	11-1
1,8	k		6B	153	92	222	22	42	22	42	12-11-2	11-2	11-2
1,8	l		6C	154	93	223	23	43	23	43	12-11-3	11-3	11-3
1,8	m		6D	155	94	224	24	44	24	44	12-11-4	11-4	11-4
1,8	n		6E	156	95	225	25	45	25	45	12-11-5	11-5	11-5
1,8	o		6F	157	96	226	26	46	26	46	12-11-6	11-6	11-6
1,8	p		70	160	97	227	27	47	27	47	12-11-7	11-7	11-7
1,8	q		71	161	98	230	28	50	28	50	12-11-8	11-8	11-8
1,8	r		72	162	99	231	29	51	29	51	12-11-9	11-9	11-9
1,8	s		73	163	A2	242	32	62	32	62	11-0-2	0-2	0-2
1,8	t		74	164	A3	243	33	63	33	63	11-0-3	0-3	0-3
1,8	u		75	165	A4	244	34	64	34	64	11-0-4	0-4	0-4
1,8	v		76	166	A5	245	35	65	35	65	11-0-5	0-5	0-5
1,8	w		77	167	A6	246	36	66	36	66	11-0-6	0-6	0-6
1,8	x		78	170	A7	247	37	67	37	67	11-0-7	0-7	0-7
1,8	y		79	171	A8	250	38	70	38	70	11-0-8	0-8	0-8
1,8	z		7A	172	A9	251	39	71	39	71	11-0-9	0-9	0-9
1,2	{	Left Brace	{ 7B	173	{ C0	300	0 00	00	+ 10	20	12-0	0	(12-0)(12)
2,3	;	Broken Vertical Line	7C	174	6A	152	1F	37	1E	36	12-11	12-7-8	12-8-8
127	}	Right Brace	} 7D	175	} D0	320	! 20	40	- 20	40	11-0	11-0	(11-0)(11)
2,3	~	Tilde	7E	176	A1	241	1F	37	1E	36	11-0-1	12-7-8	12-8-8
3	DEL	Delete	7F	177	07	007	1F	37	1E	36	12-7-9		

Notes
1. From EBCDIC or ASCII to HBCD or GBCD this is a one-way correspondence.
2. ISO defines these ASCII codes as variable for national usage.
3. Since there is no corresponding character a default character is substituted here; 36 (octal)(¤) for HBCD and 37 (octal)(\) for GBCD.
4. In HBCD the code 57 (octal) may represent 1/2 or ¦
5. TM occupies the same position as DC3. TM is an EBCDIC control character while DC3 is an ASCII control character.
6. The internal and punched card codes shown for HBCD and GBCD are for capital alphabetics.
7. There are two HBCD card code sets (HBCD1 and HBCD2), the difference being the card punch representation for (+) and (-), and for (↑) and (!). For the HBCD1 set (+) and (-) are represented with punch codes 12-0 and 11-0, while (↑) and (!) are represented by 12 and 11. For the HBCD2 set (+) and (-) are represented with punch codes 12 and 11, while (↑) and (!) are represented by 12-0 and 11-0.
8. These are EBCDIC control characters and are not defined in the ASCII standard.
9. IBM defines these EBCDIC codes as national alphabetic extenders.

UNIFIED CHARACTER SET - EBCDIC SEQUENCE

NOTE	SYMBOL	NAME	EBCDIC CODE 16 8	ASCII CODE 16 8	GBCD CODE 16 8	HBCD CODE 16 8	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
3	NUL	Null	00 000	00 000	1F 37	1E 36	12-0-1-8-9		
3	SOH	Start of Heading	01 001	01 001	1F 37	1E 36	12-1-9		
3	STX	Start of Text	02 002	02 002	1F 37	1E 36	12-2-9		
3	ETX	End of Text	03 003	03 003	1F 37	1E 36	12-3-9		
3,8	PF	Punch Off	04 004	9C 234	1F 37	1E 36	12-4-9		
3	HT	Horizontal Tab	05 005	09 011	1F 37	1E 36	12-5-9		
3,8	LC	Lower Case	06 006	88 206	1F 37	1E 36	12-6-9		
3	DEL	Delete	07 007	7F 177	1F 37	1E 36	12-7-9		
3	GE	Graphic Escape	08 010	97 227	1F 37	1E 36	12-8-9		
3,8	RLF	Reverse Line Feed	09 011	8D 215	1F 37	1E 36	12-1-8-9		
3,8	SMM	Start of Manual Message	0A 012	8E 216	1F 37	1E 36	12-2-8-9		
3	VT	Vertical Tab	0B 013	0B 013	1F 37	1E 36	12-3-8-9		
3	FF	Form Feed	0C 014	0C 014	1F 37	1E 36	12-4-8-9		
3	CR	Carriage Return	0D 015	0D 015	1F 37	1E 36	12-5-8-9		
3	SO	Shift Out	0E 016	0E 016	1F 37	1E 36	12-6-8-9		
3	SI	Shift In	0F 017	0F 017	1F 37	1E 36	12-7-8-9		
3	DLE	Data Link Escape	10 020	10 020	1F 37	1E 36	12-11-1-8-9		
3	DC1	Device Control 1	11 021	11 021	1F 37	1E 36	11-1-9		
3	DC2	Device Control 2	12 022	12 022	1F 37	1E 36	11-2-9		
3,8	TM	Tape Mark	13 023	13 023	1F 37	1E 36	11-3-9		
3,8	RES	Restore	14 024	9D 235	1F 37	1E 36	11-4-9		
3,8	NL	New Line	15 025	85 205	1F 37	1E 36	11-5-9		
3	BS	Backspace	16 026	08 010	1F 37	1E 36	11-6-9		
3,8	IL	Idle	17 027	87 207	1F 37	1E 36	11-7-9		
3	CAN	Cancel	18 030	18 030	1F 37	1E 36	11-8-9		
3	EM	End of Medium	19 031	19 031	1F 37	1E 36	11-1-8-9		
3,8	CC	Cursor Control	1A 032	92 222	1F 37	1E 36	11-2-8-9		
3,8	CU1	Customer Use 1	1B 033	8F 217	1F 37	1E 36	11-3-8-9		
3	IFS	Interchange File Separator	1C 034	1C 034	1F 37	1E 36	11-4-8-9		
3	IGS	Interchange Group Separator	1D 035	1D 035	1F 37	1E 36	11-5-8-9		
3	IRS	Interchange Record Separator	1E 036	1E 036	1F 37	1E 36	11-6-8-9		
3	IUS	Interchange Unit Separator	1F 037	1F 037	1F 37	1E 36	11-7-8-9		
3,8	DS	Digit Select	20 040	80 200	1F 37	1E 36	11-0-1-8-9		
3,8	SOS	Start of Significance	21 041	81 201	1F 37	1E 36	0-1-9		
3,8	FS	Field Separator	22 042	82 202	1F 37	1E 36	0-2-9		
3		UNDEFINED CODES	23 043	83 203	1F 37	1E 36	0-3-9		
3,8	BYP	Bypass	24 044	84 204	1F 37	1E 36	0-4-9		
3	LF	Line Feed	25 045	0A 012	1F 37	1E 36	0-5-9		
3	ETB	End of Transmission Block	26 046	17 027	1F 37	1E 36	0-6-9		
3	ESC	Escape	27 047	1B 033	1F 37	1E 36	0-7-9		
3		UNDEFINED CODES	28 050	88 210	1F 37	1E 36	0-8-9		
3		UNDEFINED CODES	29 051	89 211	1F 37	1E 36	0-1-8-9		
3,8	SM	Set Mode	2A 052	8A 212	1F 37	1E 36	0-2-8-9		
3,8	CU2	Customer Use 2	2B 053	8B 213	1F 37	1E 36	0-3-8-9		
3		UNDEFINED CODES	2C 054	8C 214	1F 37	1E 36	0-4-8-9		
3	ENQ	Enquiry	2D 055	05 005	1F 37	1E 36	0-5-8-9		
3	ACK	Acknowledge	2E 056	06 006	1F 37	1E 36	0-6-8-9		
3	BEL	Bell	2F 057	07 007	1F 37	1E 36	0-7-8-9		

NOTE	SYMBOL	NAME	EBCDIC CODE 16	EBCDIC CODE 8	ASCII CODE 16	ASCII CODE 8	GBCD CODE 16	GBCD CODE 8	HBCD CODE 16	HBCD CODE 8	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
3		UNDEFINED CODES	30	060	90	220	1F	37	1E	36	12-11-0-1-8-9		
3		UNDEFINED CODES	31	061	91	221	1F	37	1E	36	1-9		
3	SYN	Synchronous Idle	32	062	16	026	1F	37	1E	36	2-9		
3		UNDEFINED CODES	33	063	93	223	1F	37	1E	36	3-9		
3,8	PN	Punch On	34	064	94	224	1F	37	1E	36	4-9		
3,8	RS	Reader Stop	35	065	95	225	1F	37	1E	36	5-9		
3,8	UC	Upper Case	36	066	96	226	1F	37	1E	36	6-9		
3	EOT	End of Transmission	37	067	04	004	1F	37	1E	36	7-9		
3		UNDEFINED CODES	38	070	98	230	1F	37	1E	36	8-9		
3		UNDEFINED CODES	39	071	99	231	1F	37	1E	36	1-8-9		
3		UNDEFINED CODES	3A	072	9A	232	1F	37	1E	36	2-8-9		
3,8	CU3	Customer Use 3	3B	073	9B	233	1F	37	1E	36	3-8-9		
3	DC4	Device Control 4	3C	074	14	024	1F	37	1E	36	4-8-9		
3	NAK	Negative Acknowledge	3D	075	15	025	1F	37	1E	36	5-8-9		
3		UNDEFINED CODES	3E	076	9E	236	1F	37	1E	36	6-8-9		
3	SUB	Substitute	3F	077	1A	032	1F	37	1E	36	7-8-9		
		Space, Blank	40	100	20	040	10	20	0D	15	Blank	Blank	Blank
3		UNDEFINED CODES	41	101	A0	240	1F	37	1E	36	12-0-1-9		
3		UNDEFINED CODES	42	102	A1	241	1F	37	1E	36	12-0-2-9		
3		UNDEFINED CODES	43	103	A2	242	1F	37	1E	36	12-0-3-9		
3		UNDEFINED CODES	44	104	A3	243	1F	37	1E	36	12-0-4-9		
3		UNDEFINED CODES	45	105	A4	244	1F	37	1E	36	12-0-5-9		
3		UNDEFINED CODES	46	106	A5	245	1F	37	1E	36	12-0-6-9		
3		UNDEFINED CODES	47	107	A6	246	1F	37	1E	36	12-0-7-9		
3		UNDEFINED CODES	48	110	A7	247	1F	37	1E	36	12-0-8-9		
3		UNDEFINED CODES	49	111	A8	250	1F	37	1E	36	12-1-9		
9	¢	Cents Sign	¢ 4A	112	[5B	133	[0A	12	¢ 3F	77	12-2-8	2-8	0-7-8
	.	Period, Decimal Point	4B	113	2E	056	1B	33	1B	33	12-3-8	12-3-8	12-3-8
	<	Less Than	4C	114	3C	074	1E	36	30	60	12-4-8	12-8-8	8-8
	(	Left Parenthesis	4D	115	28	050	1D	35	3C	74	12-5-8	12-5-8	0-4-8
7	+	Plus Sign	4E	116	2B	053	30	60	10	20	12-6-8	12-0	(12-0)(12)
	!	Logical OR	! 4F	117	! 21	041	C! 3F	77	R 3D	75	12-7-8	0-7-8	0-8-8
	&	Ampersand	50	120	26	046	1A	32	0F	17	12	12	7-8
3		UNDEFINED CODES	51	121	A9	251	1F	37	1E	36	12-11-1-9		
3		UNDEFINED CODES	52	122	AA	252	1F	37	1E	36	12-11-2-9		
3		UNDEFINED CODES	53	123	AB	253	1F	37	1E	36	12-11-3-9		
3		UNDEFINED CODES	54	124	AC	254	1F	37	1E	36	12-11-4-9		
3		UNDEFINED CODES	55	125	AD	255	1F	37	1E	36	12-11-5-9		
3		UNDEFINED CODES	56	126	AE	256	1F	37	1E	36	12-11-6-9		
3		UNDEFINED CODES	57	127	AF	257	1F	37	1E	36	12-11-7-9		
3		UNDEFINED CODES	58	130	B0	260	1F	37	1E	36	12-11-8-9		
3		UNDEFINED CODES	59	131	B1	261	1F	37	1E	36	11-1-9		
4?9	!	Exclamation Point EBCDIC	! 5A	132	] 5D	135	] 1C	34	! 2F	57	11-2-8	12-4-8	(11)(11-0)
2,9	$	Dollar Sign	5B	133	24	044	2B	53	2B	53	11-3-8	11-3-8	11-3-8
	*	Asterisk	5C	134	2A	052	2C	54	2C	54	11-4-8	11-4-8	11-4-8
	)	Right Parenthesis	5D	135	29	051	2D	55	1C	34	11-5-8	11-5-8	12-4-8
	;	Semi-Colon	5E	136	3B	073	2E	56	1A	32	11-6-8	11-6-8	12-2-8
		Logical Not	5F	137	5E	136	↑ 20	40	1E	36	11-7-8	11-0	12-8-8

NOTE	SYMBOL	NAME	EBCDIC CODE 16	EBCDIC CODE 8	ASCII CODE 16	ASCII CODE 8	GBCD CODE 16	GBCD CODE 8	HBCD CODE 16	HBCD CODE 8	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
7	-	Minus Sign, Hyphen	60	140	2D	055	2A	52	20	40	11	11	(11-0)(11)
	/	Slash	61	141	2F	057	31	61	31	61	0-1	0-1	0-1
3		UNDEFINED CODES	62	142	82	262	1F	37	1E	36	11-0-2-8		
3		UNDEFINED CODES	63	143	83	263	1F	37	1E	36	11-0-3-8		
3		UNDEFINED CODES	64	144	84	264	1F	37	1E	36	11-0-4-8		
3		UNDEFINED CODES	65	145	85	265	1F	37	1E	36	11-0-5-8		
3		UNDEFINED CODES	66	146	86	266	1F	37	1E	36	11-0-6-8		
3		UNDEFINED CODES	67	147	87	267	1F	37	1E	36	11-0-7-8		
3		UNDEFINED CODES	68	150	88	270	1F	37	1E	36	11-0-8-8		
3		UNDEFINED CODES	69	151	89	271	1F	37	1E	36	0-1-8		
3		Vertical Line	6A	152	7C	174	1F	37	1E	36	12-11	12-7-8	12-8-8
	,	Comma	6B	153	2C	054	3B	73	3B	73	0-3-8	0-3-8	0-3-8
	%	Percent Sign	6C	154	25	045	3C	74	1D	35	0-4-8	0-4-8	12-5-8
	_	Underscore	6D	155	5F	137	3A	72	3E	76	0-5-8	0-2-8	0-8-8
	>	Greater Than Sign	6E	156	3E	076	0E	16	0E	16	0-6-8	6-8	6-8
7	?	Question Mark	6F	157	3F	077	0F	17	1F	37	0-7-8	7-8	(12)(12-0)
3		UNDEFINED CODES	70	160	BA	272	1F	37	1E	36	12-11-0		
3		UNDEFINED CODES	71	161	BB	273	1F	37	1E	36	12-11-0-1-8		
3		UNDEFINED CODES	72	162	BC	274	1F	37	1E	36	12-11-0-2-8		
3		UNDEFINED CODES	73	163	BD	275	1F	37	1E	36	12-11-0-3-8		
3		UNDEFINED CODES	74	164	BE	276	1F	37	1E	36	12-11-0-4-8		
3		UNDEFINED CODES	75	165	BF	277	1F	37	1E	36	12-11-0-5-8		
3		UNDEFINED CODES	76	166	C0	300	1F	37	1E	36	12-11-0-6-8		
3		UNDEFINED CODES	77	167	C1	301	1F	37	1E	36	12-11-0-7-8		
3		UNDEFINED CODES	78	170	C2	302	1F	37	1E	36	12-11-0-8-8		
3	`	Greve Accent	79	171	60	140	1F	37	1E	36	1-8	12-7-8	12-8-8
	:	Colon	7A	172	3A	072	0D	15	0C	14	2-8	5-8	4-8
2,8	#	Number Sign	7B	173	23	043	0B	13	2A	52	3-8	3-8	11-2-8
2,8	@	At Sign	7C	174	40	100	0C	14	3A	72	4-8	4-8	0-2-8
	'	Prime, Apostrophe	7D	175	27	047	2F	57	0A	12	5-8	11-7-8	2-8
	=	Equal Sign	7E	176	3D	075	3D	75	0B	13	6-8	0-5-8	3-8
8	"	Quotation Marks	7F	177	22	042	3E	76	2D	55	7-8	0-8-8	11-5-8
3		UNDEFINED CODES	80	200	C3	303	1F	37	1E	36	12-0-1-8		
1,8	a		81	201	61	141	11	21	11	21	12-0-1	12-1	12-1
1,8	b		82	202	62	142	12	22	12	22	12-0-2	12-2	12-2
1,8	c		83	203	63	143	13	23	13	23	12-0-3	12-3	12-3
1,8	d		84	204	64	144	14	24	14	24	12-0-4	12-4	12-4
1,8	e		85	205	65	145	15	25	15	25	12-0-5	12-5	12-5
1,8	f		86	206	66	146	16	26	16	26	12-0-6	12-6	12-6
1,8	g		87	207	67	147	17	27	17	27	12-0-7	12-7	12-7
1,8	h		88	210	68	150	18	30	18	30	12-0-8	12-8	12-8
1,8	i		89	211	69	151	19	31	19	31	12-0-9	12-9	12-9
3		UNDEFINED CODES	8A	212	C4	304	1F	37	1E	36	12-0-2-8		
3		UNDEFINED CODES	8B	213	C5	305	1F	37	1E	36	12-0-3-8		
3		UNDEFINED CODES	8C	214	C6	306	1F	37	1E	36	12-0-4-8		
3		UNDEFINED CODES	8D	215	C7	307	1F	37	1E	36	12-0-5-8		
3		UNDEFINED CODES	8E	216	C8	310	1F	37	1E	36	12-0-6-8		
3		UNDEFINED CODES	8F	217	C9	311	1F	37	1E	36	12-0-7-8		

NOTE	SYMBOL	NAME	EBCDIC CODE 16 8	ASCII CODE 16 8	GBCD CODE 16 8	HBCD CODE 16 8	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
3		UNDEFINED CODES	90 220	CA 312	1F 37	1E 36	12-11-1-8		
1,8	j		91 221	8A 152	21 41	21 41	12-11-1	11-1	11-1
1,8	k		92 222	8B 153	22 42	22 42	12-11-2	11-2	11-2
1,8	l		93 223	8C 154	23 43	23 43	12-11-3	11-3	11-3
1,8	m		94 224	8D 155	24 44	24 44	12-11-4	11-4	11-4
1,8	n		95 225	8E 156	25 45	25 45	12-11-5	11-5	11-5
1,8	o		96 226	8F 157	26 46	26 46	12-11-6	11-6	11-6
1,8	p		97 227	70 160	27 47	27 47	12-11-7	11-7	11-7
1,8	q		98 230	71 161	28 50	28 50	12-11-8	11-8	11-8
1,8	r		99 231	72 162	29 51	29 51	12-11-9	11-9	11-9
3		UNDEFINED CODES	9A 232	CB 313	1F 37	1E 36	12-11-2-8		
3		UNDEFINED CODES	9B 233	CC 314	1F 37	1E 36	12-11-3-8		
3		UNDEFINED CODES	9C 234	CD 315	1F 37	1E 36	12-11-4-8		
3		UNDEFINED CODES	9D 235	CE 316	1F 37	1E 36	12-11-5-8		
3		UNDEFINED CODES	9E 236	CF 317	1F 37	1E 36	12-11-6-8		
3		UNDEFINED CODES	9F 237	D0 320	1F 37	1E 36	12-11-7-8		
3		UNDEFINED CODES	A0 240	D1 321	1F 37	1E 36	11-0-1-8		
3	~	Tilde	A1 241	7E 176	1F 37	1E 36	11-0-1	12-7-8	12-8-8
1,8	s		A2 242	73 163	32 62	32 62	11-0-2	0-2	0-2
1,8	t		A3 243	74 164	33 63	33 63	11-0-3	0-3	0-3
1,8	u		A4 244	75 165	34 64	34 64	11-0-4	0-4	0-4
1,8	v		A5 245	76 166	35 65	35 65	11-0-5	0-5	0-5
1,8	w		A6 246	77 167	36 66	36 66	11-0-6	0-6	0-6
1,8	x		A7 247	78 170	37 67	37 67	11-0-7	0-7	0-7
1,8	y		A8 250	79 171	38 70	38 70	11-0-8	0-8	0-8
1,8	z		A9 251	7A 172	39 71	39 71	11-0-9	0-9	0-9
3		UNDEFINED CODES	AA 252	D2 322	1F 37	1E 36	11-0-2-8		
3		UNDEFINED CODES	AB 253	D3 323	1F 37	1E 36	11-0-3-8		
3		UNDEFINED CODES	AC 254	D4 324	1F 37	1E 36	11-0-4-8		
3		UNDEFINED CODES	AD 255	D5 325	1F 37	1E 36	11-0-5-8		
3		UNDEFINED CODES	AE 256	D6 326	1F 37	1E 36	11-0-6-8		
3		UNDEFINED CODES	AF 257	D7 327	1F 37	1E 36	11-0-7-8		
3		UNDEFINED CODES	B0 260	D8 330	1F 37	1E 36	12-11-0-1-8		
3		UNDEFINED CODES	B1 261	D9 331	1F 37	1E 36	12-11-0-1		
3		UNDEFINED CODES	B2 262	DA 332	1F 37	1E 36	12-11-0-2		
3		UNDEFINED CODES	B3 263	DB 333	1F 37	1E 36	12-11-0-3		
3		UNDEFINED CODES	B4 264	DC 334	1F 37	1E 36	12-11-0-4		
3		UNDEFINED CODES	B5 265	DD 335	1F 37	1E 36	12-11-0-5		
3		UNDEFINED CODES	B6 266	DE 336	1F 37	1E 36	12-11-0-6		
3		UNDEFINED CODES	B7 267	DF 337	1F 37	1E 36	12-11-0-7		
3		UNDEFINED CODES	B8 270	E0 340	1F 37	1E 36	12-11-0-8		
3		UNDEFINED CODES	B9 271	E1 341	1F 37	1E 36	12-11-0-9		
3		UNDEFINED CODES	BA 272	E2 342	1F 37	1E 36	12-11-0-2-8		
3		UNDEFINED CODES	BB 273	E3 343	1F 37	1E 36	12-11-0-3-8		
3		UNDEFINED CODES	BC 274	E4 344	1F 37	1E 36	12-11-0-4-8		
3		UNDEFINED CODES	BD 275	E5 345	1F 37	1E 36	12-11-0-5-8		
3		UNDEFINED CODES	BE 276	E6 346	1F 37	1E 36	12-11-0-6-8		
3		UNDEFINED CODES	BF 277	E7 347	1F 37	1E 36	12-11-0-7-8		

NOTE	SYMBOL	NAME	EBCDIC CODE 16 8	ASCII CODE 16 8	GBCD CODE 16 8	HBCD CODE 16 8	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
1	{	Opening Brace	{ C0 300	{ 7B 173	0 00 00	+ 10 20	12-0	0	(12-0)(12)
	A		C1 301	41 101	11 21	11 21	12-1	12-1	12-1
	B		C2 302	42 102	12 22	12 22	12-2	12-2	12-2
	C		C3 303	43 103	13 23	13 23	12-3	12-3	12-3
	D		C4 304	44 104	14 24	14 24	12-4	12-4	12-4
	E		C5 305	45 105	15 25	15 25	12-5	12-5	12-5
	F		C6 306	46 106	16 26	16 26	12-6	12-6	12-6
	G		C7 307	47 107	17 27	17 27	12-7	12-7	12-7
	H		C8 310	48 110	18 30	18 30	12-8	12-8	12-8
	I		C9 311	49 111	19 31	19 31	12-9	12-9	12-9
3		UNDEFINED CODES	CA 312	E8 350	1F 37	1E 36	12-0-2-8-9		
3		UNDEFINED CODES	CB 313	E9 351	1F 37	1E 36	12-0-3-8-9		
3		UNDEFINED CODES	CC 314	EA 352	1F 37	1E 36	12-0-4-8-9		
3		UNDEFINED CODES	CD 315	EB 353	1F 37	1E 36	12-0-5-8-9		
3		UNDEFINED CODES	CE 316	EC 354	1F 37	1E 36	12-0-6-8-9		
3		UNDEFINED CODES	CF 317	ED 355	1F 37	1E 36	12-0-7-8-9		
1,7	}	Closing Brace	} D0 320	} 7D 175	! 20 40	- 20 40	11-0	11-0	(11-0)(11)
	J		D1 321	4A 112	21 41	21 41	11-1	11-1	11-1
	K		D2 322	4B 113	22 42	22 42	11-2	11-2	11-2
	L		D3 323	4C 114	23 43	23 43	11-3	11-3	11-3
	M		D4 324	4D 115	24 44	24 44	11-4	11-4	11-4
	N		D5 325	4E 116	25 45	25 45	11-5	11-5	11-5
	O		D6 326	4F 117	26 46	26 46	11-6	11-6	11-6
	P		D7 327	50 120	27 47	27 47	11-7	11-7	11-7
	Q		D8 330	51 121	28 50	28 50	11-8	11-8	11-8
	R		D9 331	52 122	29 51	29 51	11-9	11-9	11-9
3		UNDEFINED CODES	DA 332	EE 356	1F 37	1E 36	12-11-2-8-9		
3		UNDEFINED CODES	DB 333	EF 357	1F 37	1E 36	12-11-3-8-9		
3		UNDEFINED CODES	DC 334	F0 360	1F 37	1E 36	12-11-4-8-9		
3		UNDEFINED CODES	DD 335	F1 361	1F 37	1E 36	12-11-5-8-9		
3		UNDEFINED CODES	DE 336	F2 362	1F 37	1E 36	12-11-6-8-9		
3		UNDEFINED CODES	DF 337	F3 363	1F 37	1E 36	12-11-7-8-9		
	\	Reverse Slant	\ E0 340	\ 5C 134	\ 1F 37	# 2E 56	0-2-8	12-7-8	11-8-8
3		UNDEFINED CODES	E1 341	9F 237	1F 37	1E 36	11-0-1-9		
	S		E2 342	53 123	32 62	32 62	0-2	0-2	0-2
	T		E3 343	54 124	33 63	33 63	0-3	0-3	0-3
	U		E4 344	55 125	34 64	34 64	0-4	0-4	0-4
	V		E5 345	56 126	35 65	35 65	0-5	0-5	0-5
	W		E6 346	57 127	36 66	36 66	0-6	0-6	0-6
	X		E7 347	58 130	37 67	37 67	0-7	0-7	0-7
	Y		E8 350	59 131	38 70	38 70	0-8	0-8	0-8
	Z		E9 351	5A 132	39 71	39 71	0-9	0-9	0-9
3		UNDEFINED CODES	EA 352	F4 364	1F 37	1E 36	11-0-2-8-9		
3		UNDEFINED CODES	EB 353	F5 365	1F 37	1E 36	11-0-3-8-9		
3		UNDEFINED CODES	EC 354	F6 366	1F 37	1E 36	11-0-4-8-9		
3		UNDEFINED CODES	ED 355	F7 367	1F 37	1E 36	11-0-5-8-9		
3		UNDEFINED CODES	EE 356	F8 370	1F 37	1E 36	11-0-6-8-9		
3		UNDEFINED CODES	EF 357	F9 371	1F 37	1E 36	11-0-7-8-9		

NOTE	SYMBOL	NAME	EBCDIC CODE 16 8	ASCII CODE 16 8	GBCD CODE 16 8	HBCD CODE 16 8	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
	0		F0 360	30 060	00 00	00 00	0	0	0
	1		F1 361	31 061	01 01	01 01	1	1	1
	2		F2 362	32 062	02 02	02 02	2	2	2
	3		F3 363	33 063	03 03	03 03	3	3	3
	4		F4 364	34 064	04 04	04 04	4	4	4
	5		F5 365	35 065	05 05	05 05	5	5	5
	6		F6 366	36 066	06 06	06 06	6	6	6
	7		F7 367	37 067	07 07	07 07	7	7	7
	8		F8 370	38 070	08 10	08 10	8	8	8
	9		F9 371	39 071	09 11	09 11	9	9	9
9		Long Vertical Mark	FA 372	FA 372	1F 37	1E 36	12-11-0-2-8-9		
3		UNDEFINED CODES	FB 373	FB 373	1F 37	1E 36	12-11-0-3-8-9		
3		UNDEFINED CODES	FC 374	FC 374	1F 37	1E 36	12-11-0-4-8-9		
3		UNDEFINED CODES	FD 375	FD 375	1F 37	1E 36	12-11-0-5-8-9		
3		UNDEFINED CODES	FE 376	FE 376	1F 37	1E 36	12-11-0-6-8-9		
3	EO	Eight Ones	FF 377	FF 377	1F 37	1E 36	12-11-0-7-8-9		

Notes
1. From EBCDIC or ASCII to HBCD or GBCD this is a one-way correspondence.
2. ISO defines these ASCII codes as variable for national usage.
3. Since there is no corresponding character a default character is substituted here; 36 (octal)(#) for HBCD and 37 (octal)(\) for GBCD.
4. In HBCD the code 57 (octal) may represent 1/2 or ¦
5. TM occupies the same position as DC3. TM is an EBCDIC control character while DC3 is an ASCII control character.
6. The internal and punched card codes shown for HBCD and GBCD are for capital alphabetics.
7. There are two HBCD card code sets (HBCD1 and HBCD2), the difference being the card punch representation for (+) and (-), and for (?) and (!). For the HBCD1 set (+) and (-) are represented with punch codes 12-0 and 11-0, while (?) and (!) are represented by 12 and 11. For the HBCD2 set (+) and (-) are represented with punch codes 12 and 11, while (?) and (!) are represented by 12-0 and 11-0.
8. These are EBCDIC control characters and are not defined in the ASCII standard.
9. IBM defines these EBCDIC codes as national alphabetic extenders.

UNIFIED CHARACTER SET - GBCD SEQUENCE

NOTE	SYMBOL	NAME	GBCD CODE 16 8	HBCD CODE 16 8	ASCII CODE 16 8	EBCDIC CODE 16 8	GBCD CARD CODE	HBCD CARD CODE	ASCII/EBCDIC CARD CODE
	0		00 00	00 00	30 060	F0 360	0	0	0
	1		01 01	01 01	31 061	F1 361	1	1	1
	2		02 02	02 02	32 062	F2 362	2	2	2
	3		03 03	03 03	33 063	F3 363	3	3	3
	4		04 04	04 04	34 064	F4 364	4	4	4
	5		05 05	05 05	35 065	F5 365	5	5	5
	6		06 06	06 06	36 066	F6 366	6	6	6
	7		07 07	07 07	37 067	F7 367	7	7	7
	8		08 10	08 10	38 070	F8 370	8	8	8
	9		09 11	09 11	39 071	F9 371	9	9	9
	[	Left Bracket	[0A 12	£ 3F 77	[5B 133	£ 4A 112	2-8	0-7-8	12-2-8
	#	Number Sign	0B 13	2A 52	23 043	7B 173	3-8	11-2-8	3-8
	@	At Sign	0C 14	3A 72	40 100	7C 174	4-8	0-2-8	4-8
	:	Colon	0D 15	0C 14	3A 072	7A 172	5-8	4-8	2-8
	>	Greater Than	0E 16	0E 16	3E 076	6E 156	6-8	6-8	0-6-8
7	↑	Question Mark	0F 17	1F 37	3F 077	6F 157	7-8	(12)(12-0)	0-7-8
		Space, Blank	10 20	0D 15	20 040	40 100	Blank	Blank	Blank
	A		11 21	11 21	41 101	C1 301	12-1	12-1	12-1
	B		12 22	12 22	42 102	C2 302	12-2	12-2	12-2
	C		13 23	13 23	43 103	C3 303	12-3	12-3	12-3
	D		14 24	14 24	44 104	C4 304	12-4	12-4	12-4
	E		15 25	15 25	45 105	C5 305	12-5	12-5	12-5
	F		16 26	16 26	46 106	C6 306	12-6	12-6	12-6
	G		17 27	17 27	47 107	C7 307	12-7	12-7	12-7
	H		18 30	18 30	48 110	C8 310	12-8	12-8	12-8
	I		19 31	19 31	49 111	C9 311	12-9	12-9	12-9
	&	Ampersand	1A 32	0F 17	26 046	50 120	12	7-8	12
	.	Period	1B 33	1B 33	2E 056	4B 113	12-3-8	12-3-8	12-3-8
4,7	]	Right Bracket	] 1C 34	! 2F 57	] 5D 135	! 5A 132	12-4-8	(11)(11-0)	11-2-8
	(	Left Parenthesis	1D 35	3C 74	28 050	4D 115	12-5-8	0-4-8	12-5-8
	<	Less Than	1E 36	30 60	3C 074	4C 114	12-6-8	5-8	12-4-8
	\	Reverse Slant	\ 1F 37	∮ 2E 56	\ 5C 134	\ E0 340	12-7-8	11-8-8	0-2-8
	↑	Upward Arrow	↑ 20 40	1E 36	^ 5E 136	5F 137	11-0	12-6-8	11-7-8
	J		21 41	21 41	4A 112	D1 321	11-1	11-1	11-1
	K		22 42	22 42	4B 113	D2 322	11-2	11-2	11-2
	L		23 43	23 43	4C 114	D3 323	11-3	11-3	11-3
	M		24 44	24 44	4D 115	D4 324	11-4	11-4	11-4
	N		25 45	25 45	4E 116	D5 325	11-5	11-5	11-5
	O		26 46	26 46	4F 117	D6 326	11-6	11-6	11-6
	P		27 47	27 47	50 120	D7 327	11-7	11-7	11-7
	Q		28 50	28 50	51 121	D8 330	11-8	11-8	11-8
	R		29 51	29 51	52 122	D9 331	11-9	11-9	11-9
7	-	Hyphen, Minus	2A 52	20 40	2D 055	60 140	11	(11-0)(11)	11
	$	Dollar Sign	2B 53	2B 53	24 044	5B 133	11-3-8	11-3-8	11-3-8
	*	Asterisk	2C 54	2C 54	2A 052	5C 134	11-4-8	11-4-8	11-4-8
	)	Right Parenthesis	2D 55	1C 34	29 051	5D 135	11-5-8	12-4-8	11-5-8
	;	Semi-Colon	2E 56	1A 32	3B 073	5E 136	11-6-8	12-2-8	11-6-8
	'	Apostrophe	2F 57	0A 12	27 047	7D 175	11-7-8	2-8	5-8

NOTE	SYMBOL	NAME	GBCD CODE 16 8	HBCD CODE 16 8	ASCII CODE 16 8	EBCDIC CODE 16 8	GBCD CARD CODE	HBCD CARD CODE	ASCII/EBCDIC CARD CODE
7	+	Plus	30 60	10 20	2B 053	4E 116	12-0	(12-0)(12)	12-6-8
	/	Slash	31 61	31 61	2F 057	61 141	0-1	0-1	0-1
	S		32 62	32 62	53 123	E2 342	0-2	0-2	0-2
	T		33 63	33 63	54 124	E3 343	0-3	0-3	0-3
	U		34 64	34 64	55 125	E4 344	0-4	0-4	0-4
	V		35 65	35 65	56 126	E5 345	0-5	0-5	0-5
	W		36 66	36 66	57 127	E6 346	0-6	0-6	0-6
	X		37 67	37 67	58 130	E7 347	0-7	0-7	0-7
	Y		38 70	38 70	59 131	E8 350	0-8	0-8	0-8
	Z		39 71	39 71	5A 132	E9 351	0-9	0-9	0-9
	←	Left Arrow	← 3A 72	3E 76	_ 5F 137	_ 6D 155	0-2-8	0-6-8	0-5-8
	,	Comma	3B 73	3B 73	2C 054	6B 153	0-3-8	0-3-8	0-3-8
	%	Percent Sign	3C 74	1D 35	25 045	6C 154	0-4-8	12-5-8	0-4-8
	=	Equal	3D 75	0B 13	3D 075	7E 176	0-5-8	3-8	6-8
	"	Double Quote	3E 76	2D 55	22 042	7F 177	0-6-8	11-5-8	7-8
	!	Exclamation Point (ASCII)	! 3F 77	R 3D 76	CI 21 041	! 4F 117	0-7-8	0-5-8	12-7-8

Notes
1. From EBCDIC or ASCII to HBCD or GBCD this is a one-way correspondence.
2. ISO defines these ASCII codes as variable for national usage.
3. Since there is no corresponding character a default character is substituted here; 38 (octal)(#) for HBCD and 37 (octal)(\) for GBCD.
4. In HBCD the code 57 (octal) may represent 1/2 or ¬
5. TM occupies the same position as DC3. TM is an EBCDIC control character while DC3 is an ASCII control character.
6. The internal and punched card codes shown for HBCD and GBCD are for capital alphabetics.
7. There are two HBCD card code sets (HBCD1 and HBCD2), the difference being the card punch representation for (+) and (-), and for (?) and (!). For the HBCD1 set (+) and (-) are represented with punch codes 12-0 and 11-0, while (?) and (!) are represented by 12 and 11. For the HBCD2 set (+) and (-) are represented with punch codes 12 and 11, while (?) and (!) are represented by 12-0 and 11-0.
8. These are EBCDIC control characters and are not defined in the ASCII standard.
9. IBM defines these EBCDIC codes as national alphabetic extenders.

UNIFIED CHARACTER SET - HBCD SEQUENCE

NOTE	SYMBOL	NAME	HBCD CODE 16	HBCD CODE 8	GBCD CODE 16	GBCD CODE 8	ASCII CODE 16	ASCII CODE 8	EBCDIC CODE 16	EBCDIC CODE 8	HBCD CARD CODE	GBCD CARD CODE	ASCII/EBCDIC CARD CODE
	0		00	00	00	00	30	060	F0	360	0	0	0
	1		01	01	01	01	31	061	F1	361	1	1	1
	2		02	02	02	02	32	062	F2	362	2	2	2
	3		03	03	03	03	33	063	F3	363	3	3	3
	4		04	04	04	04	34	064	F4	364	4	4	4
	5		05	05	05	05	35	065	F5	365	5	5	5
	6		06	06	06	06	36	066	F6	366	6	6	6
	7		07	07	07	07	37	067	F7	367	7	7	7
	8		08	10	08	10	38	070	F8	370	8	8	8
	9		09	11	09	11	39	071	F9	371	9	9	9
	'	Apostrophe	0A	12	2F	57	27	047	7D	175	2-8	11-7-8	5-8
	=	Equal	0B	13	3D	75	3D	075	7E	176	3-8	0-5-8	6-8
	:	Colon	0C	14	0D	15	3A	072	7A	172	4-8	5-8	2-8
		Space, Blank	0D	15	10	20	20	040	40	100	Blank	Blank	Blank
	>	Greater Than	0E	16	0E	16	3E	076	6E	156	6-8	8-8	0-6-8
	&	Ampersand	0F	17	1A	32	26	046	50	120	7-8	12	12
7	+	Plus	10	20	30	60	2B	053	4E	116	(12-0)(12)	12-0	12-6-8
	A		11	21	11	21	41	101	C1	301	12-1	12-1	12-1
	B		12	22	12	22	42	102	C2	302	12-2	12-2	12-2
	C		13	23	13	23	43	103	C3	303	12-3	12-3	12-3
	D		14	24	14	24	44	104	C4	304	12-4	12-4	12-4
	E		15	25	15	25	45	105	C5	305	12-5	12-5	12-5
	F		16	26	16	26	46	106	C6	306	12-6	12-6	12-6
	G		17	27	17	27	47	107	C7	307	12-7	12-7	12-7
	H		18	30	18	30	48	110	C8	310	12-8	12-8	12-8
	I		19	31	19	31	49	111	C9	311	12-9	12-9	12-9
	;	Semi-Colon	1A	32	2E	56	3B	073	5E	136	12-2-8	11-6-8	11-6-8
	.	Period	1B	33	1B	33	2E	056	4B	113	12-3-8	12-3-8	12-3-8
	)	Right Parenthesis	1C	34	2D	55	29	051	5D	135	12-4-8	11-5-8	11-5-8
	%	Percent Sign	1D	35	3C	74	25	045	6C	154	12-5-8	0-4-8	0-4-8
	■	Closed Box	1E	36	↑	20	40	•	5E	136	6F	137	12-6-8	11-0	11-7-8
7	?	Question Mark	1F	37		0F	17		3F	077	6F	157	(12)(12-0)	7-8	0-7-8
7	-	Hyphen, Minus	20	40		2A	52		2D	055	60	140	(11-0)(11)	11	11
	J		21	41		21	41		4A	112	D1	321	11-1	11-1	11-1
	K		22	42		22	42		4B	113	D2	322	11-2	11-2	11-2
	L		23	43		23	43		4C	114	D3	323	11-3	11-3	11-3
	M		24	44		24	44		4D	115	D4	324	11-4	11-4	11-4
	N		25	45		25	45		4E	116	D5	325	11-5	11-5	11-5
	O		26	46		26	46		4F	117	D6	326	11-6	11-6	11-6
	P		27	47		27	47		50	120	D7	327	11-7	11-7	11-7
	Q		28	50		28	50		51	121	D8	330	11-8	11-8	11-8
	R		29	51		29	51		52	122	D9	331	11-9	11-9	11-9
	#	Number Sign	2A	52		0B	13		23	043	7B	173	11-2-8	3-8	3-8
	$	Dollar Sign	2B	53		2B	53		24	044	5B	133	11-3-8	11-3-8	11-3-8
	*	Asterisk	2C	54		2C	54		2A	052	5C	134	11-4-8	11-4-8	11-4-8
	"	Double Quote	2D	55		3E	76		22	042	7F	177	11-5-8	0-6-8	7-8
	≠	Not Equal	≠	2E	56	\	1F	37	\	5C	134	\	E0	340	11-6-8	12-7-8	0-2-8
4,7	!	Exclamation Point (EBCDIC)	!	2F	57	]	1C	34	]	5D	135	!	5A	132	(11)(11-0)	12-4-8	11-2-8

NOTE	NAME	HBCD CODE 16 8	GBCD CODE 16 8	ASCII CODE 16 8	EBCDIC CODE 16 8	HBCD CARD CODE	GBCD CARD CODE	ASCII/EBCDIC CARD CODE
<	Less Than	30 60	1E 36	3C 074	4C 114	8-6	12-6-8	12-4-8
/	Slash	31 61	31 61	2F 057	61 141	0-1	0-1	0-1
S		32 62	32 62	53 123	E2 342	0-2	0-2	0-2
T		33 63	33 63	54 124	E3 343	0-3	0-3	0-3
U		34 64	34 64	55 125	E4 344	0-4	0-4	0-4
V		35 65	35 65	56 126	E5 345	0-5	0-5	0-5
W		36 66	36 66	57 127	E6 346	0-6	0-6	0-6
X		37 67	37 67	58 130	E7 347	0-7	0-7	0-7
Y		38 70	38 70	59 131	E8 350	0-8	0-8	0-8
Z		39 71	39 71	5A 132	E9 351	0-9	0-9	0-9
@	At Sign	3A 72	0C 14	40 100	7C 174	0-2-8	4-8	4-8
,	Comma	3B 73	3B 73	2C 054	6B 153	0-3-8	0-3-8	0-3-8
(	Left Parenthesis	3C 74	1D 35	28 050	4D 115	0-4-8	12-5-8	12-5-8
CR	Credit Sign	R 3D 75	] 3F 77	CR 21 041	! 4F 117	0-5-8	0-7-8	12-7-8
	Open Box	3E 76	← 3A 72	_ 5F 137	_ 6D 155	0-6-8	0-2-8	0-5-8
¢	Cents Sign	¢ 3F 77	[0A 12	[5B 133	¢ 4A 112	0-7-8	2-8	12-2-8

Notes
1. From EBCDIC or ASCII to HBCD or GBCD this is a one-way correspondence.
2. ISO defines these ASCII codes as variable for national usage.
3. Since there is no corresponding character a default character is substituted here; 38 (octal)(#) for HBCD and 37 (octal)(\) for GBCD.
4. In HBCD the code 57 (octal) may represent 1/2 or |
5. TM occupies the same position as DC3. TM is an EBCDIC control character while DC3 is an ASCII control character.
6. The internal and punched card codes shown for HBCD and GBCD are for capital alphabetics.
7. There are two HBCD card code sets (HBCD1 and HBCD2), the difference being the card punch representation for (+) and (-), and for (↑) and (!). For the HBCD1 set (+) and (-) are represented with punch codes 12-0 and 11-0, while (↑) and (!) are represented by 12 and 11. For the HBCD2 set (+) and (-) are represented with punch codes 12 and 11, while (↑) and (!) are represented by 12-0 and 11-0.
8. These are EBCDIC control characters and are not defined in the ASCII standard.
9. IBM defines these EBCDIC codes as national alphabetic extenders.

INDEX


Bull