HONEYWELL

DPS 8 & DPS 88 ASSEMBLY INSTRUCTIONS

SOFTWARE

DPS 8 & DPS 88

ASSEMBLY INSTRUCTIONS

SUBJECT

Description of the Assembly Instructions for the CP-6 and DPS 8/DPS 88 Information Systems

SPECIAL INSTRUCTIONS

This is the first revision to DH03-00, dated April 1980. Because of extensive changes, change bars have not been used.

SOFTWARE SUPPORTED

CP-6, Software Release B03
GCOS 8, Software Release 2300

ORDER NUMBER

DH03-01

June 1984

Honeywell

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 twos complement number system as employed in a sign-number machine (see Appendix F).

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.

Related manuals:

GCOS 8 OS GMAP User's Guide, Order Number DH01.

The information and specifications in this document are subject to change without notice. Consult your Honeywell Marketing Representative for product or service availability.

File No.: 1Q23, 1V23, 1W23

LISTING OF MANUALS

A listing of large system software manuals is available to any Honeywell user who has access to an uppercase and lowercase ASCII terminal with a line length of 80 or more characters. The manuals are categorized both by software release and by software category. This listing is updated regularly to enable ordering of manuals as soon as they are published. Instructions on how to order manuals are output with the listing.

To obtain the listing:

1. Dial appropriate telephone number to connect your terminal with the Multics system in Phoenix.

1200-baud terminals	300-baud terminals	150-baud terminals
(602)249-5356	249-7501 249-7801	249-7554
249-6430	249-7701	

System response - computer system identification

```
Example:  Multics MR10.1:  Honeywell LCPD Phoenix, System M
          Load=35.0 out of 125.0 units:  users=35
```

2. Enter the following login command: `login Sam` (the identifier "Sam" must be used - it is not a sample for any proper name)

Press carriage return key.

System response - request for password

```
Example:  Password:
          ████████████████████ (password strikeover mask)
```

3. Enter the password: `Multics`

Press carriage return key.

System response - Welcome message followed by a query and "r" message

```
Example:  Welcome to the Multics system
          For services available online type:
               :list
          r1111.7 Thu (ready message)
```

4. To obtain a list of commands, enter: `:list`

Press carriage return Key.

System response - list of commands available for specific topics and "r" message.

5. Enter command selected and press carriage return key.

6. To log off Multics system, enter: `logout`

System response - logout message

```
Example:  Sam SRB logged out 02/04/84 1110.3 mst fri
          CPU usage 3 sec, memory usage 22.4 units
```

CONTENTS

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

ILLUSTRATIONS

CONTENTS (cont)

TABLES

SECTION I

INTRODUCTION

The assembler contains a set of machine instructions used to produce code for the Honeywell 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 ex.cution 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 88: Central Interface 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. Overlapping instruction execution, address preparation, and instruction fetch. While an instruction is being executed, address preparation for the next operand (or the operand following it) or the next instruction pair is taking place.
5. Interleaving direct main memory accesses to interleaved system controller modules.
6. ****DPS8: Intermediate storage of address and control information in high-speed registers addressable by content (associative memory).****
7. Absolute address computation at execution time.
8. Ability to hold recently referenced operands and instructions in a high-speed look-aside memory.

Functional Units

The processor consists of independent units. The decimal unit performs decimal arithmetic and bit-string/character-string operations. The virtual unit is used to derive an absolute memory address from a virtual address. This process, called mapping, uses a page table to translate the virtual address into an absolute address.

Address Modification

The address modification capability enables the user to dynamically develop an address contained in an instruction (or indirect word). Before each main memory access, two major phases of address preparation take place:

1. Address modification by register or indirect word content, if specified by the instruction word or indirect word.
2. Address modification in which a virtual memory address is translated (mapped) into an absolute address for accessing main memory (no user control).

The address modification procedure can go on almost indefinitely (limited by lock-up time) with one type of modification leading to repetitions of the same type or to other types of modification before accessing main memory for an operand.

Faults And Interrupts

The processor detects certain 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 multiplexers and other processors) by setting and answering external interrupts. When a fault or interrupt is recognized, a "trap" results. When the processor responds to a fault or interrupt, control is transferred to an operating system module via an inter-domain transfer using an entry descriptor obtained from a fixed memory location.

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

Vector	Location	
Interrupt	30-31 (octal)	
Fault	32-33 (octal)	
Systems Entry	34-35 (octal)	
Backup Fault	40-41 (octal)	****DPS 88: No backup fault****

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.

Execution Of Interrupts

In a multiprogramming/multiprocessing computer system, both the hardware and software must be freed from the burden of checking other components of the system either for completion of, or requests for, service. To accomplish this, all active modules that have completed assigned tasks, or that require service, generate faults or interrupts to the normal flow of instructions in a processor.

Each system controller (DPS 88: Central Interface Unit) has its program interrupt cells connected in a priority sequence. Any interrupt request generated by an active module will set the particular interrupt cell that the interrupting device has been assigned to use.

Normally, upon the completion of executing each instruction word pair in the processor, a check is made for the presence of an interrupt. If no interrupts are present, or if interrupts have been inhibited, instruction execution continues in the normal sequence. If one or more interrupts are present (and not inhibited), the system controller (DPS 88: Central Interface Unit) reports the identity of the highest priority cell that is set and then resets that interrupt cell. This causes the processor to execute an inward CLIMB. The processor servicing an interrupt may load the interrupt enable registers with suitable combinations of bits to prevent any undesired interrupts and to prevent other processors from being interrupted. Servicing of the interrupt can then proceed without use of the interrupt inhibit bit. The processor can be protected against undesirable interrupts but can be interrupted, in turn, by enabled, higher-priority interrupts.

Each input/output module will generate interrupts to indicate events such as:

1. Successful completion of a requested I/O action
2. Unsuccessful initiation of a requested I/O action
3. Special interrupts (e.g., unit becoming READY)
4. Error conditions

PROCESSOR MODES OF OPERATION

The three processor modes of operation are Privileged Master mode, Master mode, and Slave mode. The determinants involved in defining these processor modes are 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.

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

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

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

[b] When working space zero is referenced, the housekeeping bit is assumed to be ON and the processor addresses real memory directly.

Figure 1-1. Status Of Processor Mode Determinants

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 not marked 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. Refer to the CLIMB instruction definition, documented later in the manual, for a detailed description of the variations of the CLIMB instruction.

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 type T = 0, 2, 4, or 6 segments

3. Executing instructions from housekeeping pages of type T = 0 segments

4. Executing an ICLIMB or GCLIMB instruction or 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. Accessing or executing in working space zero (absolute addressing)

4. Writing on housekeeping pages of type T = 0, 2, 4, or 6 segments, using instructions other than CLIMB, SDRn, STDn

ADDRESSING MODES

Absolute Mode

Virtual memory provides an absolute addressing mode. When the processor utilizes 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**26 bytes (64 mb) ****DPS 88 maximum size is 2**28 bytes (256 mb) ****).

The processor utilizes the absolute addressing mode each time working space number zero is referenced. Any time a working space other than zero (WSN=0) is referenced the processor utilizes the paging mode. For example, assume that the descriptor contained in the instruction segment register (ISR) points to working space register (WSR) 1, which contains zero, that the instruction refers to DR2, which points to WSR 3, and that WSR 3 contains 20. Then, the instructions and operands with ISR modification (bit 29 OFF) would be accessed in the absolute addressing mode, and operands referenced with bit 29 ON and DR2 selected would be accessed in the paging mode from working space 20.

To utilize 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, an attempted reference to WSN 0 results in a Command fault. The housekeeping bit is assumed ON when WSN 0 is referenced.

Paging Mode

The 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 is supported by a page table. The location of the page table supporting a particular working space (WS) is found by using the nine-bit working space (WS) number to index a 512-word table that contains the supporting page table´s absolute memory address. This 512-word table is called the page table directory (PTD). This table is located in memory by a special base register called the page directory base register (PDBR).

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 II

REPRESENTATION OF DATA

BIT GROUPINGS

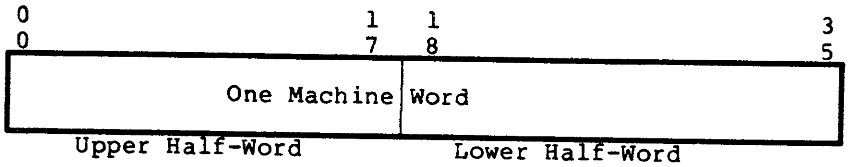
The processor is functionally organized to process 36-bit groupings of information. Special features are also included for ease in manipulating 4-bit groups, 6-bit groups, 9-bit groups, 18-bit groups, and 72-bit double-precision groups. These bit groupings are used by the hardware and software to represent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

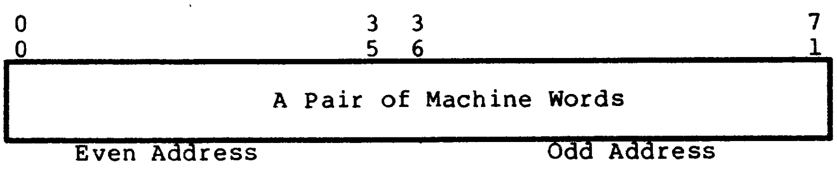
THE MACHINE WORD

The machine word consists of 36 bits arranged as follows:



Data transfers between the processor and memory are double-word oriented; 36 bits are transferred at a time for single-precision data and two parallel 36-bit word transfers occur for double-precision data. When words are transferred to a memory unit, EDAC bits are added to each 36-bit word before storing it. When words are requested from a memory unit, the EDAC bits are read from memory, verified, and removed from the transferred word before sending the word to the processor.

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



In addressing such a pair of memory locations in an instruction that is intended for handling pairs of machine words, either of the two addresses may be used as the effective address (Y). Thus,

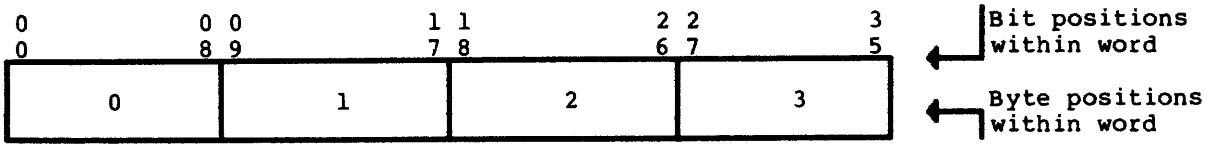
> If Y is even, the pair of locations (Y, Y+1) is accessed. If Y is odd, the pair of locations (Y-1, Y) is accessed. The term "Y-pair" is used for each such pair of addresses. Preferred coding practice refers to the even address; the GMAP assembler issues a warning diagnostic if Y is odd.

CHARACTER-STRINGS

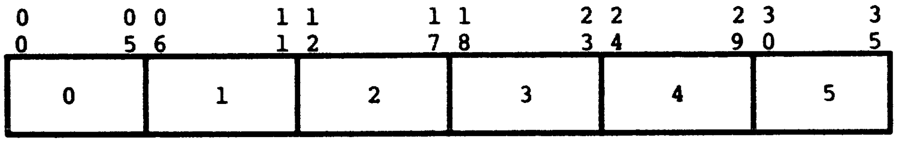
Character Positions

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):

00	01 04	05 08	09	10 13	14 17	18	19 22	23 26	27	28 31	32 35
Z	0	1	Z	2	3	Z	4	5	Z	6	7

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:

0	1	2	3	4-bit character

0	1	2	3	4	5	6-bit character

0	1	2	3	4	5	6	7	8	9-bit character

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

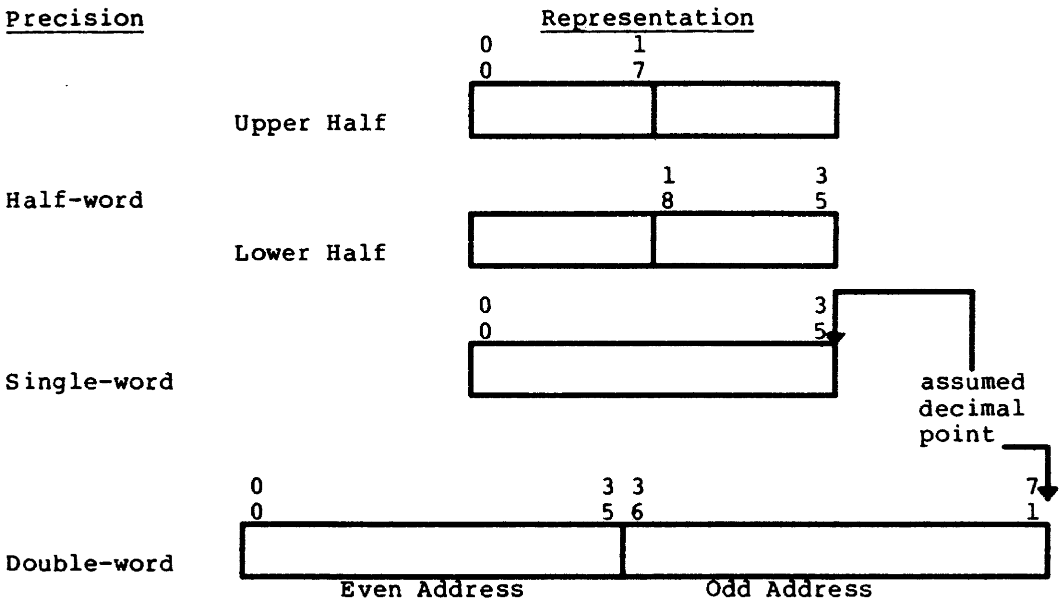
LITERALS

For information on literals refer to the 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 reflected 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 absolute value of the number; when the sign is negative, they represent the twos complement of the absolute value of the number.

In the case of addition and subtraction, the occurrence of an overflow is reflected 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).

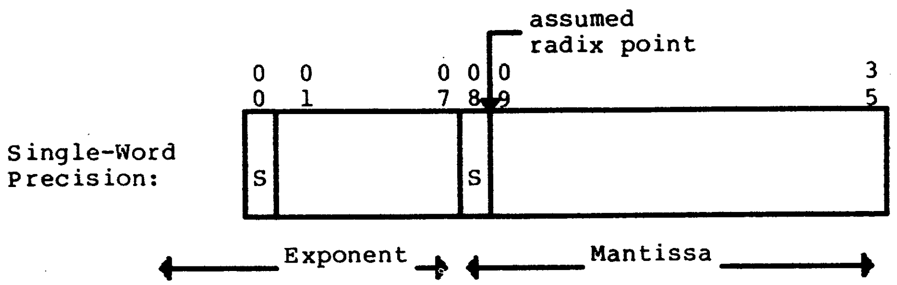
The number ranges for the various cases of precision, interpretation, and arithmetic are given in Table 2-1.

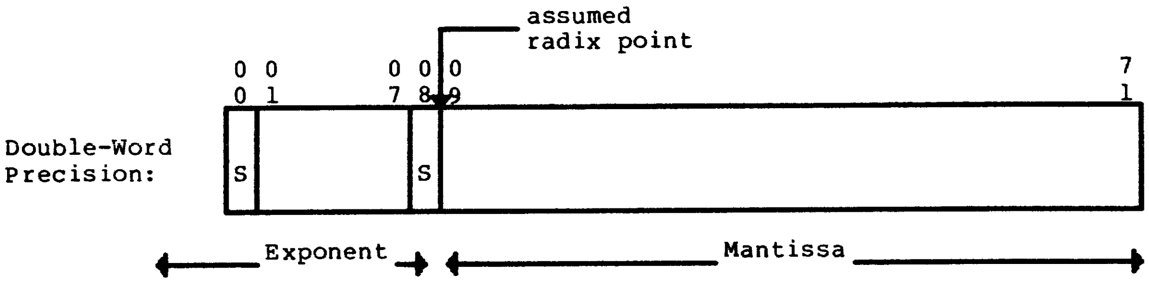
Table 2-1. Ranges Of Fixed-Point Numbers

Inter-pretation	Arithmetic	Precision		
		Half-Word (Xn, $Y_{0...17}$)	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

Binary floating-point numbers are represented with single-word and double-word precision. The upper eight bits represent the integral exponent to the base 2 in twos complement form, and the lower 28 or 64 bits represent the fractional mantissa in twos complement form. The format for a floating-point number is:





where S = sign bit

Before performing 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.

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 (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. Zeros fill in the vacated bit positions.

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	$2^{-129} \le N \le (1-2^{-27})2^{127}$	$2^{-129} \le N \le (1-2^{-63})2^{127}$
	Negative	$(-1+2^{-26})2^{-129} \ge N \ge -2^{127}$	$(-1+2^{-62})2^{-129} \ge N \ge -2^{127}$
Unnormalized	Positive	$2^{-155} \le N \le (1-2^{-27})2^{127}$	$2^{-191} \le N \le (1-2^{-63})2^{127}$
	Negative	$-2^{-155} \ge N \ge -2^{127}$	$-2^{-191} \ge N \ge -2^{127}$

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

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.

When decimal data is declared in source images, the characters "X" or "XD" are specified in the variable field of the DEC pseudo-operation in place of "E" or "D" to indicate single- or double-precision hexadecimal floating-point binary data, respectively. (See the GMAP User's Guide.) These characters control the computation of the exponent, the positioning of the binary mantissa, and the storage required by the data. When reading the converted data, the user should be aware that the exponent represents a power of 16; therefore, a normalized positive mantissa may have as many as three leading binary zeros.

The hexadecimal floating-point mode is enabled only when both bit 32 of the Indicator Register and bit 33 of the Mode Register (DPS 88: bit 0 of the Option Register) are set to 1. The operating system sets the Mode Register (DPS 88: Option Register) via an operating system service request before giving control to a process. After the hexadecimal floating-point mode is requested, the user controls the floating-point mode via the Indicator Register. If the bits are not both 1s, the floating-point mode will be binary.

If a decimal point is present in the variable field of the DEC pseudo-operation and no other controls are defined, the mechanism defaults to floating-point format. The HXFLPT pseudo-operation will alter the default mechanism to hexadecimal floating-point format. The default mechanism may be further controlled by including the ON, OFF, SAVE, or RESTORE options in the variable field of the HXFLPT pseudo-operation. (See the GMAP User's Guide for additional information.)

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

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

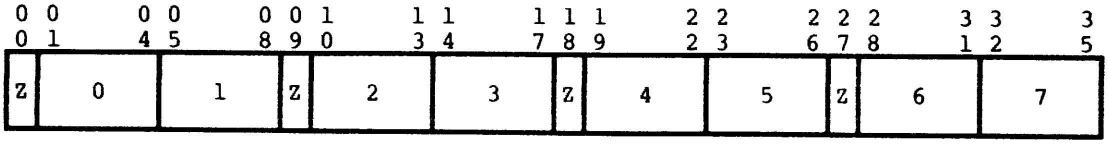
which means that the bit pattern 0011 in the binary expansion keeps repeating indefinitely. If the decimal value 0.2 is converted to a binary expansion of 71 bits and then converted back, the one-digit result would be 0.1, quite different from 0.2. The four-digit result would be 0.1999, which is almost (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. The sure way is to use decimal numbers; alternatively, 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 ten necessary in the course of a computation.

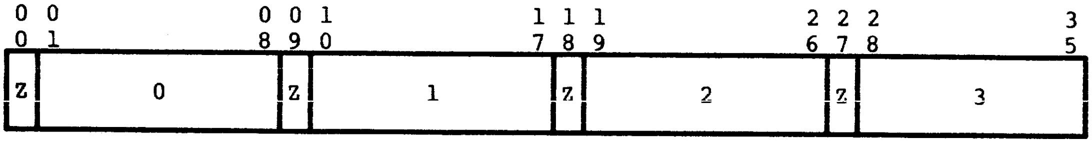
DECIMAL NUMBERS

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 (9-bit)

The 'Z' represents the bit value 0 while other numbers in the fields represent the character 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	
1000	8	
1001	9	
1010	+	found where descriptor specifies digits
1011	+	
1100	+	
1101	-	
1110	+	
1111	+	

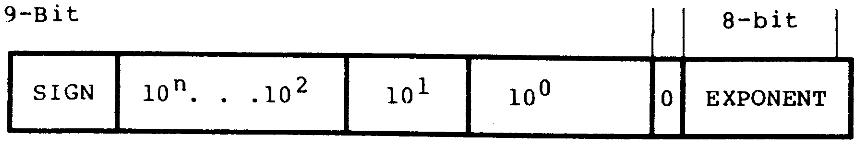
The following codes (9-bit zones are created by prefixing binary 00010) are generated for output signs; the octal values are:

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

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

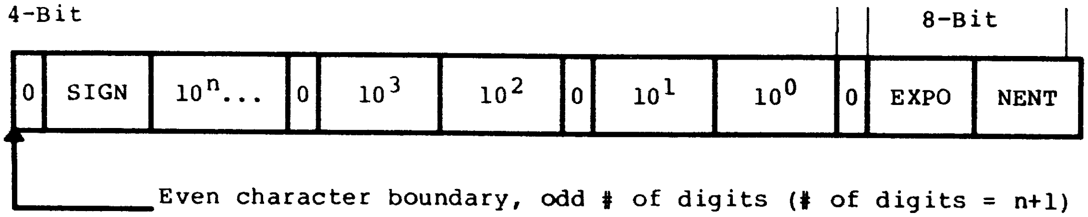
Floating-Point Decimal Numbers

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

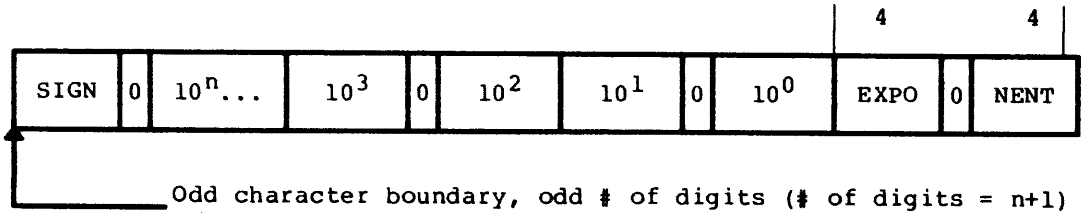


where: SIGN can start at any legal 9-bit character boundary.

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



4-Bit
8-Bit
0
SIGN
10ⁿ...
0
10³
10²
0
10¹
10⁰
0
EXPO
NENT
Even character boundary, odd # of digits (# of digits = n+1)



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 = $0 \ldots 10^{63}$

2. Fixed-point signed integer:

   Range = $\pm 10^{62}$

3. Floating-point (implicitly signed):

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

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

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

SECTION III

MEMORY CHARACTERISTICS

GENERAL DESCRIPTION

Each memory module is composed of a system controller (DPS 88: Central Interface Unit) and associated memory units. Systems are memory-oriented, permitting processor and I/O multiplexer functions to execute asynchronously and simultaneously.

The memory module has neither program execution nor arithmetic capability, but acts as a passive system component. It serves the processor and I/O multiplexer modules that call upon the memory module to save or retrieve information or to communicate with other system components.

In the memory module 36-bit words are paired with EDAC bits to provide error detection and correction. For purposes of memory management, the memory is organized into pages of 1024 words (4096 bytes) each.

Increased system throughput is achieved by operating the memory module 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 address interleaving. Address interleaving considerably reduces the possibility of the same memory unit being accessed in succession. Furthermore, the processor and system controller are especially designed to utilize memory accesses of two memory units in rapid succession. These two factors contribute to higher access rates and more effective memory cycle times.

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. To provide for efficient management and control, the VM space is divided into equal parts called "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 a hardware algorithm, of which memory paging is an integral part.

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; that is, 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. Segment descriptors are passed to a process either by the operating system or by another process (all descriptors are created by the operating system but they may be passed by one process to another process).

In the most secure form of operation, segment descriptors are passed to a process only through one or more of three segment descriptor "stacks" maintained in main memory. Each of these stack areas of memory is defined by a special hardware register. A unique transfer of domain (CLIMB) instruction is provided that allows the process to specify which descriptors in the stacks are to be passed to another process. Then, during the execution of this instruction, the descriptor stack registers are manipulated by the hardware to pass descriptors as specified by the process performing the transfer.

The hardware environment for the virtual memory is composed of four elements: 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 And Pages

The virtual memory is divided into 512 (0 through 511) equal working spaces of 2**34 bytes, each of which is divided into fixed-length parts called pages. These pages are used for memory management and have a fixed size of 1024 words (4096 bytes) each.

Each working space has an associated page table that identifies the real memory allocation. The page table for each working space is located in real memory by a pointer that resides in the page table directory. This directory has 512 entries and the pointer to the directory is stored in the page directory base register (PDBR) that can only be altered in the Privileged Master mode.

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

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. Consequently, a segment may reside on a portion of a page or span several pages (see Figure 3-1).

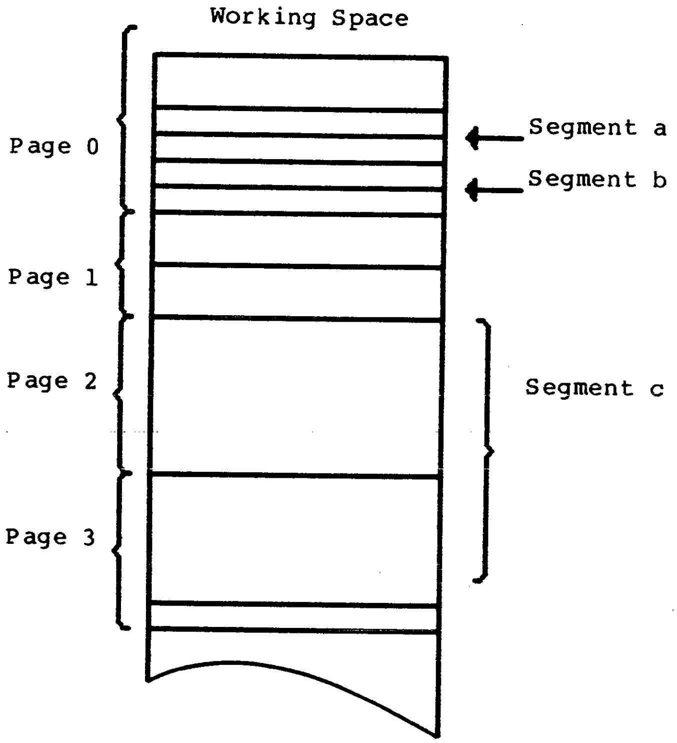
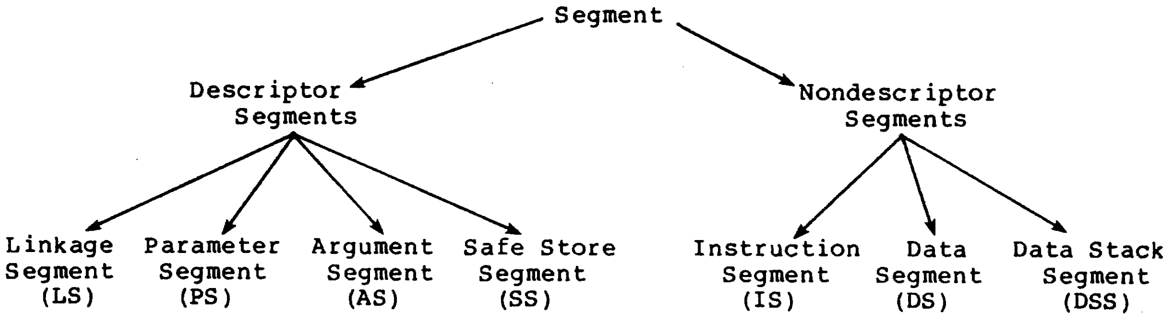


Figure 3-1. Layout Of Segments On Pages

The relationship of a segment and a page is analogous to the relationship of a file and a tape reel. As a multifile reel may contain many files on one reel, a page may contain several segments. As a multireel file has one file that occupies several reels, a segment may extend over several pages.

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 nondescriptor segments. The descriptor segments may be used as linkage, parameter, argument, or safe store segments; whereas the nondescriptor 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 described in one of the eight operand descriptor registers (DRn).

Descriptors

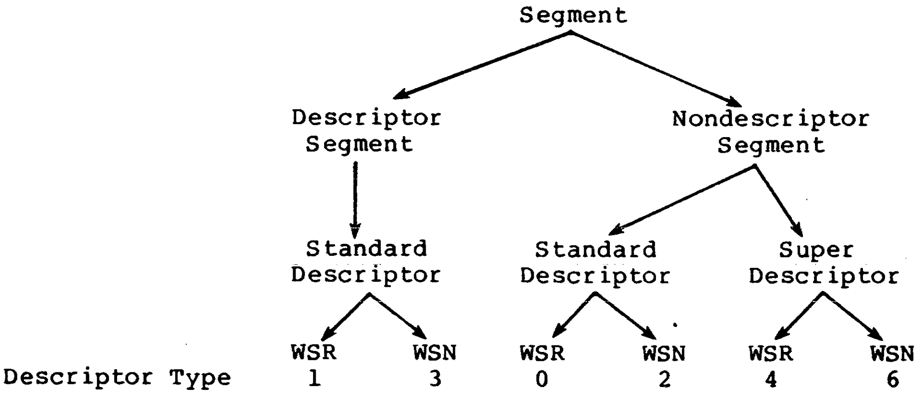
A descriptor consists of a 72-bit word-pair and locates a segment in virtual memory. When the processor hardware obtains a descriptor from memory, the processor assumes that the descriptor is located on an even-word boundary and ignores the least significant bit of the virtual word address. If a descriptor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

Those segments containing instructions, data, or a combination of both (nondescriptor segments) are commonly called operand segments and have descriptors that are either type 0, 2, 4, or 6 to indicate operand storage. The segments containing only descriptors, that is, 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 through descriptor segment descriptors to housekeeping pages.

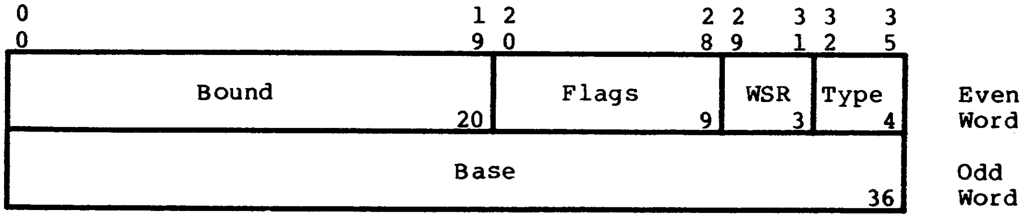
Ten types of descriptors are available. Four of the ten descriptor types are used to define segments that contain data or instructions, and two are used for segments containing segment descriptors. The remaining four descriptors are used only during the execution of the special transfer-of-domain (CLIMB) instruction. The list of descriptor types is given below.

Type	Descriptor	Contents
0	Standard	Instructions/operands
2	Standard with WSN	Operands
4	Super	Operands
6	Super with WSN	Operands
1	Standard	Descriptors
3	Standard with WSN	Descriptors
5	Dynamic linking	Used only with CLIMB
8	Entry	
9	Entry	
11	Entry	



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 one-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 0 <<L66 Cache (2K or 8K) is not used for any fetches through this descriptor. <<DPS 8/20 and 8/44 Cache (2K) is not used for fetches through this descriptor. (8K cache not bypassed.) <<DPS 8/47, 8/49, 8/52, 8/62, 8/70, and 88 Cache is always used. Not interpreted by hardware. 1 Cache is utilized for all memory references through this descriptor.
24	X	Reserved for software.
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 is not valid; segment is empty. 1 Bound field is maximum valid address
28	A	Available segment 0 Segment not available; references not allowed. 1 Segment available; references are 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:

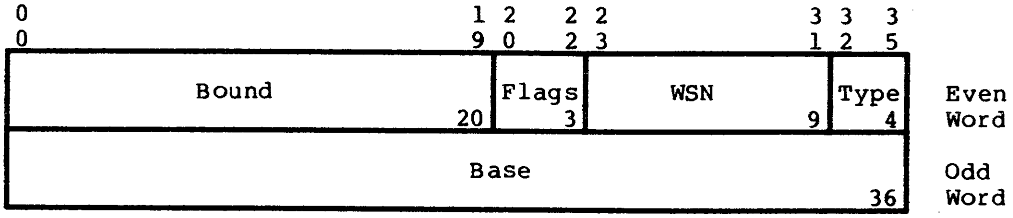
Type = 0 The descriptor "frames" instruction/operand space.

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 with the exception 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:

1. Execute not allowed (NE)
2. Not privileged (NP)
3. Bound valid (B)
4. Segment is available (A)
5. Bypass cache (for DPS 8/20 and 8/44 only) (NC)

WSN - The actual working space number.

Type - The two types of the standard descriptor with WSN are:

Type = 2 The descriptor "frames" operand space.

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

SUPER-DESCRIPTOR

When segments larger than 256K (2**18) words are required, super-descriptors are used to define the 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 1s. 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:

00 09	10 19	20 28	29 31	32 35	
Base 10	Bound 10	Flags 9	WSR 3	Type 4	Even Word
Location 36					Odd Word

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 - The flags field describes the access privileges associated with the descriptor and is 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).

SUPER-DESCRIPTOR WITH WORKING SPACE NUMBER

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

00	09 10	19 20	22 23	31 32	35	
Base 10	Bound 10	Flags 3	WSN 9	Type 4		Even Word
Location 36						Odd Word

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:

1. Execute not allowed (NE)
2. Not privileged (NP)
3. Bound valid (B)
4. Segment is available (A)
5. Bypass cache (For DPS 8/20 and 8/44 only) (NC)

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.

Domains

Another logical element of the virtual environment is the domain. The domain is a flexible and temporary range of operation that may encompass several noncontiguous segments in one or more working spaces (see Figure 3-2). Two or more domains may interact by including the same segment. Each domain contains exactly one linkage segment to define the domain. A change of domain implies a change of linkage segment and vice versa. The linkage segment contains descriptors for the segments constituting the domain. Descriptors for the domain may be in descriptor segments described in the linkage segment, in descriptor registers, or in the parameter segment.

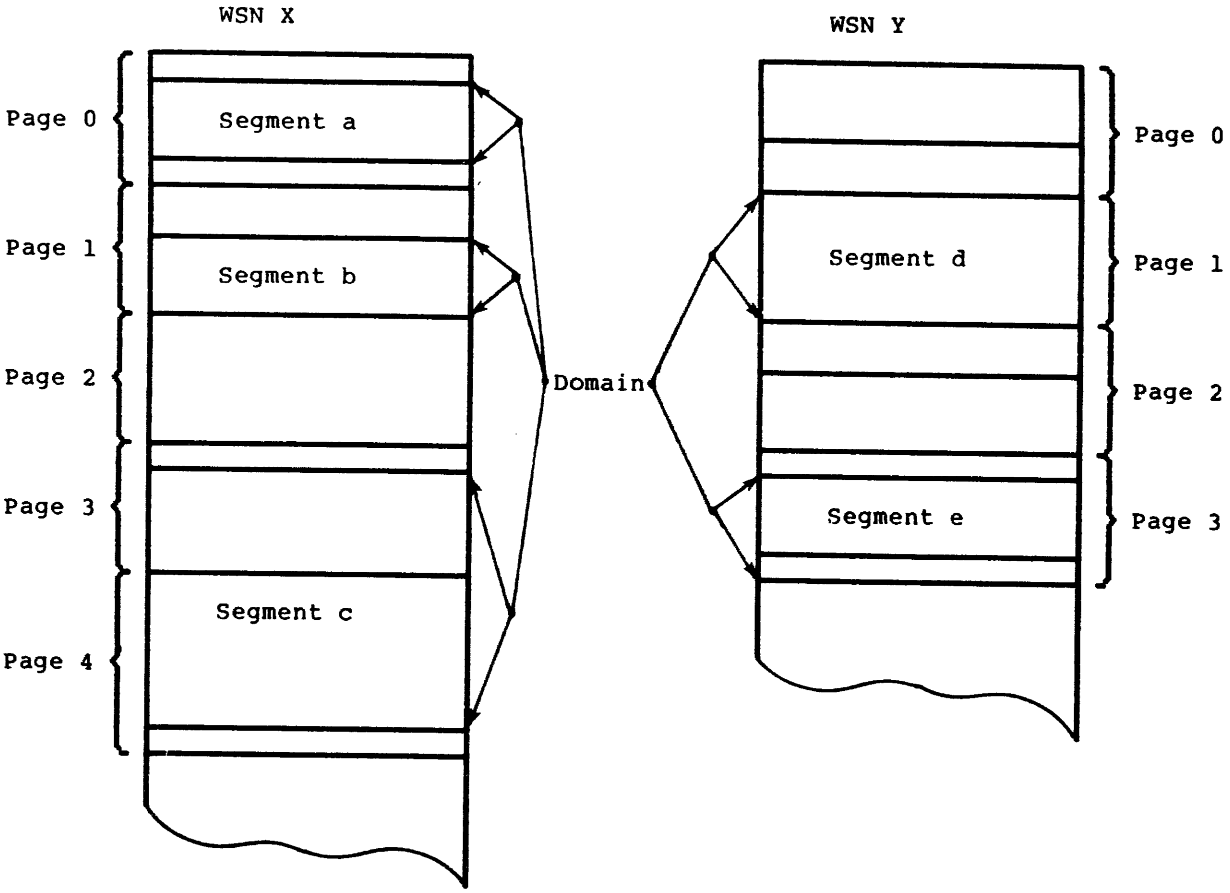


Figure 3-2. Domain Of Noncontiguous Segments

Like the linkage segment, only one argument segment is contained in a domain. This segment provides additional descriptor storage in the form of a descriptor stack which is accessed through the argument stack register (ASR). The stack is empty until descriptors are entered during execution. This segment is used mainly to store descriptors previously loaded in registers, while the registers are used for other descriptors, and to form descriptor segments for communication across domains.

The parameter segment contains one descriptor for each parameter and its contents may vary from call to call. Unlike the descriptors in the linkage segment which are available each time control is passed to a domain, the descriptors in the parameter segment are specific to the call and become unavailable when control is returned from the called domain. Thus, the descriptors in the parameter segment for a domain provide accessibility in the called domain to the described segments only while the call is active.

The bounds and forms of access of the domain are set by the descriptors that define the segments that contain the items to be accessed within a domain. Change from one domain to another is normally performed by the execution of an ICLIMB instruction that establishes a new linkage segment and, usually, a new parameter segment. An interrupt or fault also causes a change of domain.

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 domain registers (LSR, ASR, PSR, and DSAR), as well as those of any other registers specified by the type of entry descriptor, are safestored.

The change-of-domain CLIMB instruction indicates whether there are parameters and the number of arguments. The arguments may be either vectors or descriptors. If the arguments are vectors, descriptors are prepared for the vectors, stored in the parameter segment of the called domain, and the argument segment becomes empty. Refer to the description of the LDDn instruction documented later, for information concerning vector operations.

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, the contents of the calling-domain´s domain registers and any other register contents that were safestored are restored.

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:

```
0                        1 1 1                    2 2    3 3   3
0                        7 8 9                    8 9    1 2   5
+--------------------------+-+----------------------+------+------+
|      Entry Location      |F|       ISEG No.       | WSR  | Type |  Even
|                        18| |                    10|     3|     4|  Word
+----------------+---------+-+----------------------+------+------+
|     LBOUND     |            Linkage Base                  000   |  Odd
|              10|                                       26       |  Word
+----------------+------------------------------------------------+
```

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 bit and is interpreted the same as flag bit 22 of the standard and super-descriptors.

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:

```
      11  = bits 0-1
ISEG No. = bits 2-11
```

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 available to software for defining how the linkage is to be resolved. Bit 22 is for store permission. A dynamic linking fault will occur 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 one equal to zero in a type T=1 or 3 segment causes an SCL2 fault. The dynamic linking descriptor has the following format:

00	22	31	32 35	
Reserved for Software 22	1	Reserved for Software 9	Type 4	Even Word
Reserved for Software 36				Odd Word

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

A feature commonly used to provide descriptor access control is called shrinking. 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.

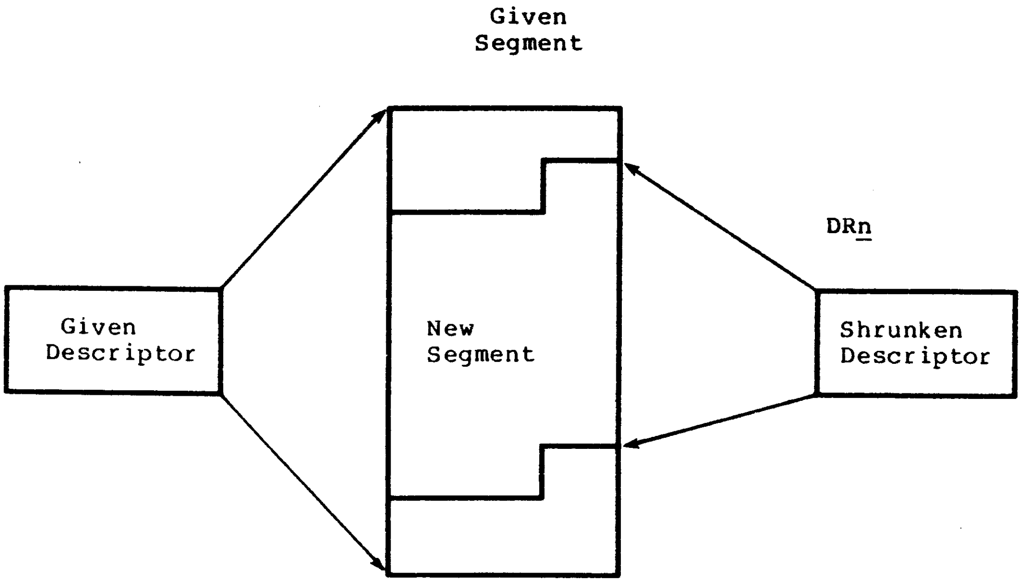


Figure 3-3. Shrunken Descriptor For Corresponding New Segment

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 has store permission.

Shrinking is done via the Load Descriptor Register n (LDDn) instruction, or a domain call or transfer version of the CLIMB instruction (ICLIMB or PCLIMB). In both instances, 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 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 Load Descriptor instruction may be in the same segment as the Load Descriptor Register n 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. The pointer 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.

SECTION IV

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 "Processor Instructions" section 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
Index Registers	Xn	18	8
Indicator Register	IR	18	1
Timer Register	TR	27	1
Instruction Counter	IC	18	1
Address Registers	ARn	24	8
Mode Register (Not in DPS 88)	MR	34	1
Cache Mode Register (Not in DPS 88)	CMR	28	1
Fault Register	FR	72	1
Control Unit History Registers (Not in DPS 88)	CUn	72	16
Operations Unit History Registers (Not in DPS 88)	OUn	72	16
Decimal Unit History Registers (Not in DPS 88)	DUn	72	16
Virtual Unit History Registers (Not in DPS 88)	VUn	72	16
Working Space Registers	WSRn	9	8
Safe Store Register	SSR	72	1
Linkage Segment Register	LSR	72	1
Argument Stack Register	ASR	72	1
Parameter Stack Register	PSR	72	1
Instruction Segment Register	ISR	72	1
Operand 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
Data Stack Descriptor Register	DSDR	72	1
Data Stack Address Register (DPS 8)	DSAR	17	1
Data Stack Address Register (DPS 88)	DSAR	15	1
Page Directory Base Register (DPS 8)[3]	PDBR	15	1
Page Directory Base Register (DPS 88)	PDBR	17	1
Option Register (DPS 8)[3]	OR	3	1
Option Register (DPS 88)	OR	36	1
Pointer and Length Registers	P&L	36	8
Pointer and Length Registers (DPS 88)[4]	P&L	36	2
Stack Control Register	SCR	2	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] The PDBR uses 15 bits for DPS 8; 17 for DPS 88.
The OR uses 3 bits for DPS 8; 36 for DPS 88.

[4] The pointer and length registers are described later in this document.

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.

ACCUMULATOR REGISTER (A)

Format: 36 bits

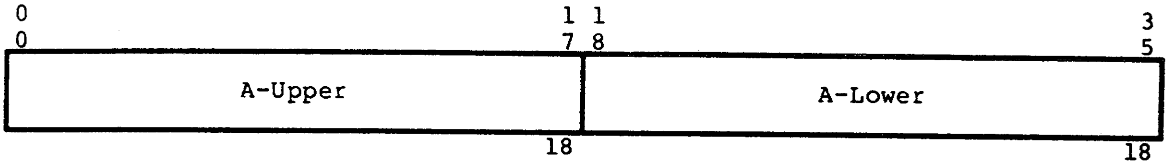


Figure 4-1. Accumulator Register (A) Format

Description:

A 36-bit physical register.

Function:

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

In floating-point binary 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

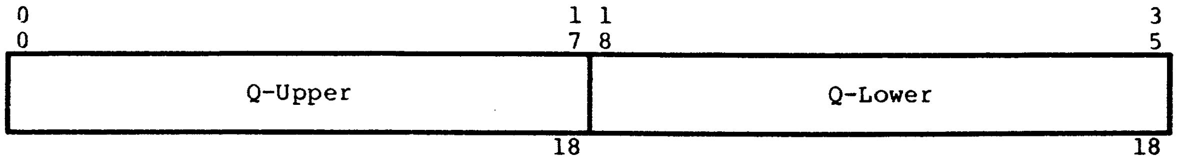


Figure 4-2. Quotient Register (Q) Format

Description:

A 36-bit physical register.

Function:

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

In floating-point binary 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

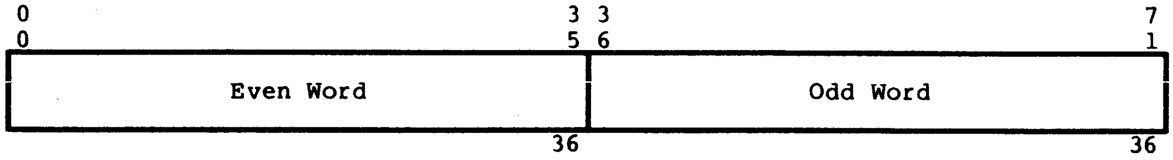


Figure 4-3. Accumulator-Quotient Register (AQ) Format

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 binary instructions, holds the mantissa and the result.

In shifting instructions, holds original data and shifted results.

EXPONENT REGISTER (E)

Format: 8 bits

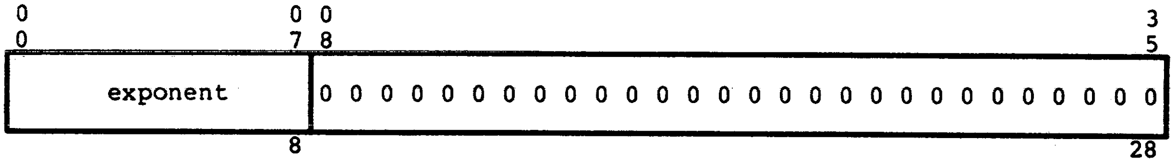


Figure 4-4. Exponent Register (E) Format

Description:

An 8-bit physical register.

Function:

In floating-point binary instructions, holds the exponent.

EXPONENT-ACCUMULATOR-QUOTIENT REGISTER (EAQ)

Format: 80 bits

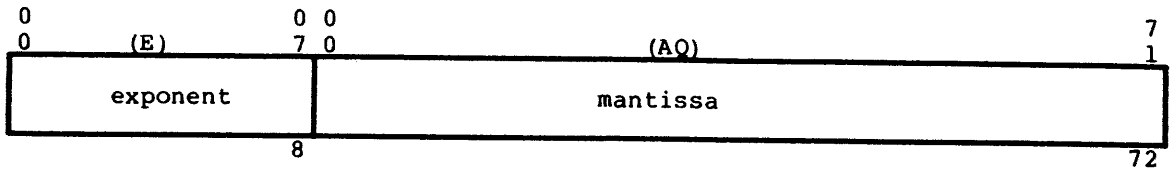


Figure 4-5. Exponent-Accumulator-Quotient Register (EAQ) Format

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" documented later in this manual.

Function:

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

INDEX REGISTERS (Xn)

Format: 18 bits each

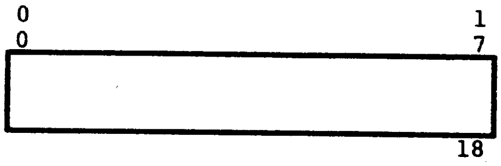


Figure 4-6. Index Register (Xn) Format

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 held extended range bit- or character-string lengths.

INDICATOR REGISTER (IR)

Format: 18 bits

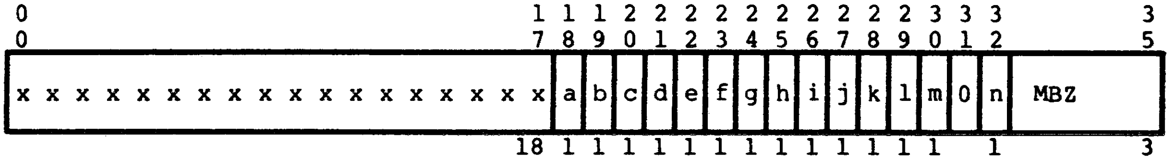


Figure 4-7. Indicator Register (IR) Format

Description:

An assemblage of 14 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 are given below.

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 shifting instruction. (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 is 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 is 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.)

Key	Indicator name	Action
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 is 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.)
g	Overflow mask	This indicator is set ON or OFF only by the instructions that load the IR. When set ON, it inhibits the generation of the fault for those events that normally cause an overflow fault. If the overflow mask indicator is set OFF after occurrence of 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 exhaust. (2) If a Repeat Link (RPL) instruction terminates because of a zero link address. (3) If a tally exhaust is detected for an Indirect then Tally modifier. The instruction is executed whether or not tally exhaust 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 ON whenever a system controller (DSP 88: Central Interface Unit) signals an uncorrectable error or the processor detects an internal parity error condition. The indicator is set OFF only by instructions that load the IR.
j	Parity mask	This indicator is set ON or OFF only by the instructions that load the IR. When it is set ON, it inhibits the generation of the parity fault for all events that set the parity error indicator. If the parity mask indicator is set OFF after the occurrence of 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.

Key	Indicator name	Action
k	Master mode	This indicator is set OFF only by the execution of the Transfer After Setting Slave (TSS) instruction or the execution of an OCLIMB or RET instruction with an operand in which the bit is OFF. It is set ON only by the execution of the PMME version of the CLIMB instruction, the execution of an OCLIMB instruction with an operand in which the bit is ON, or an occurrence of a fault or interrupt.
l	Truncation	This indicator is set ON whenever the target string of a decimal numeric instruction is too small to hold all the digits of the result or the target string of a bit string or alphanumeric instruction is too small to hold all the bits or characters to be stored. Also see the overflow indicator for decimal numeric instructions. The event that sets this indicator ON may also cause an overflow fault. (See overflow mask indicator above.)
m	Multi-word instruction interrupt	This indicator is set OFF by the execution of the SPL instruction and by the end of execution of all multi-word instructions, and is set ON by the events described below. The indicator has meaning only when determining the proper restart resequence for an interrupted multi-word instruction. The events that set this indicator are: (1) Any fault during the execution of a multi-word instruction. (2) Occurrence of an interrupt signal during execution of those multi-word instructions that are interruptible. (3) If the processor is in Master or Privileged Master mode, by the execution of a Load Indicator Register (LDI) or Return (RET) instruction with bit 30 set to 1 in the IR data.
n	Hex mode	This indicator is set ON or OFF only by the instructions that load the IR. When set ON, it causes the floating-point instructions to be executed in the hexadecimal exponent mode if bit 33 of the mode register (DPS 88: bit 0 of the option Register) is also ON. (This function may not be available on all processors.)
MBZ		Bit 31 and bits 33-35 must be zero (MBZ).

TIMER REGISTER (TR)

Format: 27 bits

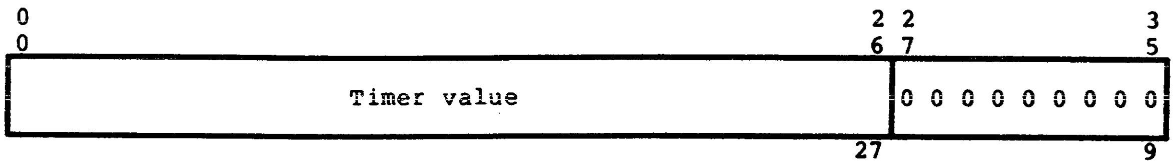


Figure 4-8. Timer Register (TR) Format

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 Delay Until Interrupt Signal (DIS) instruction.

INSTRUCTION COUNTER (IC)

Format: 18 bits

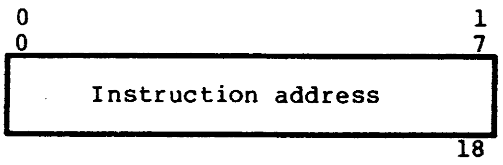


Figure 4-9. Instruction Counter (IC) Format

Description:

An 18-bit physical register.

Function:

Holds the address of the current instruction being executed. The IC is incremented by one by the control unit for the sequential execution of single-word instructions or by the appropriate amount (2, 3, or 4) for multi-word instructions. The content of the IC is changed by a transfer-of-control instruction or by a fault or interrupt. Upon recognition of a fault, the contents of the instruction counter are as shown in the list of faults in Table 4-2.

Faults in Groups I and II terminate the operations in the processor unconditionally.

Faults in Groups III and IV (DPS 88: Groups III, IV, V, VI) terminate the operations in the processor when the operation currently being executed is completed.

Faults in Group V (DPS 88: Group VII) are recognized under the same conditions that program interrupts are recognized. Faults in Group V (DPS 88: Group VII) have priority over program interrupts and are also subject to being inhibited from recognition by engaging the inhibit bit in the instruction word.

Table 4-2. Processor Faults By Priority

Fault Code (5)	Fault Name	Priority		Group Priority		IC Contents (1)
		DPS 8	DPS 88	DPS 8 8/47 8/49	DPS 88	
01100	Startup (SUF)	1	1	I	I	N+0, +1, or +2
01111	Execute (EXF)	2	2	I	I	N+0, +1, or +2
01011	Operation not completed (FONC)	3	4	II	II	N+0, +1, or +2
00111	Lockup (LUF)	4	5	II	II	N+0, +1, or +2
01110	Divide check (FDIV)	5	7	III	III	N(3)
01101	Overflow (FOVF)	6	8	III	III	N
01001	Parity (FPAR)	7		IV	II	N(2)
	DPS 88: (MEM SYS)		6			
00101	Command (FCMD)	8	9	IV	IV	N
00001	Store memory (STR)	9				
	DPS 88: (BND)		10	IV	IV	N(3)
00010	Master mode entry (MME)	10	11	IV	V	N(3)
00110	Derail (DRL)	11	12	IV	V	N(3)
01010	Illegal procedure (IPR)	12	13	IV	V	N
00011	Fault tag (FTAG)	13	14	IV	V	N(3)
10000	Security fault, Class 1 (SCL1)	14	17	IV	V	N
10001	Dynamic linking (DYNLF)	15	18	IV	V	N
10010	Missing segment (MSE)	16	19	IV	VI	N
10011	Missing working space (MWS)	17	20	IV	V	N
10100	Missing page (MPG)	18	21	IV	VI	N
10101	Security fault, Class 2 (SCL2)	19	22	IV	VI	N
00000(4)	Safe store stack fault (SSSF)	20		IV	VI	
10111	DPS 88: (SSSF)		23			
01000	Connect (CON)	21	27	V	VII	N
00100	Timer runout (TROF)	22	28	V	VII	N
00000	Shutdown (SDF)	23	29	V	VII	N

NOTES:

1. N = address of last instruction executed.

2. The processor stops the execution stream at the point where the parity error is detected. Therefore, depending upon what the processor was doing the following may result:

   o If parity fault occurred on operand fetch, operation N+1 was completed with faulty data

   o If parity fault occurred on instruction fetch, operation N+1 was not completed

- o If parity fault occurred on Indirect Tally (IT), IT was not completed

3. These operations are considered complete when the fault is recognized.

4. The Safestore Stack fault occurs in conjunction with a programmed CLIMB instruction, or in conjunction with the wired-in CLIMB instruction that is the result of a fault or interrupt. The Safestore Stack fault is an indication to the operating system that the Safestore Stack has only one or two 64-word frames remaining. See Section VIII for additional information.

5. **** DPS 8: A specific value may not be predictable when the cache memory option is enabled.****

ADDRESS REGISTERS (ARn)

Format: 24 bits each

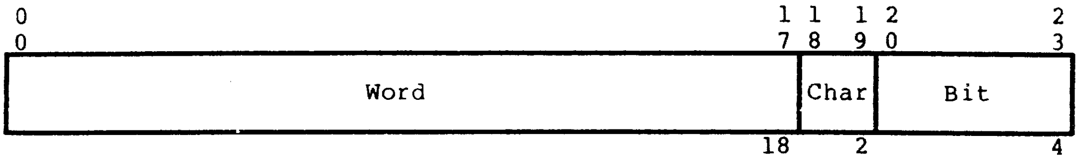


Figure 4-10. Address Register (ARn) Format

Description:

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

Function:

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

Word - 18 bits; a word offset within the segment described by the associated operand 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.

POINTER AND LENGTH REGISTERS (DPS 8)

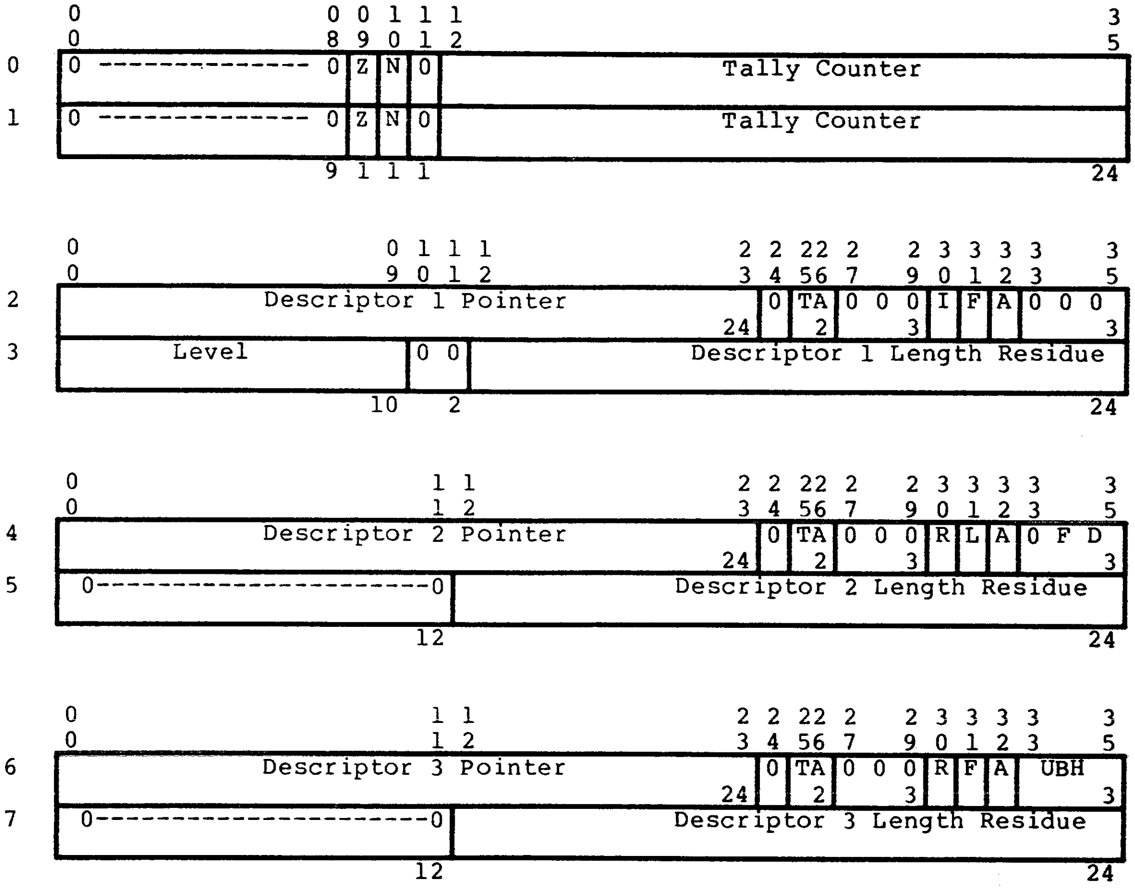


Figure 4-11. Pointer And Length Register Formats (DPS 8)

Z - Bit string instruction results are all zero.

N - Negative overpunch found in 6-4 alphanumeric move.

Tally Counter - The number of characters examined by the SCD, SCDR, SCM, SCMR, TCT, or TCTR instruction (up to the interrupt).

Descriptor Pointer - The last double-word accessed by the descriptor (bits 17-23 valid only for initial access).

TA - Bits 21-22 (alphanumeric type) of each descriptor.

I - Used by hardware to control restarting of interrupted instruction (ignore request).

F - First time. (Information in descriptor is valid.)

A - Used by hardware to control restarting of interrupted instruction.

Level - The difference in the number of characters received by the processor and the number sent from the processor.

L - Logical OR of bits 34-35 of descriptor 2.

D - Descriptor 2 is a direct type (DU).

Descriptor Length Residue - The amount of data left in each descriptor.

R - The last cycle performed must be repeated. (This bit cannot be loaded.)

UBH - Used by hardware; may contain any bit pattern.

POINTER AND LENGTH REGISTERS (DPS 88)

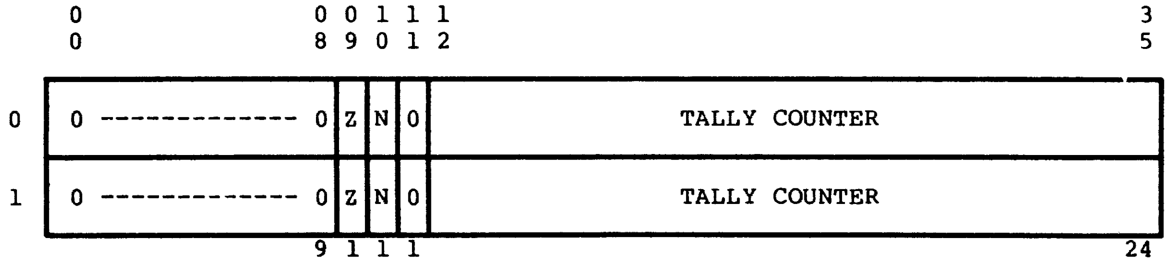


Figure 4-12. Pointer And Length Register Formats (DPS 88)

Z - All bit string instruction results are zero.

N - Negative overpunch found in 6-4 alphanumeric move.

Tally Counter - The number of characters examined by the SCD, SCM, SCMR, TCT, or TCTR instruction up to the interrupt.

MODE REGISTER (MR)

**** DPS 8 ONLY ****

Format: 34 bits

Even-word of Y-pair as stored by Store Central Processor Register (SCPR) instruction with TAG = 06.

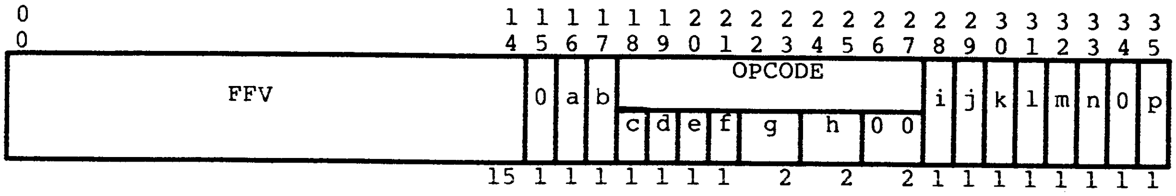


Figure 4-13. Mode Register (MR) Format

Description:

An assemblage of flags and registers from the control unit. The mode register and the cache mode register are both stored into a Y-pair by a Store Central Processor Register (SCPR) instruction with TAG = 06. The mode register is loaded by a Load Central Processor Register (LCPR) instruction with TAG =04.

Function:

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

The functions of the constituent flags and registers are:

Key	Flag or register	Function
	FFV	A floating fault vector address. The 15 high-order bits of the beginning address of an 8-word block constituting a floating fault vector. Traps to these floating faults are generated by other conditions settable by the mode register.
a	OC TRAP	Trap on OPCODE match. If this bit is set ON and OPCODE matches the operation code of the instruction for which an address is being prepared (including indirect cycles), generate the second floating fault (XED FFV+2). (See NOTE below.)
b	ADR TRAP	Trap on ADDRESS match. If this bit is set ON and the computed address (TPR.CA) matches the setting of the address switches on the processor maintenance panel, generate the fourth floating fault (XED FFV+6). (See NOTE below.)
	OPCODE	The operation code on which to trap if OC TRAP (bit 16, key a) is set ON or for which to strobe all control unit cycles into the control unit history registers if O.C$¢ (bit 29, key j) is set ON. or Processor conditions (codes as follows) if OC TRAP (bit 16, key a) and O.C$¢ (bit 29, key j) are set OFF and ¢ VOLT (bit 32, key m) is set ON.

Key Flag or register Function

Key Condition

c Set control unit overlap inhibit if set ON. The control unit waits for the operations unit to complete execution of the even instruction of the current instruction pair before it begins address preparation for the associated odd instruction. The control unit also waits for the operations unit to complete execution of the odd instruction before it fetches the next instruction pair.

d Set store overlap inhibit if set ON. The control unit waits for completion of a current main memory fetch (read cycles only) before requesting a main memory access for another fetch.

e Set store incorrect data parity if set ON. The control unit causes incorrect data parity to be sent to the system controller for the next store instruction and then resets bit 20 (key e).

f Set store incorrect zone-address-command (ZAC) parity if set ON. The control unit causes incorrect zone-address-command (ZAC) parity to be sent to the system controller for each main memory cycle of the next store instruction and resets bit 21 (key f) at the end of the instruction.

g Set timing margins. If ¢ VOLT (bit 32, key m) is set ON and the margin control switch on the processor maintenance panel is in PROG position, set processor timing margins as follows:

22,23	Margin
0,0	normal
0,1	slow
1,0	normal
1,1	fast

h Set +5 voltage margins. If ¢ VOLT (bit 32, key m) is set ON and the margin control switch on the processor maintenance panel is in the PROG position, set +5 voltage margins as follows:

24,25	Margin
0,0	normal
0,1	low
1,0	high
1,1	normal

Key	Flag or register	Function
i		Trap on control unit history register counter overflow if set ON. If this bit and STROBE ¢ (bit 30, key k) are set ON and the control unit history register counter overflows, generate the third floating fault (XED FFV+4). Further, if FAULT RESET (bit 31, key l) is set, reset STROBE ¢ (bit 30, key k), locking the history registers. A Load Central Processor Register (LCPR) instruction (with TAG = 04) that sets bit 28 (key i) ON resets the control unit history register counter to zero. (See NOTE below.)
j	O.C$¢	Strobe control unit history registers on OPCODE match. If this bit and STROBE ¢ (bit 30, key k) are set ON and the operation code of the current instruction matches OPCODE, strobe the control unit history registers on all control unit cycles (including indirect cycles).
k	STROBE ¢	Enable history registers. If this bit is set ON, all history registers are strobed at appropriate points in the various processor cycles. If this bit is set OFF or MR ENABLE (bit 35, key n) is set OFF, all history registers are locked. This bit is set OFF with a Load Central Processor Register (LCPR) instruction (with TAG = 04) providing a 0 bit, by an Operation Not Completed fault and, conditionally, by other faults (see FAULT RESET (bit 31, key l) below). Once set OFF, this bit must be set ON with a Load Central Processor Register (LCPR) instruction (with TAG = 04) providing a 1 bit to re-enable the history registers.
l	FAULT RESET	History register lock control. If this bit is set ON, set STROBE ¢ (bit 30, key k) OFF, locking the history registers for all faults including the floating faults. (See NOTE below.)
m	¢ VOLT	Test mode indicator. This bit is set ON whenever the TEST/NORMAL switch on the processor maintenance panel is in TEST position and is set OFF otherwise. It serves to enable the program control of voltage and timing margins.
n	HEX	Hexadecimal exponent mode floating-point format is enabled.
p	MR ENABLE	Enable mode register. When this bit is set ON, all other bits and controls of the mode register are active. When this bit is set OFF, the mode register controls are disabled.

NOTE: The traps described above (ADDRESS match, OPCODE match, control unit history register counter overflow) occur after completion of the next odd instruction following their detection. The complete priority sequence (in increasing order) is:

1 - Connect
2 - Timer runout
3 - Shutdown
4 - OPCODE trap
5 - Control unit history register counter overflow
6 - Address match trap
7 - Interrupts

****

CACHE MODE REGISTER (CMR)

**** DPS 8 ONLY ****

Format: 28 bits

Odd-word of Y-pair as stored by Store Central Processor Register (SCPR) instruction with TAG = 06.

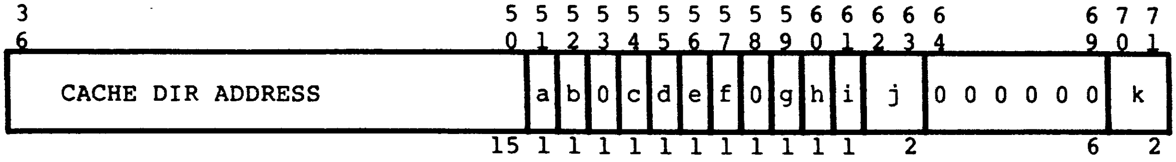


Figure 4-14. Cache Mode Register (CMR) Format

Description:

An assemblage of flags and registers from the control unit. The mode register and cache mode register are both stored into the Y-pair by a Store Central Processor Register (SCPR) instruction with TAG = 06. The cache mode register is loaded by a Load Central Processor Register (LCPR) instruction with TAG = 02.

The data stored from the cache mode register is address-dependent. The algorithm used to map main memory into the cache memory is effective for the Store Central Processor Register (SCPR) instruction. In general, the user may read out data from the directory entry for any cache memory block by proper selection of certain subfields in the 24-bit absolute main memory address. In particular, the user may read out the directory entry for the cache memory block involved in a suspected cache memory error by ensuring that the required 24-bit absolute main memory address subfields are the same as those for the access that produced the suspected error.

The fault handling procedure(s) should bypass cache (segment descriptor bit 23 = 0) and the history registers and cache memory should be disabled as quickly as possible in order that vital information concerning the suspected error not be lost.

Function:

The cache mode register provides configuration information and software control over the operation of the cache memory. Except for those items identified below by an "x" in the column headed L, the cache mode register can be loaded by a Load Central Processor Register (LCPR) instruction with TAG = 02.

The functions of the constituent flags and registers are:

Key	L	Register	Function
	x	CACHE DIR ADDRESS	15 high-order bits of the cache memory block address from the cache directory.
a	x	PAR BIT	Cache memory directory parity bit.
b	x	LEV FUL	The selected column and level is loaded with active data.
c		CSH1 ON	Enable the upper 1024 words of cache memory (4096 words if 8K cache memory).
d		CSH2 ON	Enable the lower 1024 words of cache memory (4096 words if 8K cache memory).
e		OPND ON	Enable cache memory for operands.
f		INST ON	Enable cache memory for instructions.
g		CSH REG	Enable cache-to-register (dump) mode. When this bit is set ON, double-precision operations unit read operands (e.g., Load AQ (LDAQ) operands) are read from the cache memory according to the mapping algorithm and without regard to matching of the full 24-bit main memory address. All other operands address main memory as though the cache memory were disabled. This bit is reset automatically by the hardware for any fault or interrupt.
h	x	STR ASD	Enable store aside. The processor proceeds after the cache memory cycle is complete.
i	x	COL FULL	Selected cache memory column is full.
j	x	RRO A,B	Cache round-robin counter.
k		LUF MSB,LSB	Lockup fault timer setting. The lockup fault timer may be set to one of four different values according to the value of this field.

LUF value	Lockup time
0	2 ms
1	4 ms
2	8 ms
3	16 ms

The lockup timer is set to 32 ms when the processor is initialized in Master mode.

****

FAULT REGISTER (FR)

**** DPS 8 ONLY ****

Format: 72 bits

Even-word of Y-pair as stored by Store Central Processor Register (SCPR) instruction with TAG = 01.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16 19	20 23	24 27	28 31	32	33	34	35
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	0	IAA	IAB	IAC	IAD	p	q	r	s
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	1	1	1	1

Odd-word of Y-pair as stored by Store Central Processor Register (SCPR) instruction with TAG = 01 (if 8K cache memory is installed).

36	37	38	39	40	41	42	43	44	45	46	47 71
t	u	v	w	x	y	z	a	b	c	d	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1	1	1	1	1	1	1	1	1	1	1	25

Figure 4-15. Fault Register (FR) Format

Description:

A combination of flags and registers all located in the control unit. The register is stored and cleared by a Store Central Processor Register (SCPR) instruction with TAG = 01. Note that the data is stored into the word pair at location Y and that bits 47-71 of Y+1 are cleared. The fault register cannot be loaded.

Function:

The fault register contains the conditions in the processor for several of the hardware faults. Data is stored into the fault register during a fault sequence. Once a bit or field in the fault register is set, it remains set until the register is stored and cleared. The data is not overwritten during subsequent fault events, except that bits 16-31 are stored for each memory error and may be overwritten.

The functions of the constituent flags and registers are:

Key	Flag or register	Function
a	ILL OP	An illegal operation code was detected.
b	ILL MOD	An illegal address modifier was detected.
c	ILL SLV	An illegal slave procedure was encountered.
d	ILL PROC	All illegal procedure other than the above three was encountered.
e	NEM	A nonexistent main memory address was requested.
f	OOB	A boundary violation occurred.
g	DU MISC	An illegal decimal digit or sign or invalid micro-operation was detected by the decimal unit.
h	PROC PARU	A parity error was detected in the upper 36 bits of data.
i	PROC PARL	A parity error was detected in the lower 36 bits of data.
j	$CON A	A $CONNECT signal was received through port A.
k	$CON B	A $CONNECT signal was received through port B.
l	$CON C	A $CONNECT signal was received through port C.
m	$CON D	A $CONNECT signal was received through port D.
n	DA ERR1	Operation is not complete. Processor/system controller interface sequence error 1 was detected. ($DATA-AVAIL received with no prior $INTERRUPT sent.)
o	DA ERR2	Operation not complete. Processor/system controller interface sequence error 2 has been detected. (Multiple $DATA-AVAIL received or $DATA-AVAIL received out of order.)
	IAA	Coded illegal action, port A (see Table 4-3).
	IAB	Coded illegal action, port B (see Table 4-3).
	IAC	Coded illegal action, port C (see Table 4-3).
	IAD	Coded illegal action, port D (see Table 4-3).
p	CPAR DIR	A parity error was detected in the cache memory primary directory.
q	CPAR STR	A data parity error was detected in the cache memory.
r	CPAR IA	An illegal action was received from a system controller during a store operation with cache memory enabled. This implies that the data is correct in cache memory and incorrect in main memory.
s	CPAR BLK	A cache memory parity error occurred during a cache memory data block load.

The following functions are stored only if the 8K cache memory option is installed:

Key	Flag or register	Function
t	BUFO-A	Buffer overflow, port A
u	BUFO-B	Buffer overflow, port B
v	BUFO-C	Buffer overflow, port C
w	BUFO-D	Buffer overflow, port D
x	BUFO-PD	Buffer overflow, primary directory
y	WNI-PE	Interface parity error, system controller to processor (any port)
z	DIR-0-PE	Parity error, level 0
a	DIR-1-PE	Parity error, level 1
b	DIR-2-PE	Parity error, level 2
c	DIR-3-PE	Parity error, level 3
d	MTCH-ERR	Multimatch error (duplicate directory)

Table 4-3. System Controller Illegal Action Codes

Code (Octal)	Priority	Fault	Reason
00	--	None	No illegal action
01	--	Command	Unassigned
02	05	Store	Nonexistent address
03	01	Command	Stop on condition[1]
04	--		Unassigned
05	12	Parity	Data parity, store unit to system controller
06	11	Parity	Data parity in store unit
07	10	Parity	Data parity in store unit and store unit to system controller
10	04	Command	NOT control[1]
11	13	Command	Port not enabled
12	03	Command	Illegal command
13	07	Store	Store unit not ready
14	02	Parity	Zone-address-command parity, processor to system controller
15	06	Parity	Data parity, processor to system controller
16	08	Parity	Zone-address-command parity, system controller to store unit
17	09	Parity	Data parity, system controller to store unit

[1] Fault not returned if 4 megaword system controller

****

FAULT REGISTER FORMAT

**** DPS 88 ONLY ****

Table 4-4. Fault Register Format

Reg. Bit	Prior-ity	Group	Fault Mnemonic	Description
00	1	1	SUF	Start Up Fault
01	2	1	EXF	Execute Fault
02	3	2	---	(undefined)
03	4	2	ONC	Operation Not Complete Fault
04	5	2	LUF	Lockup Fault
05	6	2	MEMSYS	Memory System Fault
06	7	3	DIV	Divide Check Fault
07	8	3	OFL	Overflow Fault
08	9	4	CMD	Command Fault
09	10	4	BND	Bound Fault
10	11	5	MME	Master Mode Entry Fault
11	12	5	DRL	Derail Fault
12	13	5	IPR	Illegal Procedure Fault
13	14	5	FTAG	Fault Tag
14	15	5	---	(undefined)
15	16	5	---	(undefined)
16	17	5	SCL1	Security Fault, Class 1
17	18	5	DYNL	Dynamic Linking Fault
18	19	6	MSE	Missing Segment Fault
19	20	5	MWS	Missing Work Space Fault
20	21	6	MPG	Missing Page Fault
21	22	6	SCL2	Security Fault, Class 2
22	23	6	SSSF	Safe-store Stack Fault
23	24	6	---	(undefined)
24	25	7	DIS	DIS Hypermode Entry Fault
25	26	7	CIOC	CIOC Hypermode Entry Fault
26	27	7	CON	Connect Received Fault (CPU is destination)
27	28	7	TRO	Timer Runout Fault
28	29	7	SDF	Shut Down Fault
29	30	7	---	(undefined)
30	31	7	---	(undefined)
31	32	7	HTRO	Hypertimer Runout
32	33	---	IFLT	Interrupt
33 34 35	Bits 33, 34, and 35 are currently not implemented. On occurrence of a SFR instruction, these bits are zeroed.			

****

CONTROL UNIT HISTORY REGISTERS (CUn)

**** DPS 8 ONLY ****

Format: 72 bits each

Even-word of Y-pair as stored by Store Central Processor Register (SCPR) instruction with TAG = 20.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18 27	28	29	30 35
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	OPCODE	I	P	TAG
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	10	1	1	6

Odd-word of Y-pair as stored by Store Central Processor Register (SCPR) instruction with TAG = 20.

36 53	54 58	59 6	6	6	6	6	6	6	6	7	7
ADDRESS	CMD	SEL	s	t	u	v	w	x	y	z	*
18	5	4	1	1	1	1	1	1	1	1	1

Figure 4-16. Control Unit History Register (CUn) Format

Description:

Sixteen combinations of flags and registers from the control unit (may be optionally increased). The sixteen registers are handled as a rotating queue controlled by the control unit history register counter. The counter is always set to the number of the oldest entry and advances by one for each history register reference (data entry or Store Central Processor Register (SCPR) instruction). Multicycle instructions such as Load Address Registers (LAREG) have an entry for each of their cycles.

Function:

A control unit history register entry shows the conditions at the end of the control unit cycle to which it applies. The sixteen registers hold the conditions for the last sixteen control unit cycles. Entries are made according to controls set in the mode register.

NOTE: Bits 54-71 of the odd-word of the control unit history register are undefined when the virtual memory option is installed and operational.

The meanings of the constituent flags and registers are:

Key	Flag Name	Meaning
a	PIA	1 = Prepare instruction address
b	POA	1 = Prepare operand address
c	RIW	1 = Request indirect word
d	SIW	1 = Restore indirect word
e	POT	1 = Prepare operand tally (indirect tally chain)
f	PON	1 = Prepare operand no tally (as for POT except no chain)
g	RAW	1 = Request read-alter-rewrite word
h	SAW	1 = Restore read-alter-rewrite word
i	TRGO	1 = Transfer GO (conditions met)
j	XDE	1 = Execute even instruction from Execute Double (XED) pair
k	XDO	1 = Execute odd instruction from Execute Double (XED) pair
l	IC	1 = Execute odd instruction of the current pair
m	RPTS	1 = Execute a repeat instruction
n	WI	1 = Wait for instruction fetch
o	AR F/E	1 = Address register has valid data
p	$\overline{\text{XIP}}$	1 = NOT prepare interrupt address
q	$\overline{\text{FLT}}$	1 = NOT prepare fault address
r	$\overline{\text{BASE}}$	1 = NOT slave mode
	OPCODE	Operation code from current instruction word
	I	Interrupt inhibit bit from current instruction word
	P	Pointer register flag bit from current instruction word
	TAG	Current address modifier (this modifier is replaced by the contents of the TAG fields of indirect words as they are fetched during indirect chains)
	ADDRESS	Current computed address (lower 18 bits)
	CMD	System controller command
	SEL	Port select bits (valid only if port A-D is selected)
s	XEC-INT	1 = An interrupt is present
t	INS-FETCH	1 = Perform an instruction fetch
u	CU-STORE	1 = Control unit store cycle

Key	Flag Name	Meaning
v	OU-STORE	1 = Operations unit store cycle
w	CU-LOAD	1 = Control unit load cycle
x	OU-LOAD	1 = Operations unit load cycle
y	DIRECT	1 = Direct cycle (for example, DU, DL, shift)
z	$\overline{\text{PC-BUSY}}$	1 = Port control logic not busy
*	BUSY	1 = Port interface busy/cache memory read

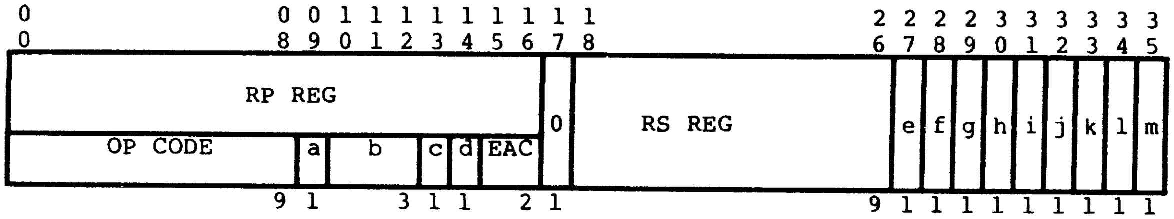
****

OPERATIONS UNIT HISTORY REGISTERS (OUn)

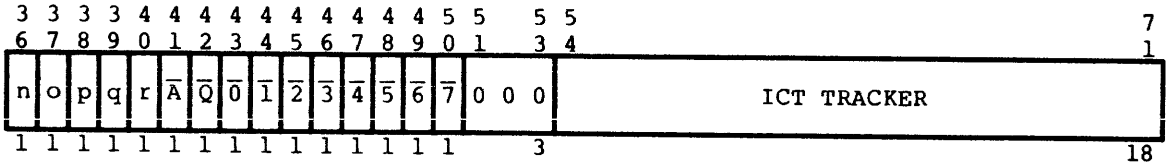
**** DPS 8 ONLY ****

Format: 72 bits each

Even-word of Y-pair as stored by Store Central Processor Register (SCPR) instruction with TAG = 40.



Odd-word of Y-pair as stored by Store Central Processor Register (SCPR) instruction with TAG = 40.



3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5 7
6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 3 4 1
n o p q r $\overline{A}$ $\overline{Q}$ $\overline{0}$ $\overline{1}$ $\overline{2}$ $\overline{3}$ $\overline{4}$ $\overline{5}$ $\overline{6}$ $\overline{7}$
0 0 0
ICT TRACKER
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 18

Figure 4-17. Operations Unit History Register (OUn) Format

Description:

Sixteen combinations of flags and registers from the operations unit and control unit (may be optionally increased). The sixteen registers are handled as a rotating queue controlled by the operations unit history register counter. The counter is always set to the number of the oldest entry and advances by one for each history register reference (data entry or Store Central Processor Register (SCPR) instruction).

Function:

An operations unit history register entry shows the conditions at the end of the operations unit cycle to which it applies. The sixteen registers hold the conditions for the last sixteen operations unit cycles. As the operations unit performs various cycles in the execution of an instruction, it does not advance the counter for each such cycle. The counter is advanced only at successful completion of the instruction or if the instruction is terminated for a fault condition. Entries are made according to controls set in the mode register.

The meanings of the constituent flags and registers are:

Key	Flag Name	Meaning
	RP REG	Primary operations unit operation register. RP REG receives the operation code and other data for the next instruction from the control unit during the control unit instruction fetch cycle while the operations unit may be busy with a prior instruction. RP REG is further substructured as:
	OP CODE	The 9 high-order bits of the 10-bit operation code from the instruction word. Note that some instructions do not involve bit 27; hence the 9-bit field is sufficient to determine the instruction.
a	9 CHAR	Character size for Indirect then Tally address modifiers (bit 30 of IT word): 0 = 6-bit 1 = 9-bit
b	TAG1,2,3	The 3 low-order bits of the address modifier from the instruction word. This field may contain a character position for an Indirect then Tally address modifier.
c	CR FLG	Character modification (IT) flag
d	DR FLG	Direct operation flag (0 = DU, 1 = DL)
	EAC	Address counter for LREG/SREG instructions
	RS REG	Secondary operations unit operation register. OP CODE is moved from RP REG to RS REG during the operand fetch cycle and is held until completion of the instruction.
e	FRB1-FULL	1 = OP CODE buffer is loaded
f	FRP-FULL	1 = RP REG is loaded
g	FRS-FULL	1 = RS REG is loaded

Key	Flag Name	Meaning
h	FGIN	1 = First cycle for all OU operations. RP operation code in execution
i	FGOS	1 = Second cycle for multicycle OU operations
j	FGD1	1 = First divide cycle
k	FGD2	1 = Second divide cycle
l	FGOE	i = Exponent compare cycle
m	FGOA	1 = Mantissa alignment cycle
n	FGOM	1 = General operations unit cycle
o	FGON	1 = Normalize cycle
p	FGOF	1 = Final operations unit cycle
q	FSTR-OP-AV	1 = Store (output) data available (reset by CU)
r	$\overline{\text{DA-AV}}$	1 = Data not available
$\overline{\text{A}}$	$\overline{\text{A-REG}}$	1 = A-register not in use
$\overline{\text{Q}}$	$\overline{\text{Q-REG}}$	1 = Q-register not in use
$\overline{0}$	$\overline{\text{X0-RG}}$	1 = X0 not in use
$\overline{1}$	$\overline{\text{X1-RG}}$	1 = X1 not in use
$\overline{2}$	$\overline{\text{X2-RG}}$	1 = X2 not in use
$\overline{3}$	$\overline{\text{X3-RG}}$	1 = X3 not in use
$\overline{4}$	$\overline{\text{X4-RG}}$	1 = X4 not in use
$\overline{5}$	$\overline{\text{X5-RG}}$	1 = X5 not in use
$\overline{6}$	$\overline{\text{X6-RG}}$	1 = X6 not in use
$\overline{7}$	$\overline{\text{X7-RG}}$	1 = X7 not in use
	ICT TRACKER	The current value of the instruction counter. Since the control unit and operations unit run asynchronously and overlap is usually enabled, the value of ICT TRACKER may not be the address of the operations unit instruction currently being executed.

****

DECIMAL UNIT HISTORY REGISTERS (DUn)

**** DPS 8 ONLY ****

Format: 72 bits each

Decimal unit history register data is stored by a Store Central Processor Register (SCPR) instruction with TAG = 10. No format diagram is given since the data is defined as individual bits.

Description:

Sixteen combinations of flags from the decimal unit (may be optionally increased). The sixteen registers are handled as a rotating queue controlled by the decimal unit history register counter. The counter is always set to the number of the oldest entry and advances by one for each history register reference (data entry or Store Central Processor Register (SCPR) instruction).

The decimal unit and the control unit run synchronously. There is a control unit history register entry for every decimal unit history register entry and vice versa (except for instruction fetch and descriptor fetch cycles). If the processor is not executing a decimal instruction, the decimal unit history register entry shows an idle condition.

Function:

A decimal unit history register entry shows the conditions in the decimal unit at the end of the control unit cycle to which it applies. The sixteen registers hold the conditions for the last sixteen control unit cycles. Entries are made according to controls set in the mode register.

A minus sign (-) preceding the flag name indicates that the complement of the flag is shown. Unused bits are set ON.

The meanings of the constituent flags are:

Bit	Flag Name	Meaning
0	-FPOL	Prepare operand length
1	-FPOP	Prepare operand pointer
2	-NEED-DESC	Need descriptor
3	-SEL-ADR	Select address register
4	-DLEN=DIRECT	Length equals direct
5	-DFRST	Descriptor processed for first time
6	-FEXR	Extended register modification
7	-DLAST-FRST	Last cycle of DFRST

Bit	Flag Name	Meaning	
8	-DDU-LDEA	Decimal unit load	
9	-DDU-STAE	Decimal unit store	
10	-DREDO	Redo operation without pointer and length update	
11	-DLVL<WD-SZ	Load with count less than word size	
12	-EXH	Exhaust	
13	DEND-SEQ	End of sequence	
14	-DEND	End of instruction	
15	-DU=RD+WRT	Decimal unit read or write	
16	-PTRA00	PR address bit 0	load/store registers
17	-PTRA01	PR address bit 1	
18	FA/I1	Descriptor 1 active	
19	FA/I2	Descriptor 2 active	
20	FA/I3	Descriptor 3 active	
21	-WRD	Word operation	
22	-NINE	9-bit character operation	
23	-SIX	6-bit character operation	
24	-FOUR	4-bit character operation	
25	-BIT	Bit operation	
26		Unused	
27		Unused	
28		Unused	
29		Unused	
30	FSAMPL	Sample for multiword instruction interrupt	
31	-DFRST-CT	Specified first count of a sequence	
32	-ALI	Adjust length	
33	-MIF	Multiword instruction interrupt	
34	-INHIB-STC1	Inhibit STC1 (force "STC0")	
35		Unused	
36	DUD	Decimal unit idle	
37	-GDLDA	Descriptor load gate A	
38	-GDLDB	Descriptor load gate B	
39	-GDLDC	Descriptor load gate C	

Bit	Flag Name	Meaning
40	NLD1	Prepare alignment count for first numeric operand load
41	GLDP1	Numeric operand one load gate
42	NLD2	Prepare alignment count for second numeric operand load
43	GLDP2	Numeric operand two load gate
44	ANLD1	Alphanumeric operand one load gate
45	ANLD2	Alphanumeric operand two load gate
46	LDWRT1	Load rewrite register one gate
47	LDWRT2	Load rewrite register two gate
48	-DATA-AVLDU	Decimal unit data available
49	WRT1	Rewrite register one loaded
50	GSTR	Numeric store gate
51	ANSTR	Alphanumeric store gate
52	-FSTR-OP-AV	Operand available to be stored
53	-FEND-SEQ	End sequence flag
54	-FLEND<128	Length less than 128
55	FGCH	Character operation gate
56	FANPK	Alphanumeric packing cycle gate
57	FEXMOP	Execute MOP gate
58	FBLNK	Blanking gate
59		Unused
60	DGBD	Binary-to-decimal execution gate
61	DGDB	Decimal-to-binary execution gate
62	DGSP	Shift procedure gate
63	FFLTG	Floating result flag
64	FRND	Rounding flag
65	DADD-GATE	Add/subtract execution gate
66	DMP+DV-GATE	Multiply/divide execution gate
67	DXPN-GATE	Exponent network execution gate
68		Unused
69		Unused
70		Unused
71		Unused

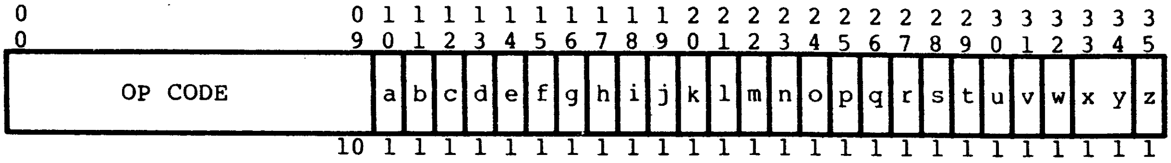
****

VIRTUAL UNIT HISTORY REGISTERS (VUn)

**** DPS 8 ONLY ****

Format: 72 bits each

Even-word of Y-pair as stored by Store Central Processor Register (SCPR) instruction with TAG = 00.



Odd-word of Y-pair as stored by Store Central Processor Register (SCPR) instruction with TAG = 00.

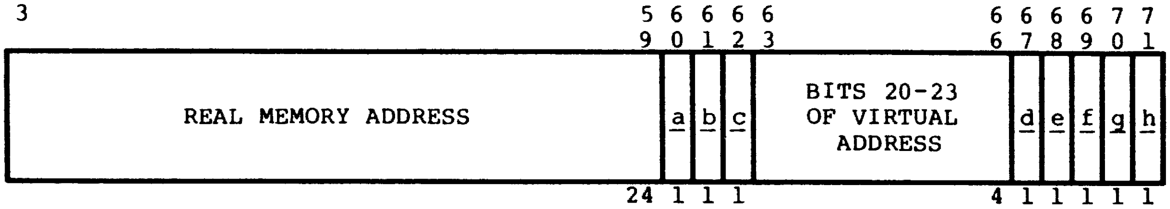


Figure 4-18. Virtual Unit History Register (VUn) Format

Description:

Sixteen combinations of flags and registers from the virtual unit (may be optionally increased). The sixteen registers are handled as a rotating queue controlled by the virtual unit history register counter. The counter is always set to the number of the oldest entry and advances by one for each history register reference (data entry or Store Central Processor Register (SCPR) instruction).

NOTE: The virtual memory option must be installed in the processor and enabled to enter data into and retrieve data from the virtual unit history registers.

Function:

A virtual unit history register entry shows the conditions in the virtual unit at the end of an address preparation cycle in the virtual mode. The sixteen registers hold the conditions for the last sixteen such address preparation cycles. Entries are made according to controls set in the mode register.

The meanings of the constituent flags and registers are:

Key	Flag Name	Meaning
	OP CODE	The ten bits of the operation code from the instruction word.
a	DFA	Final address preparation cycle
b	FABS	Absolute address preparation cycle
c	FAM-MCH	Associative memory match
d	FPTD	Fetch page table directory word cycle
e	FKTW	Fetch key table word cycle
f	FPTWK	Fetch page table word (PTW) cycle for fragmented page table
g	FPTWD	Fetch PTW cycle for dense page table
h	FWR-PTW	Write (modify) PTW cycle
i	FV1	Fetch vector word 0 and 1 cycle
j	FV2	Fetch vector word 2 and 3 cycle
k	FCD	Fetch descriptor for copy or shrink cycle
l	FCLR	Clear memory
m	FSAS	Store to argument stack cycle
n	FXD	Fetch transfer descriptor cycle
o	FFXD	Fetch fault/interrupt transfer descriptor cycle
p	FXID	Fetch transfer descriptor from indirect cycle
q	FDT	Domain transfer
r	FIDT	Interdomain transfer
s	FSSW	Safe store write cycle
t	FVU-OP	Virtual unit operational
u	FSSR	Safe store read cycle
v	DSLAVE	Slave mode
w	DMASTER	Master mode
x	FVU-STR-FLT	Store fault
y	FVU-CMD-FLT	Command fault
z	FVU-ILP-FLT	Virtual unit IPR fault
	Bits 36-59 are RADDR00 through RADDR23	Real memory address

Key	Flag Name	Meaning
a	FPIA-VU	Prepare instruction address for virtual unit
b	FTRGO-VU	Transfer to GO flag for virtual unit
c	FEA-VU	Effective address for virtual unit
	Bits 63-66 are RVA20 through RVA23	Virtual address bits 20-23; associative memory row select
d	DAMSEL1+3	Associative memory column select
e	DAMSEL2+3	
f	FVU-FAULT	Virtual unit fault indicator
g	EXT-SEG-FLG	External segment flag
h	FHOLD-START	Inhibit virtual unit initialization

****

WORKING SPACE REGISTERS (WSRn)

Format: 9 bits each

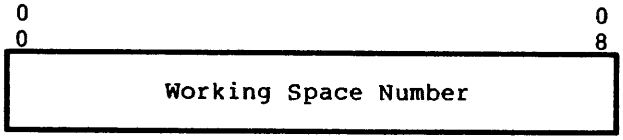


Figure 4-19. Working Space Register (WSRn) Format

Description:

Eight 9-bit registers located in the virtual unit that hold the 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 (DPS 88: working space registers 0-7 are set to zeros). The working space registers provide the means for sharing and isolating working spaces.

SAFE STORE REGISTER (SSR)

Format: 72 bits

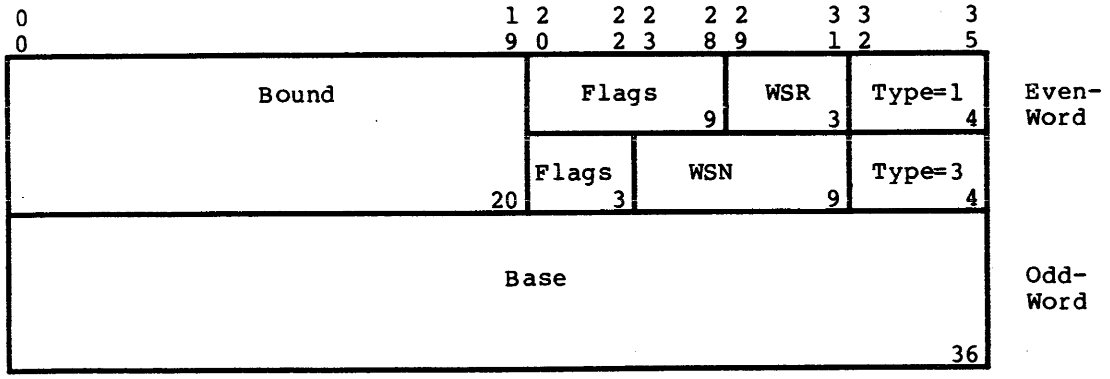


Figure 4-20. Safe Store Register (SSR) Format

Description:

A 72-bit register located in the virtual unit that holds 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 (see Figure 8-3). 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 SCR determines the size of the safe store frame as follows:

00 - 16 words
01 - 24 words
11 - 64 words

When the frame size is 64 words, the actual number of words stored may depend on the state of indicator register bit 30 (multiword instruction interrupt or fault). The actual number of words stored is:

**** DPS 8:

DPS 8/70, 8/50, 8/52, and 8/62 store 48 words; however, if IR bit 30=1, 56 words will be stored.

DPS 8/20 and 8/44 store 48 words; however, if IR bit 30=1, 52 words will be stored. ****

**** DPS 88 stores 50 words. ****

LINKAGE SEGMENT REGISTER (LSR)

Format: 72 bits

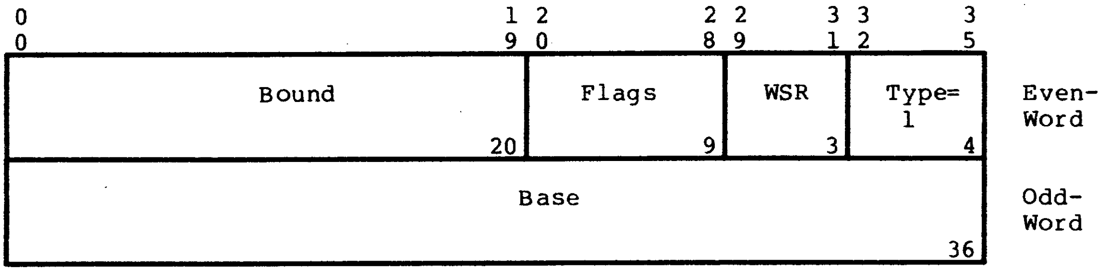


Figure 4-21. Linkage Segment Register (LSR) Format

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 operand 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.

ARGUMENT STACK REGISTER (ASR)

Format: 72 bits

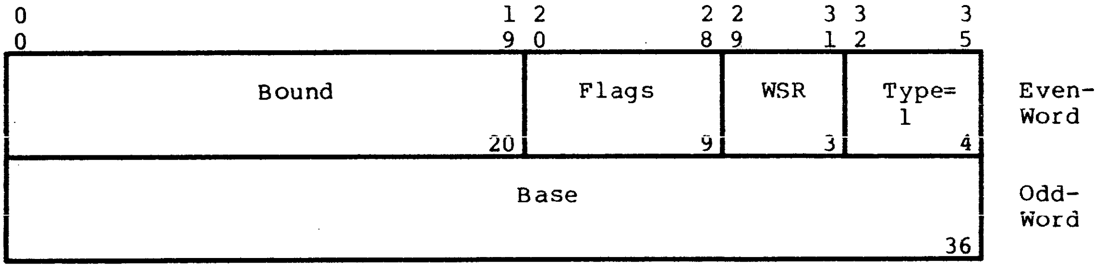


Figure 4-22. Argument Stack Register (ASR) Format

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 (not empty), bits 17-19 are forced to 111. Thus, the size of the argument stack is effectively limited to 1024 descriptors.

PARAMETER STACK REGISTER (PSR)

Format: 72 bits

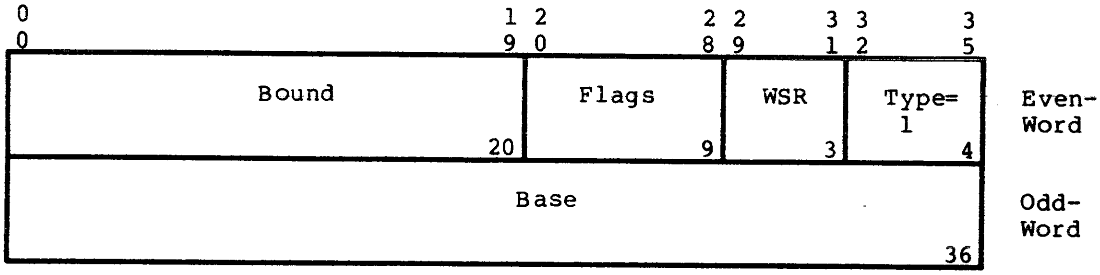


Figure 4-23. Parameter Stack Register (PSR) Format

Description:

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

Function:

Instructions are provided for loading (Privileged Master mode) and storing the parameter stack 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 (not empty) bits 17-19 are forced to 111. Thus, the size of the parameter stack is effectively limited to 1024 descriptors.

INSTRUCTION SEGMENT REGISTER (ISR)

Format: 72 bits

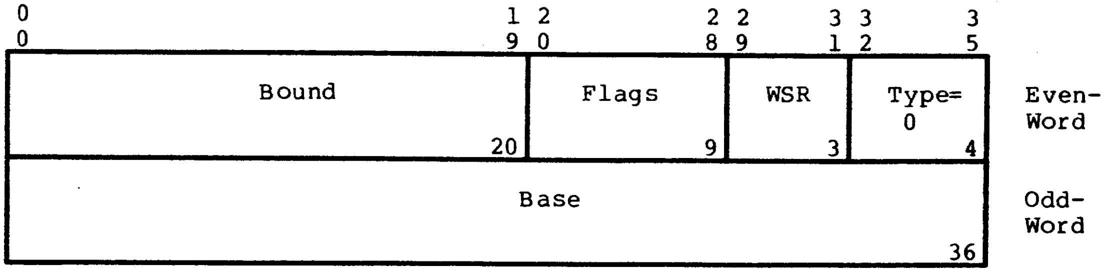


Figure 4-24. Instruction Segment Register (ISR) Format

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 operand 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 1s.

OPERAND DESCRIPTOR REGISTERS (DRn)

Format: 72 bits each

Description:

Eight 72-bit registers that hold operand 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 type fields; type fields 0, 2, 4, and 6 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 operand descriptor registers and for modifying their contents. An operand descriptor register is invoked for virtual operand address development when bit 29 of the instruction is 1, and address bits 0, 1, and 2 specify which the combined operand descriptor register (DRn) and address register n (ARn) is to be used. Each of these eight operand 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" documented later in this manual.

SEGMENT IDENTITY REGISTERS (SEGIDn)

Format: 12 bits each

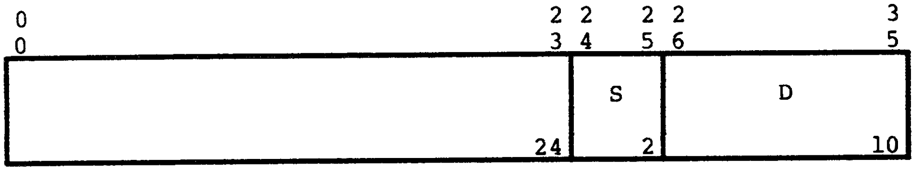


Figure 4-25. Segment Identity Register (SEGIDn) Format

Description:

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

Function:

The Load Pointer Register (LDPn) and Store Pointer (STPn) instructions are available for directly loading and storing the segment identity registers. The S and D field codes used in these registers indicate the origin of the descriptor (S = segment, D = descriptor offset).

When S = 0:

For D = 1760 through 1777 (octal), the descriptors identified by S, D were obtained from:

D = 1760	Undefined	
D = 1761	Undefined	
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 Stack Register (PSR)	
D = 1770	DR0, Descriptor Register 0	Self Identifying
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	

For D = 0000 through 1757 (octal), the descriptor in DRn was loaded from the parameter stack 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

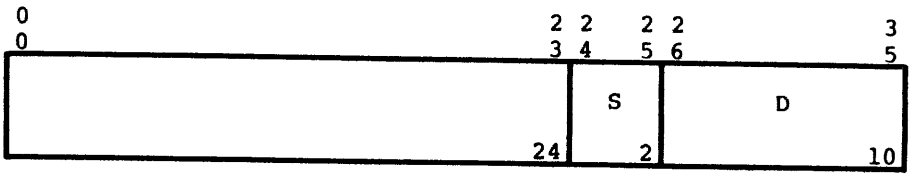


Figure 4-26. Instruction Segment Identity Register - SEGID (IS) Format

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 operand 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 codes.

POINTER REGISTERS (PRn)

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 operand descriptor register (DRn), segment identity register (SEGIDn), and address register (ARn) utilized as a collective register.

DATA STACK DESCRIPTOR REGISTER (DSDR)

Format: 72 bits

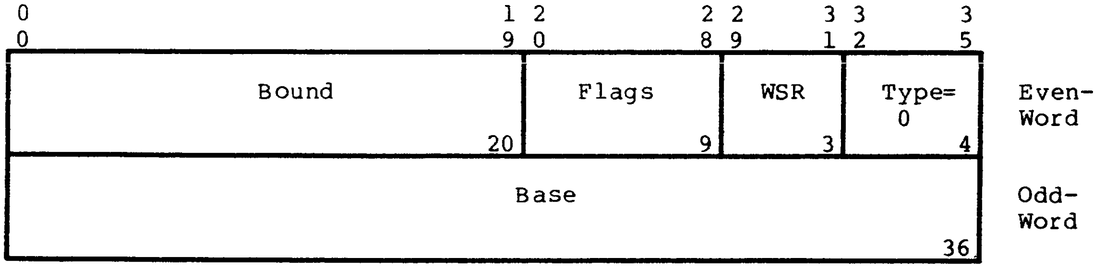


Figure 4-27. Data Stack Descriptor Register (DSDR) Format

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 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:

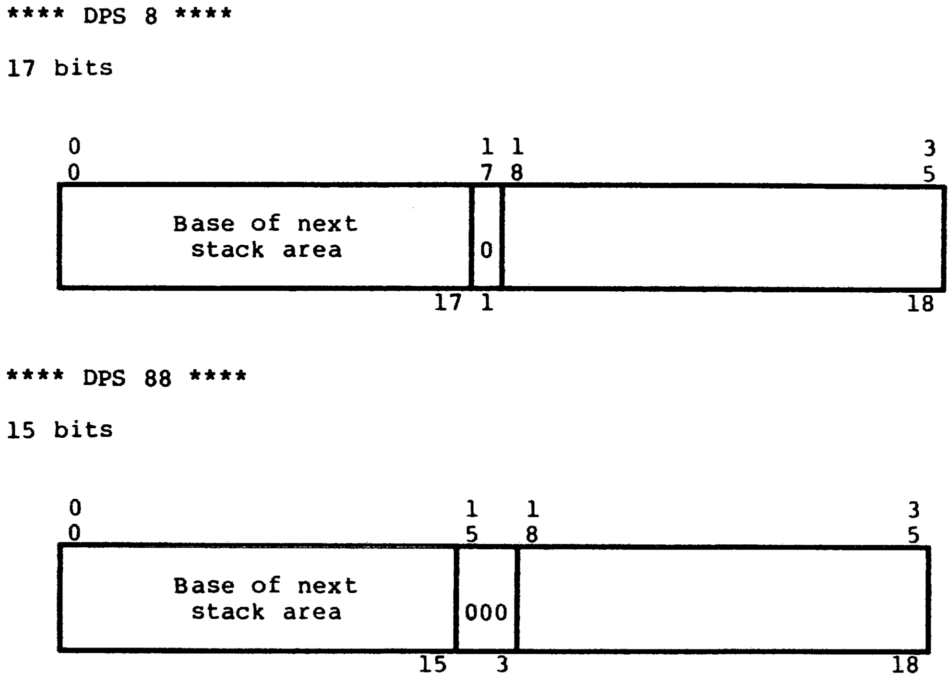


Figure 4-28. Data Stack Address Register (DSAR) Format

Description:

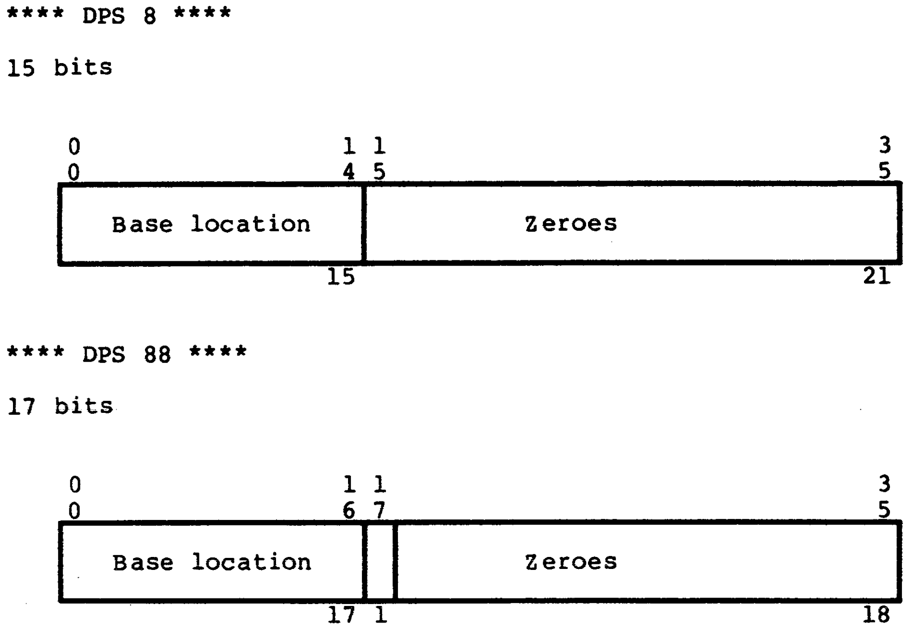
A 17-bit (DPS 88: 15-bit) special-purpose index register that points to the next available double-word (DPS 88: mod 8 word) location within the data stack area of memory framed by the Data Stack Descriptor Register (DSDR). Bit 17 (DPS 88: 15-17) is always zero.

Function:

Privileged Master mode instructions 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:



**** DPS 8 ****

15 bits

**** DPS 88 ****

17 bits

Figure 4-29. Page Directory Base Register (PDBR) Format

Description:

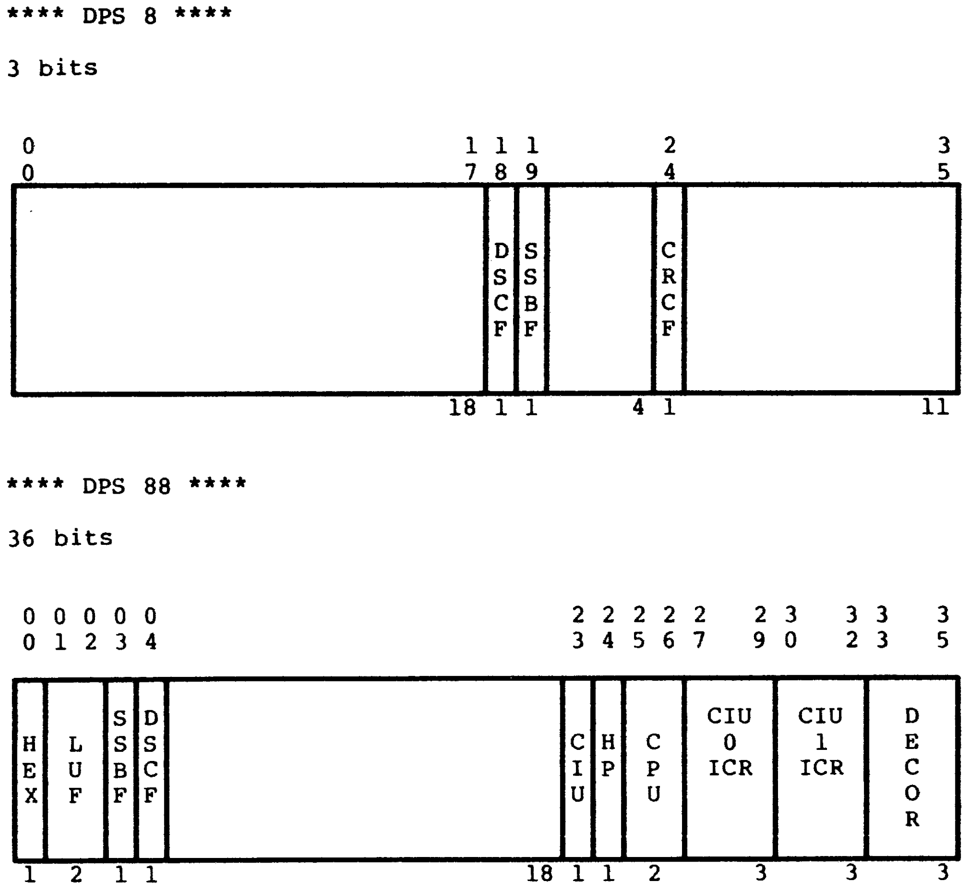
A 15-bit (DPS 88: 17-bit), modulo 512 word register that contains the base location of the working space page table directory.

Function:

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

OPTION REGISTER (OR)

Format:



**** DPS 8 ****

3 bits

**** DPS 88 ****

36 bits

Figure 4-30. Option Register (OR) Format

Description:

**** DPS 8: A 3-bit register located in the virtual unit that controls the clearing of data stack space, bypassing the safe store portion of an inward CLIMB (ICLIMB) instruction, and bypassing cache memory. Bit 18 is the Data Stack Clear Flag (DSCF), bit 19 is the Safe Store Bypass Flag (SSBF), and bit 24 is the Cache Read Control Flag (CRCF). ****

**** DPS 88: This 36-bit register controls various options in the CPU. Instructions are provided for loading (LDO, LDHC, LGCOS, LMSD, LVMS) and Storing (STO). ****

Function:

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

SECTION V

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.

Basic Modification

Address modification is performed in 4 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 a number of 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

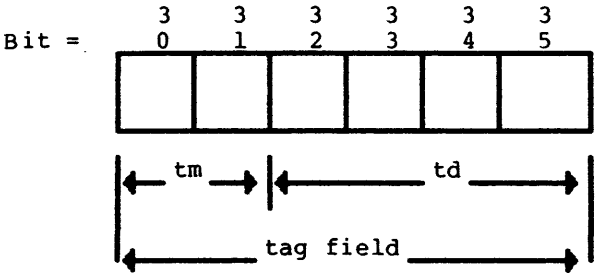
Generally, in indirect addressing, the content of bits 0-17 in the word addressed by the instruction address (y) is treated as another address, rather than as the operand of the instruction. Indirect address modification is performed by the hardware whenever called for by a program instruction. When 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), appear 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. In the case of tm = R, RI, or IR, td is called the register designator and generally specifies the register to be used in indexing.

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

The following table shows the valid 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	Coding Mnemonic	Binary Forms	Example
		30 31 32 35 tm | td	
R	BETA,(R)	30 31 32 35 0 0 | 1 1 0 1	BETA,5
RI	BETA,(R)*	30 31 32 35 0 1 | 1 0 1 0	BETA,2*
IR	BETA,*(R)	30 31 32 35 1 1 | 1 1 1 1	BETA,*7
IT	BETA,(T)	30 31 32 35 1 0 | 1 0 1 0	BETA,SC

The parentheses enclosing R and T indicate that substitutions are 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 modifier subfield tm of the general instruction. Accordingly, 1 of 16 variations under R will be performed by the processor, depending upon bit configurations generated by the assembler, and will be 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.

A type of address modification variation is provided under R modification. The use of the instruction address field as the operand is called direct operand address modification, of which there are two types: (1) Direct Upper (DU) and (2) 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 an 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:

```
                                          Effective
         1      8         16              Address
         ______________________________________________________
(1)             EAXO      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

(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.

- (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

o (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        With Address Modification

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

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:

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

Coding examples of IR modification follow:

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

Indirect Then Register and then Register or Indirect Then Tally

```
(IR) = *(Xn) where n = 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

Indirect Then Register and then Register Then Indirect

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

        EAX5    16

        EAX2    17

ALPHA  LDA     ADRES1,*5

ADRES1 LDQ     ADRES2,2*

ADRES2+17 LDA     ADRES4
```

the effective address is:

```
ADRES4+16
```

Example 4

Indirect Then Register and then Indirect Then Register

(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)	
(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)=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)	
(7)		EAX5	12		
		LDA	Z,*N	(IR)	Y=M+12
	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. 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.)

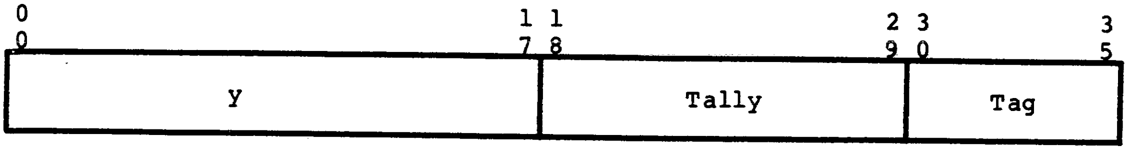
Variation	Mnemonic Substitution	Binary Form (td Field)	Effect on Processor and Indirect (Tally) Word for Each Reference
Fault	F	0000	None. The processor is forced to a fault trap. The indirect word is not examined.
Character indirect	CI	1000	None. Applies to TALLY, TALLYB.
Sequence character	SC	1010	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 reverse	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.

Variation	Mnemonic Substitution	Binary Form (td Field)	Effect on Processor and Indirect (Tally) Word for Each Reference
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.
Increment address, decrement tally, and continue	IDC	1111	Obtain the operand address from the tally word and then 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.
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.
Add delta	AD	1011	Obtain the operand address from the tally word and then 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:



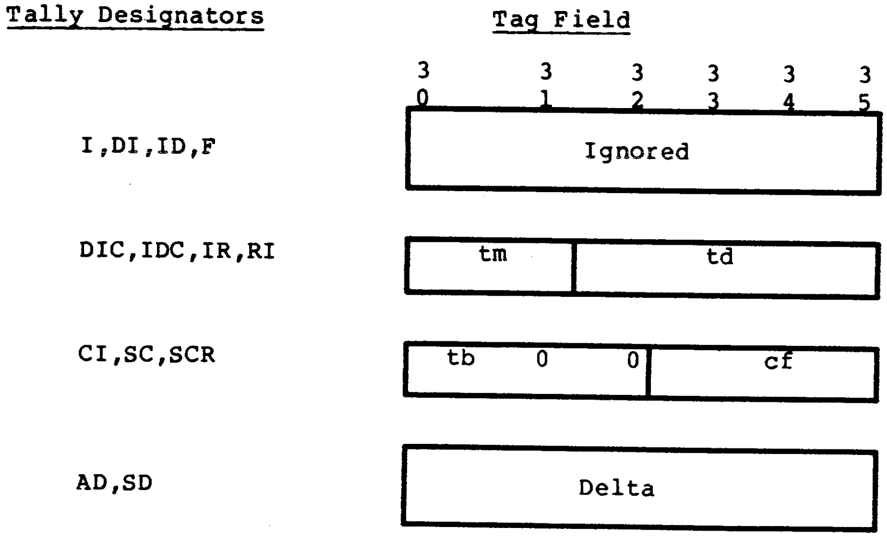
where:

y = address field

Tally = tally field

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. (This will usually indicate some abnormal condition for which the user desires protection.)

Character Indirect (T) = CI Variation. The Character Indirect (CI) variation is provided for operations on 6-bit or 9-bit characters in any situation 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 substitution, 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. Some examples are:

1	8	16	Modification Type	Effective Address	Character Position
	LDA	Z,CI	(IT)	Y=B	4
Z	TALLY	B,,4			

1	8	16
	LDA	ADDR,CI
ADDR	TALLY	ADD,,3
	or	
ADDR	TALLYB	ADD,,3

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 programmed operations of 6-bit or 9-bit characters that are accessed sequentially in memory. The character size indicator (tb) of the indirect word is used to specify the character size. Processor instructions that exclude character operations are so indicated in the individual instruction descriptions. For the SC variation, the effective operand address is the address field of the indirect word obtained via the tentative operand address of the instruction or preceding indirect word that specified the SC variation.

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 to be involved in the operation. The Tally Runout indicator is set when the tally field of the indirect word reaches 0. The following is an example of the coding:

```
1       8       16
_____________________________________
        LDA     A,SC

A       TALLY   TABLE,70,4
TABLE   BSS     13
```

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

For register loads under 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. 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:

```
                                Effective   Character
1       8       16              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 tally occurs. 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. For example:

1	8	16	Modification Type	Effective 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.

The following examples show the effect of ID:

1	8	16	Modification Type	Effective Address	Reference
	LDA	Z,ID	(IT)	B	1
Z	TALLY	B,12		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
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 one specified the DI variation. Each time a 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. The incrementing and decrementing are performed prior to providing the effective address for the current instruction operation.

The effect of DI is shown in the following examples:

```
                            Modification   Effective
1        8        16            Type        Address     Reference

         LDA      Z,DI          (IT)          B-1          1
Z        TALLY    B,-18                       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
ADRES1 LDA     ADRES2,DI

       TTF     ADRES1
.
.
ADRES2 TALLY   ADRES3,-10
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 gives the sequential stepping 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 form of IT or IR; but if RI or R is used, R must equal N.

The effect of successive references using IDC modification is indicated in the following examples:

```
                              Effective
1      8       16              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

ADRES1 LDA     ADRES2,IDC
       TTF     ADRES1

ADRES2 TALLYC  ADRES3,4,*
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 allows 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:

```
                                  Modification   Effective
1        8         16                 Type        Address     Reference
---------------------------------------------------------------------------
         LDA       Z,DIC              (IT)

                   B,-10,I
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.

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

ADRES1   LDA       ADRES2,DIC
         TTF       ADRES1

ADRES2   TALLYC    ADRES3,-4,*N
         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 Type	Effective Address	Reference
	LDAQ	Z,AD	(IT)	B	1
Z	ETALLY	B,20,2		B+2	2
B	EBSS	40		B+4	3
				.	.
				.	.
				B+2*n*	*n*+1
The Tally Runout indicator is set on the 20th reference.				.	.
				.	.

```
1        78          16
ADRES1    LDAQ       ADRES2,AD
          TTF        ADRES1
           .
           .
ADRES2   ETALLYD     ADRES3,10,2

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 the indirect word is used. 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

LOW ORDER OCTAL DIGIT

HIGH 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 5-1. Address register modification is not included in this example.

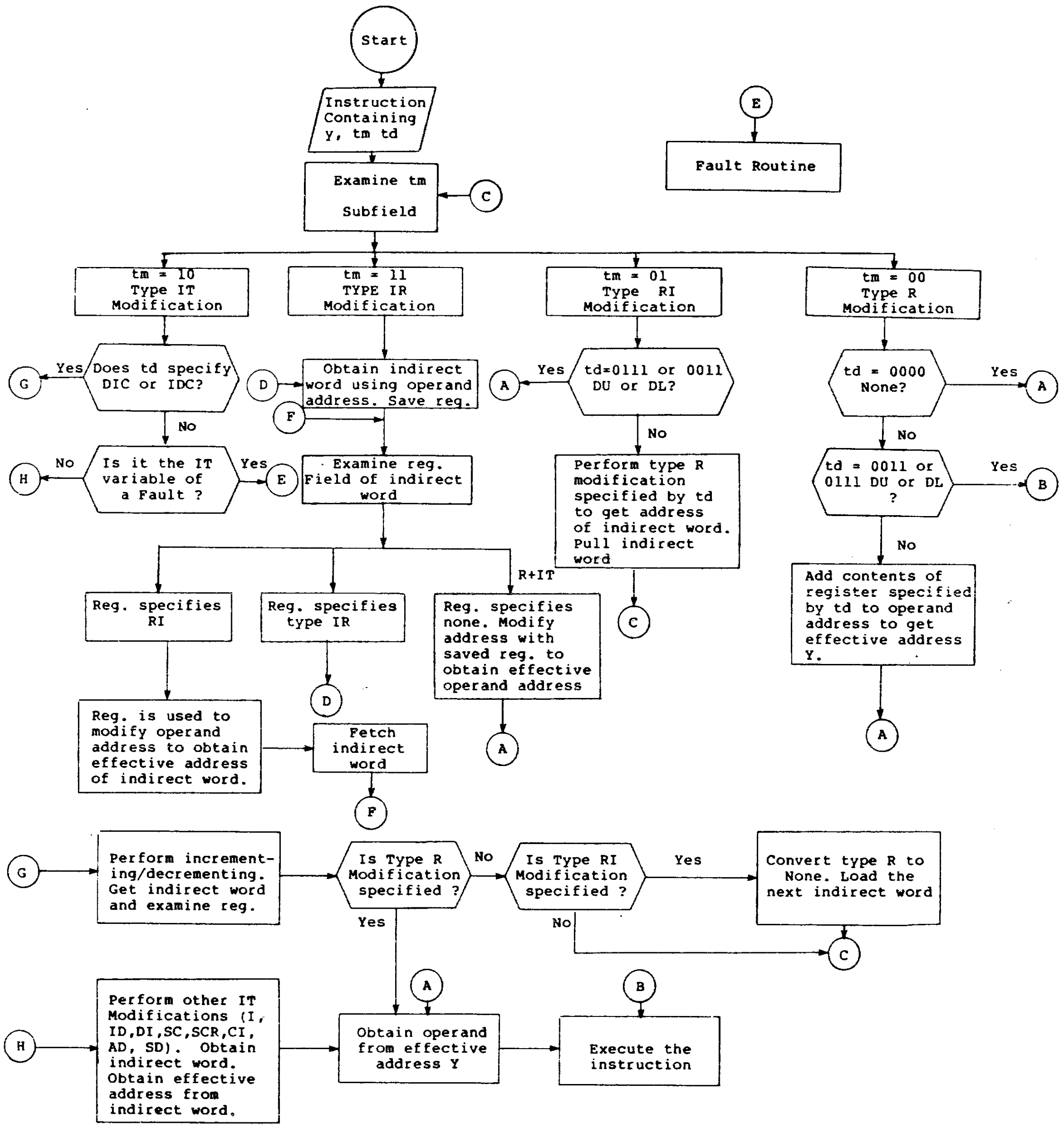

Start
Instruction Containing y, tm td
Examine tm Subfield
C
E
Fault Routine
tm = 10 Type IT Modification
tm = 11 TYPE IR Modification
tm = 01 Type RI Modification
tm = 00 Type R Modification
G
Yes
Does td specify DIC or IDC?
No
D
Obtain indirect word using operand address. Save reg.
F
A
Yes
td=0111 or 0011 DU or DL?
No
td = 0000 None?
Yes
A
No
H
No
Is it the IT variable of a Fault ?
Yes
E
Examine reg. Field of indirect word
Perform type R modification specified by td to get address of indirect word. Pull indirect word
td = 0011 or 0111 DU or DL ?
Yes
B
No
Add contents of register specified by td to operand address to get effective address Y.
R+IT
Reg. specifies RI
Reg. specifies type IR
Reg. specifies none. Modify address with saved reg. to obtain effective operand address
C
D
A
A
Reg. is used to modify operand address to obtain effective address of indirect word.
Fetch indirect word
F
G
Perform incrementing/decrementing. Get indirect word and examine reg.
Is Type R Modification specified ?
No
Is Type RI Modification specified ?
Yes
Convert type R to None. Load the next indirect word
Yes
No
C
A
B
H
Perform other IT Modifications (I, ID,DI,SC,SCR,CI, AD, SD). Obtain indirect word. Obtain effective address from indirect word.
Obtain operand from effective address Y
Execute the instruction

Figure 5-1. Address Modification Flow Chart

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:

00 02	03	04 17	18 27	28	29	30 31	32 35
AR#	S	y	OP CODE	I	AR	TAG	
						Tm	Td

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. 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.

NOTE: Address register modification is illegal (DPS 88, DPS 8/20 and 8/44: legal) for instructions executed under control of RPT, RPD, and RPL instructions. Address register modification is ignored in an indirect word in a multilevel indirection condition.

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

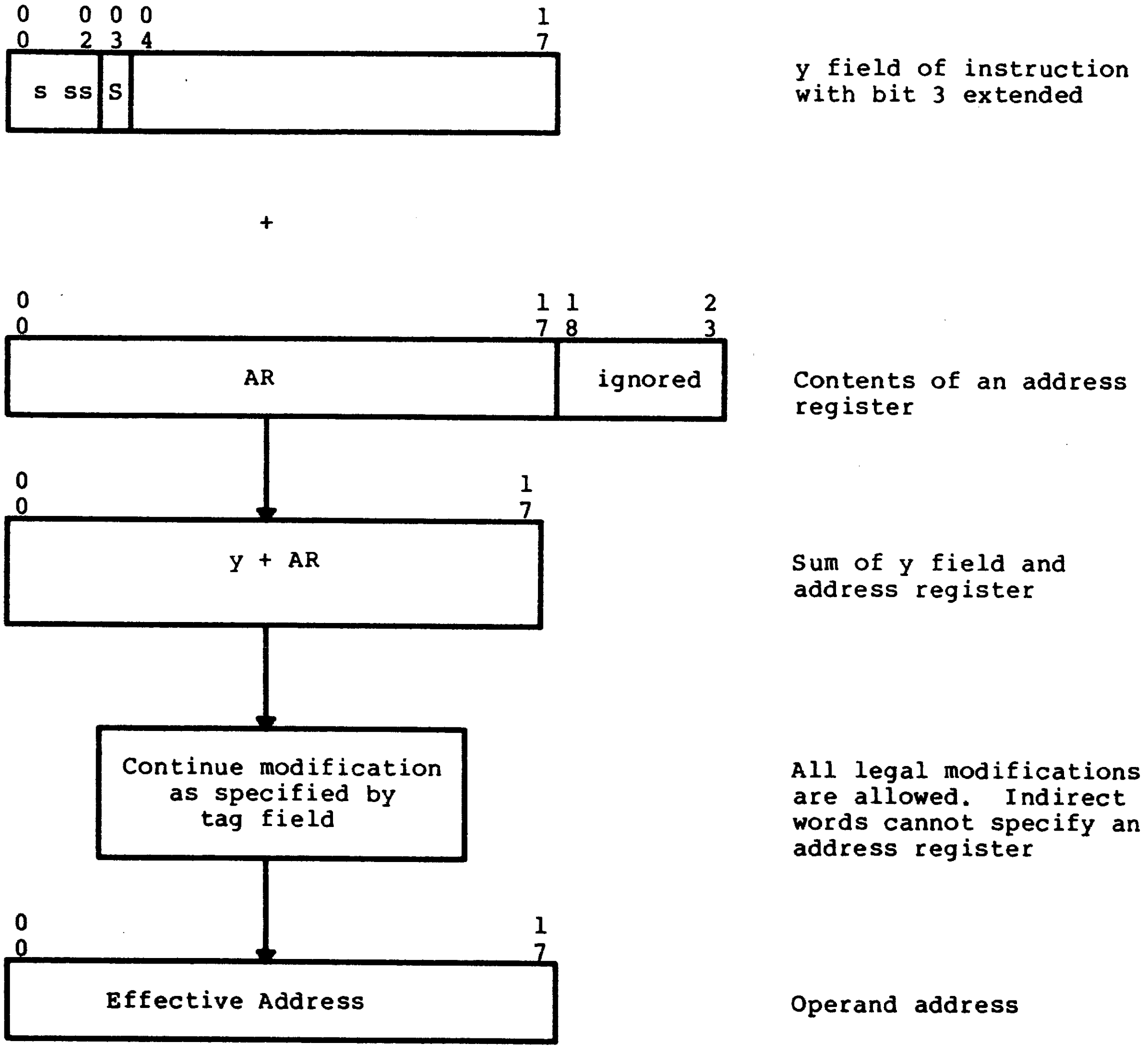
1. The three most significant bits of y (0, 1, 2) are decoded to determine which of the eight address registers is to be used.

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

3. The twos 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.

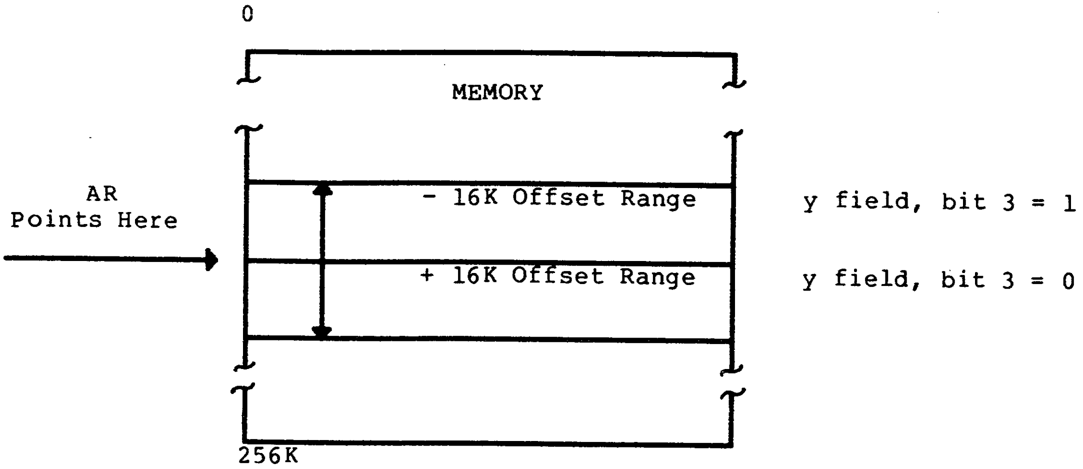
4. Address modification continues as specified by the tag field of the instruction word.

Diagramatically, address preparation is described below:


0 0
0 2
0 3
0 4
1 7
s ss
S
y field of instruction with bit 3 extended
+
0 0
1 7
1 8
2 3
AR
ignored
Contents of an address register
0 0
1 7
y + AR
Sum of y field and address register
Continue modification as specified by tag field
All legal modifications are allowed. Indirect words cannot specify an address register
0 0
1 7
Effective Address
Operand address

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 twos 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 + bits 0-17 of C(AR2)

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 as follows:

Memory Loc.	00 17	18 27	28	29 35	
0	Variable Field	OP CODE	I	MF1	Instruction Word
1	Operand Descriptor 1 or Indirect Word				Descriptor 1
2	Operand Descriptor 2 or Indirect Word				Descriptor 2
3	Operand Descriptor 3 or Indirect Word				Descriptor 3

where:

Variable Field - Contains additional information concerning the operation to be performed, depending on the particular instruction.

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. 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. Exceptions to this are the MVT, TCT, and TCTR 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	RL	ID	REG	← subfield
1	1	1	4	← number of bits

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.

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 follows.

Table 5-2. Register Codes

Octal Code	R Type (MF Field)	REG (MF Field) (See Note 1)	Operand Descriptor N (32-35) If RL = 1 (See Note 2)
00	No Register(N)	No Register(N)	Illegal (causes IPR)
01	AU	AU	AU
02	QU	QU	QU
03	DU	Illegal (causes IPR)	Illegal (causes IPR)
04	IC	IC	Illegal (causes IPR)
05	AL	A	A
06	QL	Q	Q
07	DL	Illegal (causes IPR)	Illegal (causes IPR)
10	0	0	X0
11	1	1	X1
12	2	2	X2
13	3	3	X3
14	4	4	X4
15	5	5	X5
16	6	6	X6
17	7	7	X7

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 follow the multiword instruction immediately - 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.

NOTE 1 (When used as a REF field of an indirect operand descriptor)

When the REG field of an indirect word contains one of the register codes, the specified register contents are interpreted as a word index (see "Indirect Word" later in this section).

When the REG field of the modification field contains one of the register codes, the designated 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.

The A- and Q-registers provide for indexing by a number greater than 2**18-1. When one of these registers 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

20 bits for 9-bit character operations

21 bits for 6-bit and 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 character	4	4 x 2^{18} -N	(2^{20} -N)
4-bit character	8	8 x 2^{18} -N	(2^{21} -N)
6-bit character	6	6 x 2^{18} -N	
1 bit	36	36 x 2^{18} -N	

Since the modulo addressing for 9- and 4-bit characters is a power of 2 (2^{20} and 2^{21} respectively) and the hardware ignores the remaining high-order bits, the A and Q can be loaded with a -N directly. For 1-bit and 6-bit characters, A and Q can be respectively loaded with 36,DU and 6,DU and N can then be subtracted.

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.

NOTE 2 (When used as a register designator in a descriptor)

Except in the cases of A and Q, when a string length is contained in a register, the full 18 bits is interpreted as the length. Lengths in A or Q utilize the same number of bits as stated in Note 1 above for the REG field of a modification field (MF).

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 follows:

BIT STRING OPERAND DESCRIPTOR

00 02	03 17	18 19	20 23	24 35
	y	c	b	N

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

BDSC - Bit descriptor

1	8	16
	BDSC	LOCSYM,N,c,b,AM

ALPHANUMERIC OPERAND DESCRIPTORS

00 02	03 17	18 20	21 22	23	24 35
	Y	CN	TA	0	N

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.

NUMERIC OPERAND DESCRIPTORS

00 02	03 17	18 20	21	22 23	24 29	30 35
	Y	CN	TN	S or SX	SF	N

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 = original 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; 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 = original character position within a word of 9-bit characters.

Code	Char.
00	0
01	1
10	2
11	3

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

Code	Bit	Code	Bit
0000	0	0101	5
0001	1	0110	6
0010	2	0111	7
0011	3	1000	8
0100	4		

All other combinations of these 4 bits are illegal codes and will cause an IPR fault.

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.

CN = original character number within the data word specified by the original data word address. Code for the CN depends on the data type as shown below. Coding entry is by 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).

S 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 twos 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 six 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.

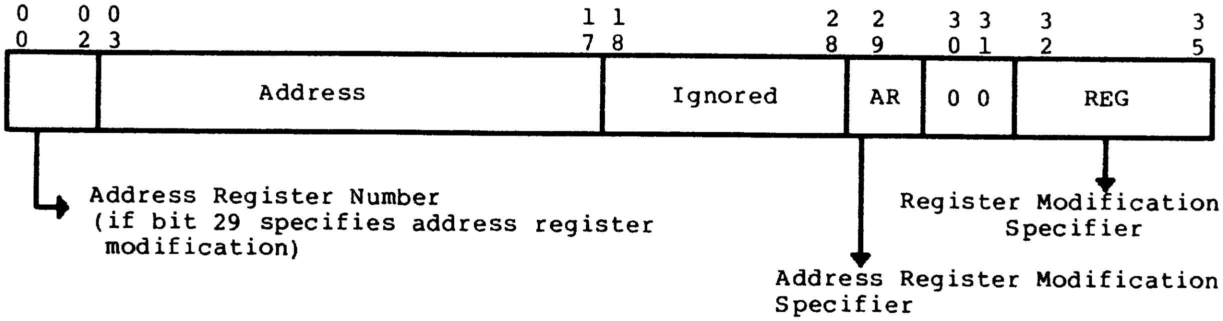
This field is ignored if S = 00.

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

AM = address register modification

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 as follows:



0 0
0 2
0 3
1 7
1 8
2 8
2 9
3 0
3 1
3 2
3 5
Address
Ignored
AR
0 0
REG
Address Register Number (if bit 29 specifies address register modification)
Register Modification Specifier
Address Register Modification Specifier

The AR and REG fields are identical in function with 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 (7,,4)
```

OPERAND DESCRIPTOR ADDRESS PREPARATION

A flowchart of the operations involved in operand descriptor address preparation is shown in Figure 5-2. 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 one 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.

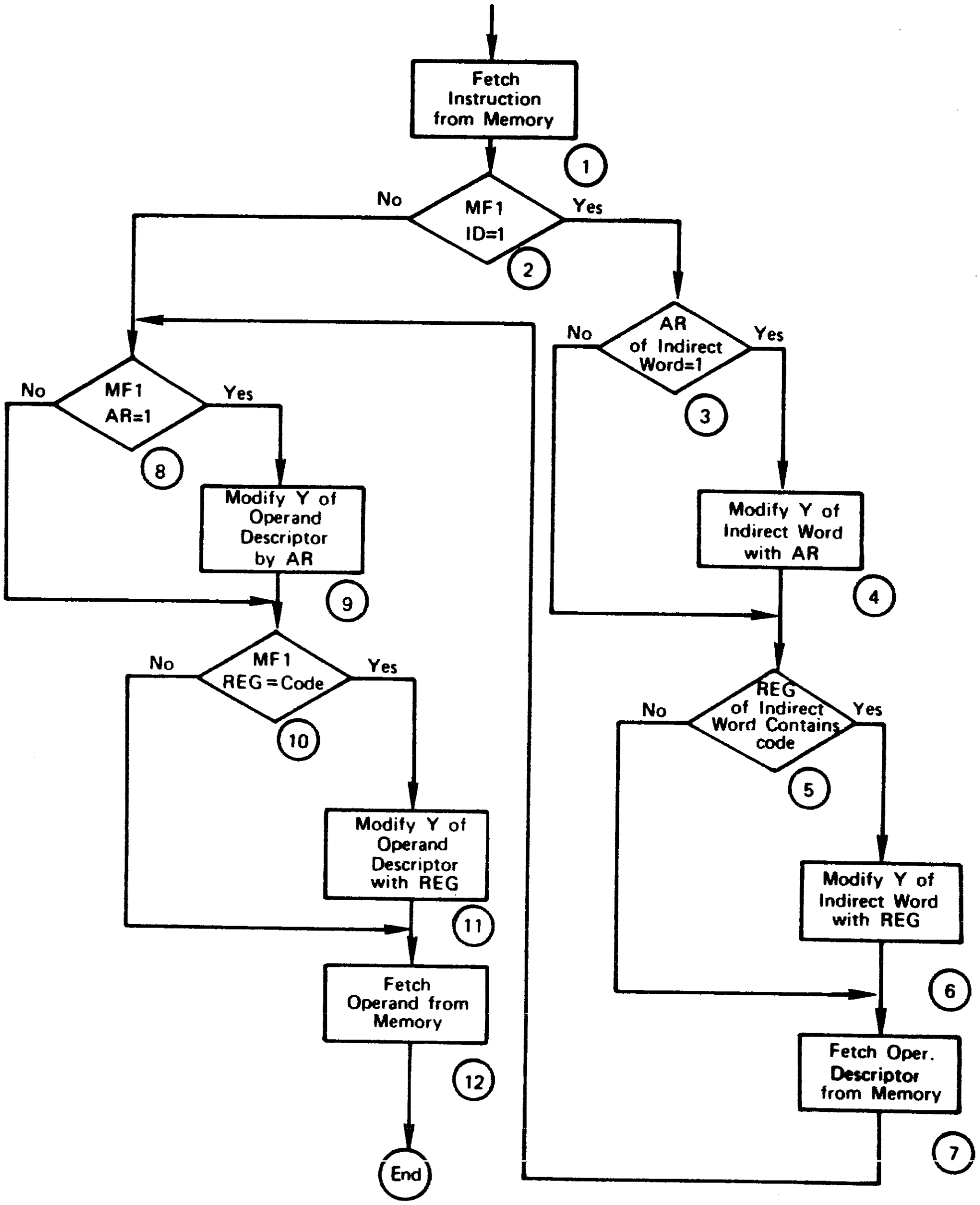
⑤ A check is now made to determine if the REG field of the indirect word specifies that a register type modification be performed.

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 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.

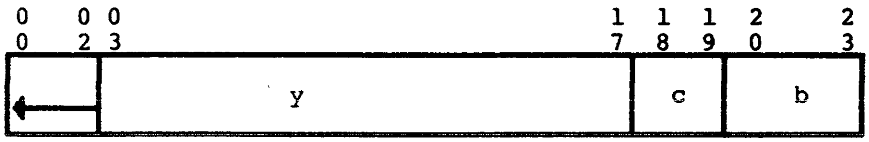
⑨ Address register modification is performed on the operand descriptor as described under "Address Modification with Address Registers" above except that the character and bit positions of the specified address register are not ignored. Rather, they are used in one of two ways depending upon the type of operand descriptor; i.e., whether the type is a bit string descriptor or a numeric or alphanumeric descriptor.



Fetch Instruction from Memory
1
MF1 ID=1
No
Yes
2
AR of Indirect Word=1
No
Yes
3
MF1 AR=1
No
Yes
8
Modify Y of Operand Descriptor by AR
9
Modify Y of Indirect Word with AR
4
MF1 REG=Code
No
Yes
10
REG of Indirect Word Contains code
No
Yes
5
Modify Y of Operand Descriptor with REG
11
Modify Y of Indirect Word with REG
6
Fetch Operand from Memory
12
Fetch Oper. Descriptor from Memory
7
End

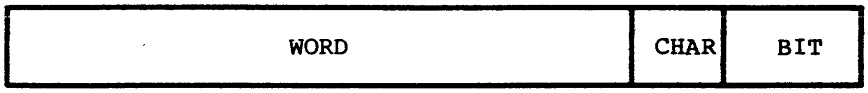
Figure 5-2. Flowchart For Operand Descriptor Address Preparation

Bit String Address Preparation



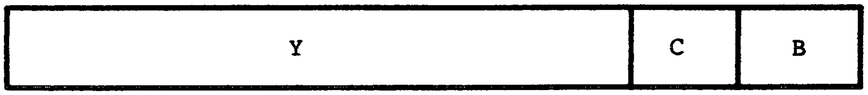
y, c, and b fields of descriptor with bit 3 of y extended

+



contents of address register specified by bits 0, 1, 2 of y

yields



modified descriptor address

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:

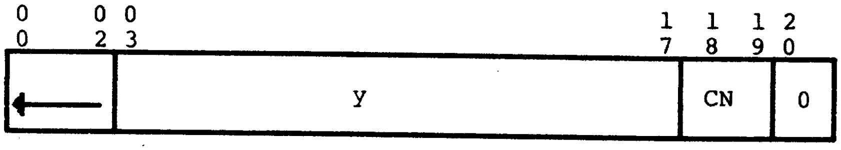
```
          BIT =  7
            b =  5
      BIT + b = 12, carry 1 to C and B = 12 -9 = 3
```

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
```

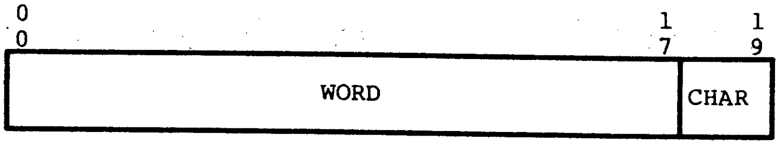
Alphanumeric/Numeric Address Preparation

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:



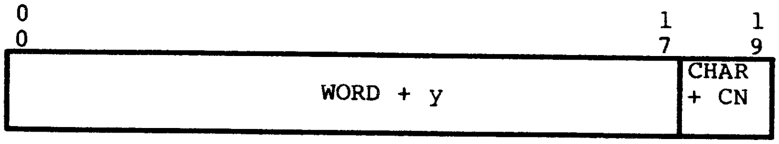
y and CN fields of the numeric or alpha numeric descriptor, bit 3 extended

+



contents of WORD and 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:

$C = 2\ (\text{CHAR}) + [\,(\text{BIT} + 4)/9 \text{ truncated}\ ]$

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:

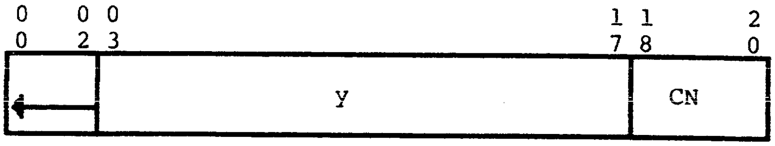
$$C = \frac{9\ (\text{CHAR}) + \text{BIT}}{6} \quad \text{(truncated)}$$

If CHAR = 3 and BIT = 7,

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

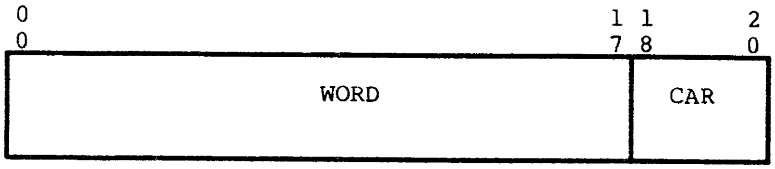
The remainder of 4 which represents the bit position within character position 5 is ignored. This means forcing the address register to point to the next lower character boundary.

The address modification can now take place.



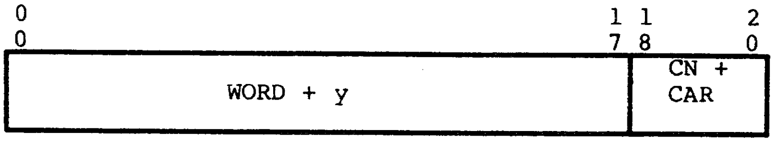
y and CN fields of the numeric or alphanumeric descriptor, bit 3 extended

+



contents of WORD position of address register designated by bits 0, 1, 2 of y; CAR is the character location translated from CHAR and BIT of address register

yields

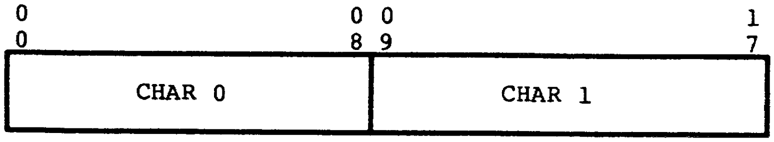


For 4-bit character mode, if CN + CAR is greater than 7, a carry is generated to WORD + y and CN + CAR = (CN + CAR) -8.

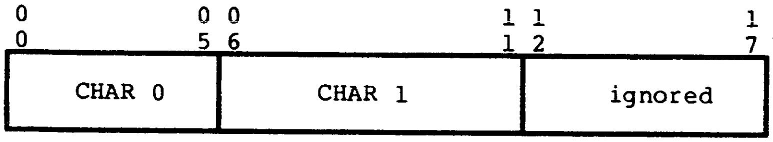
For 6-bit character mode, a carry is generated to WORD + y when CN + CAR is greater than 5 and CN + CAR = (CN + CAR) -6.

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, 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:

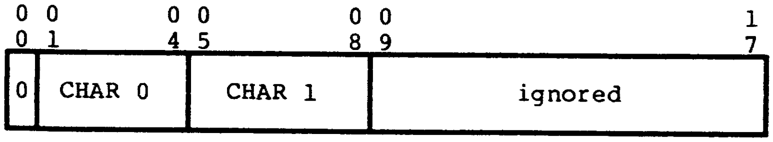
Operand descriptor word address field (y) Character type (TA)



9-bit characters



6-bit characters



4-bit characters

In the cases where only one character is involved (SCM, SCMR), only character 0 is used.

⑪ 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

00 – 17	18 – 19	20 – 23	
WD	CC	BC	converted bit count

+

00 – 17	18 – 19	20 – 23	
ym	cm	bm	modified bit descriptor operand address

yields YCB:

00 – 17	18 – 19	20 – 23	
WD + ym	CC+ cm	BC+ bm	effective bit address

Carries may occur from (BC + bm) to (CC + cm) and from (CC + cm) to (WD + ym) as described in ⑨.

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 twos 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:

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

   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.

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

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 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 converting a virtual memory address to a real memory address. 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, and the total address space is limited to 2**26 (DPS 88: 2**28) bytes.

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.

a. Working Spaces (WS)

The 2**43 virtual memory space is first divided into 512 working spaces. Each WS is 2**34 bytes in size. The WS number to be used in generating a particular virtual memory address is contained either in one of the eight working space registers (WSRs) or in the descriptor register being used.

b. Segments

A segment is part of a working space and may be as small as one byte or as large as four working spaces (2**36 bytes). Thus, unlike the fixed size of a WS, a segment size is variable. Segments are described by two 72-bit data items called descriptors.

When used in virtual address generation, 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 registers. The area of virtual memory constituting a segment is "framed" by the segment descriptor by defining a base value relative to the WS and a bound value relative to the base.

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

OPERAND ADDRESS PROCEDURE

The first phase of operand address development proceeds as follows: The effective address (EA) of the operand is formed. The EA is defined as the address that is formed after all register modification and indirection have been completed and is either an 18-bit (word), 20-bit (byte), or 24-bit (bit) address, depending upon the instruction.

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 one, 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 and bound 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 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

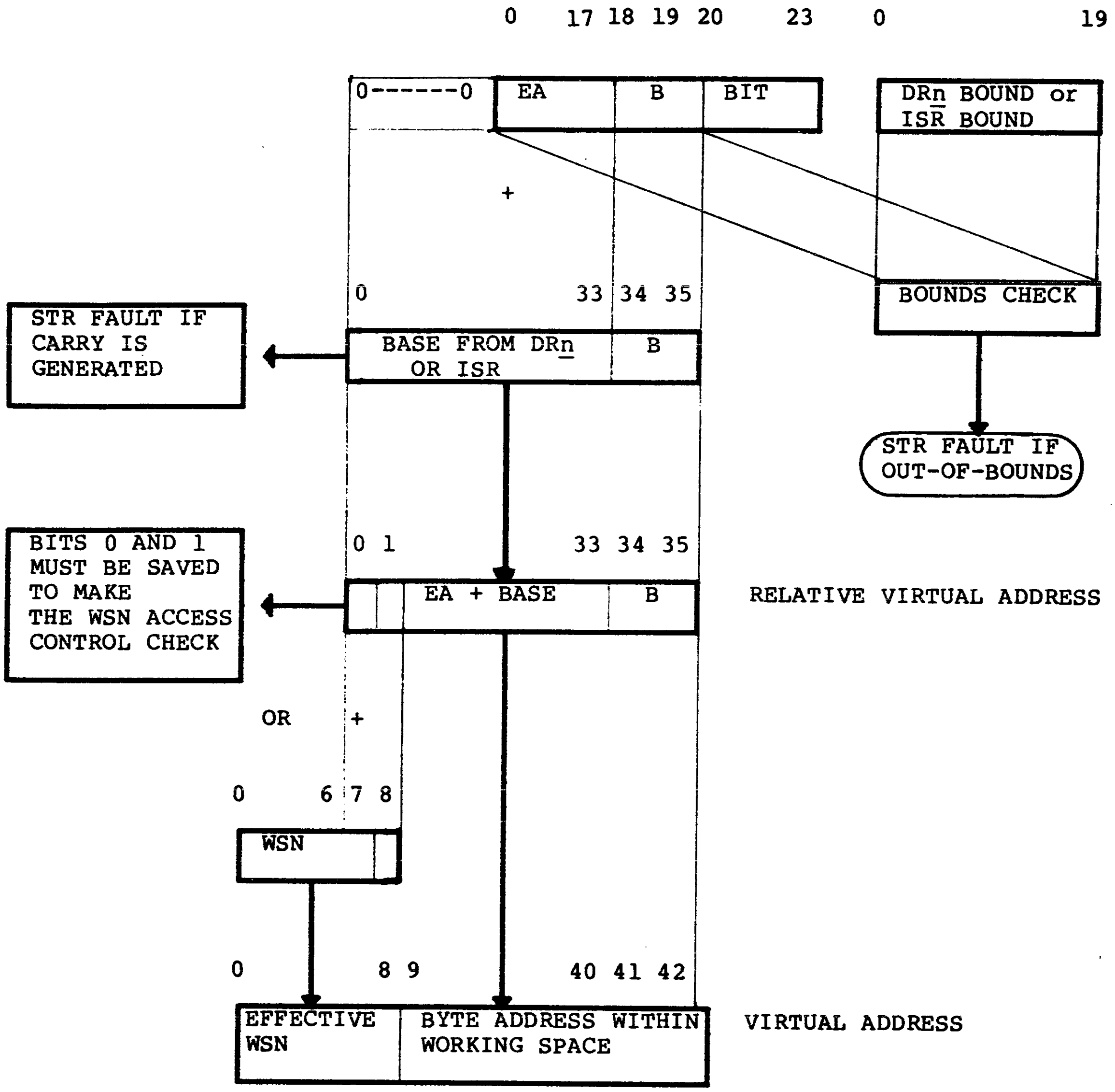
Virtual addresses for instructions are always formed using the value in the instruction counter (IC) and the base, bound, and WS the ISR.

Virtual Address Generation

The mechanics of generating the virtual memory address depend on whether the involved segment descriptor is a standard descriptor or a super-descriptor. For all memory accesses, a virtual address must be generated. 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).

STANDARD DESCRIPTOR

The method of forming an operand virtual address with a standard descriptor is shown in Figure 5-3. If instruction bit 29 = 0, the ISR is used; if bit 29 = 1, then DRn is used.



where: B - byte
WNS - working space number

Figure 5-3. Virtual Address Generation Using Standard Descriptor

The bound check is applied to the effective address at the byte level. The bound check is shown for byte or bit instruction; 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 an STR or BND fault.

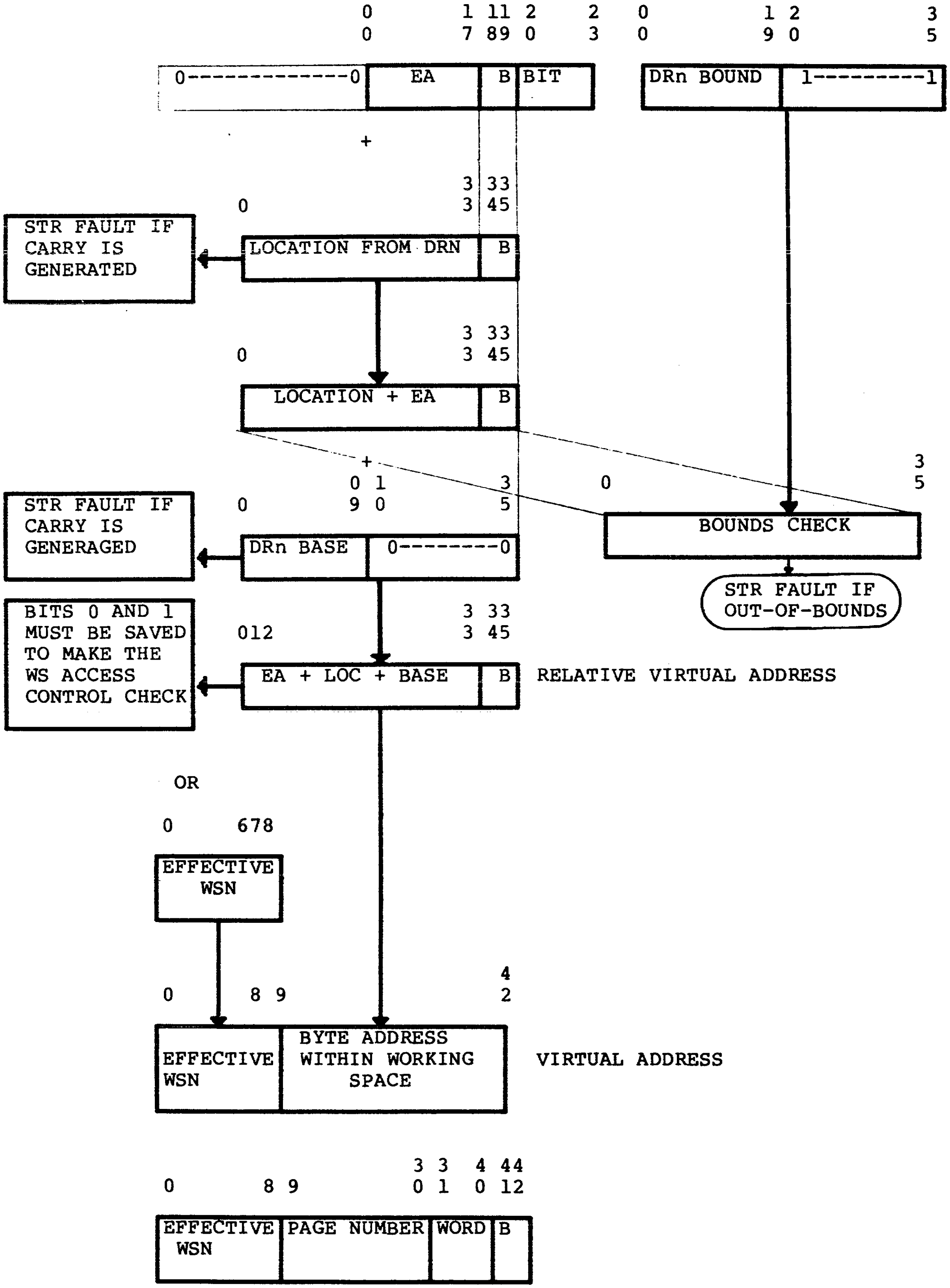
The effective WSN is formed by ORing the low-order two bits of the working space number with bits 0 and 1 of the sum of EA + BASE.

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

SUPER-DESCRIPTOR

**** DPS 8 ****

The method of forming an operand virtual address with a super-descriptor is shown in Figure 5-4.



WSN = WORKING SPACE NUMBER
B = BYTE

Figure 5-4. Virtual Address Generation Using Super-Descriptor

**** DPS 88 ****

The processor does not use the super-descriptor directly for address generation. Instead, each time a DRn is loaded with a super-descriptor, or each time the LDEAn instruction is executed, the processor generates a standard descriptor from the super descriptor and holds this generated descriptor in a temporary working register. Then, any time a DRn containing a super descriptor is referenced for address generation, the processor uses the standard descriptor previously generated.

The above procedure is transparent to software, and improves processor efficiency when super-descriptors are used. Any software operation (such as copy to another DR or store in memory) with a super-descriptor contained in a DRn is performed using the super-descriptor, not the generated standard descriptor.

The following steps describe how the processor generates a standard descriptor from a super-descriptor:

1. Base for standard descriptor is formed as shown in Figure 5-5. If a carry occurs, flag bit 27 of the formed descriptor is forced to zero (empty). Thus, any attempt to generate an address using the formed standard descriptor will result in a BND fault.

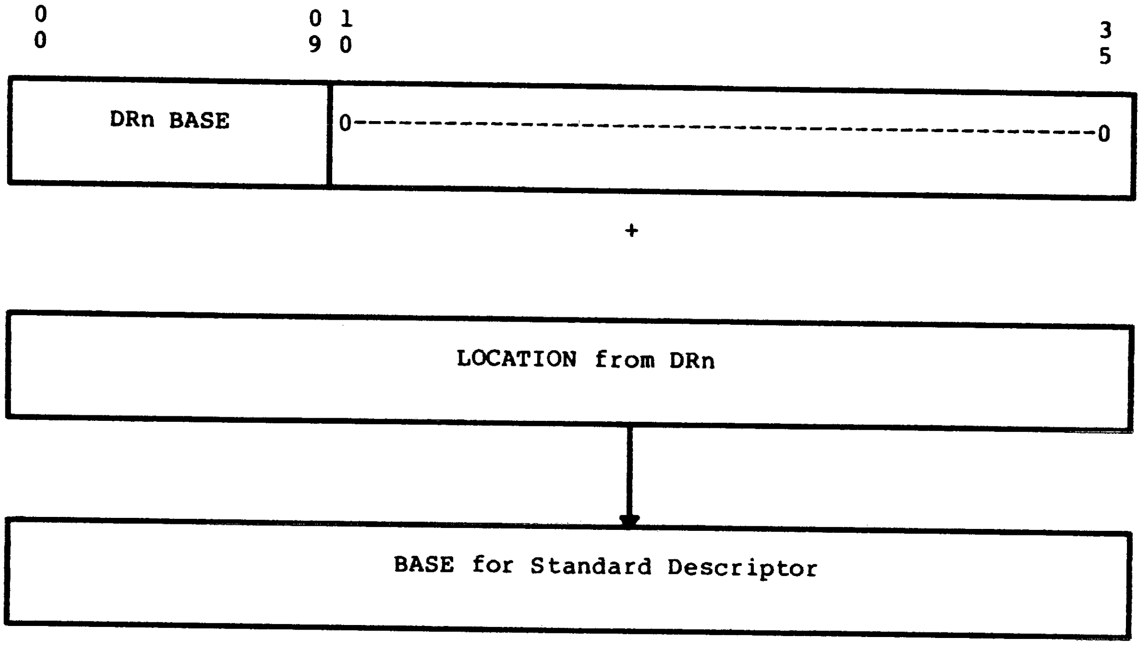


Figure 5-5. BASE For Standard Descriptor (DPS 88)

2. Bound for standard descriptor is formed as shown in Figure 5-6.

   - If resulting bits 0-15 are zero, bits 16-35 become the 20-bit bound field.
   - If resulting bits 0-15 are not zero, the 20-bit bound field of the standard descriptor is forced to all ones.
   - If a borrow occurs in the above operation, flag bit 27 of the formed descriptor is forced to zero (empty). Thus any attempt to access the segment using the formed standard descriptor will result in a BND fault.

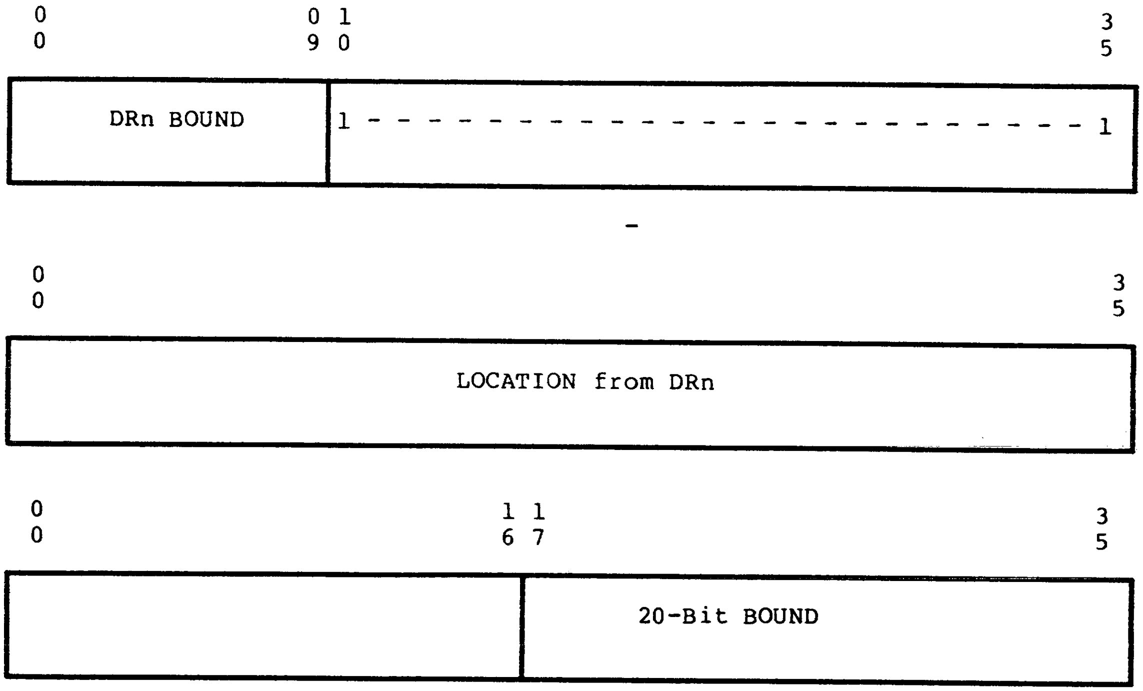


Figure 5-6. BOUNDS For Standard Descriptor (DPS 88)

When a T = 6 descriptor is loaded into a DRn register, a "standardized" descriptor is formed. If this standardized descriptor is to be marked "empty", i.e., bit 27 = 0, the instruction loading the DRn will terminate with a BND fault. This action is required since T = 2, 3, 6 descriptors are assumed to have bit 27 = 1.

Absolute Addressing Mode

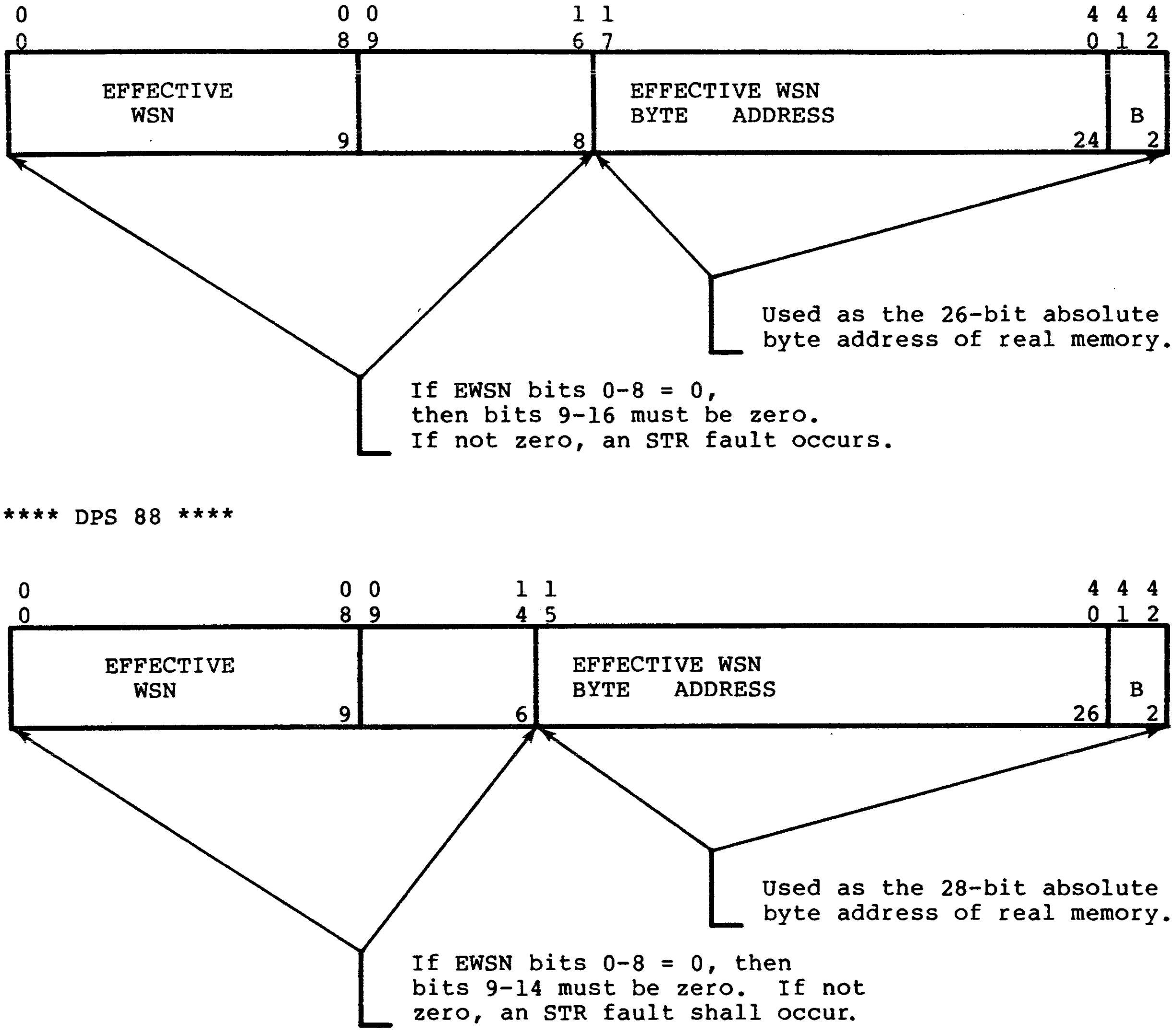
Virtual memory provides an absolute addressing mode. When the processor utilizes the absolute addressing mode, the virtual address is generated as previously described. However, the virtual address is not mapped to a real address; it is used as the real address but with a maximum size limitation of 2**26 (DPS 88: 2**28) bytes.

The processor utilizes the absolute addressing mode each time working space number zero is referenced. For example, assume that the descriptor contained in the instruction segment register (ISR) points to working space register 1, containing zeros; that the instruction refers to DR2, that points to WSR 3; and that WSR 3 contains 20. Then, the instruction and operands with bit 29 OFF would be accessed in the absolute addressing mode, and operands referenced with bit 29 ON and the DR2 selected would be accessed in the virtual addressing mode from working space 20 (assuming bits 0-1 of the resulting virtual address = 00).

To utilize the absolute addressing mode, the processor must be in Privileged Master mode. The master mode bit in the indicator register and the privileged bit of the segment descriptor must be ON. If these two conditions are not met, an attempted reference to working space zero in Master or Slave mode causes a Command fault. The housekeeping bit is assumed ON when working space zero is referenced.

When the processor utilizes the absolute addressing mode, address preparation proceeds as in normal virtual address development. After the resulting virtual address has been generated and bound checks have been made, the processor performs the checks indicated below.

**** DPS 8 ****



Used as the 26-bit absolute byte address of real memory.

If EWSN bits 0-8 = 0, then bits 9-16 must be zero. If not zero, an STR fault occurs.

**** DPS 88 ****

Used as the 28-bit absolute byte address of real memory.

If EWSN bits 0-8 = 0, then bits 9-14 must be zero. If not zero, an STR fault shall occur.

Figure 5-7. Resulting Virtual Address Check

Paging Addressing Mode

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 a page table. The location of the page table supporting a particular WSN is found by using the 9-bit WSN to index a 512-word table that contains the supporting page table directory words. This 512-word table is called the working space page table directory (WSPTD). This table is located in real memory by a special register called the page directory base register (PDBR).

PAGE TABLE DIRECTORY WORD FORMAT

The format of the page table directory word (PTDW) is given below.

**** DPS 8 ****

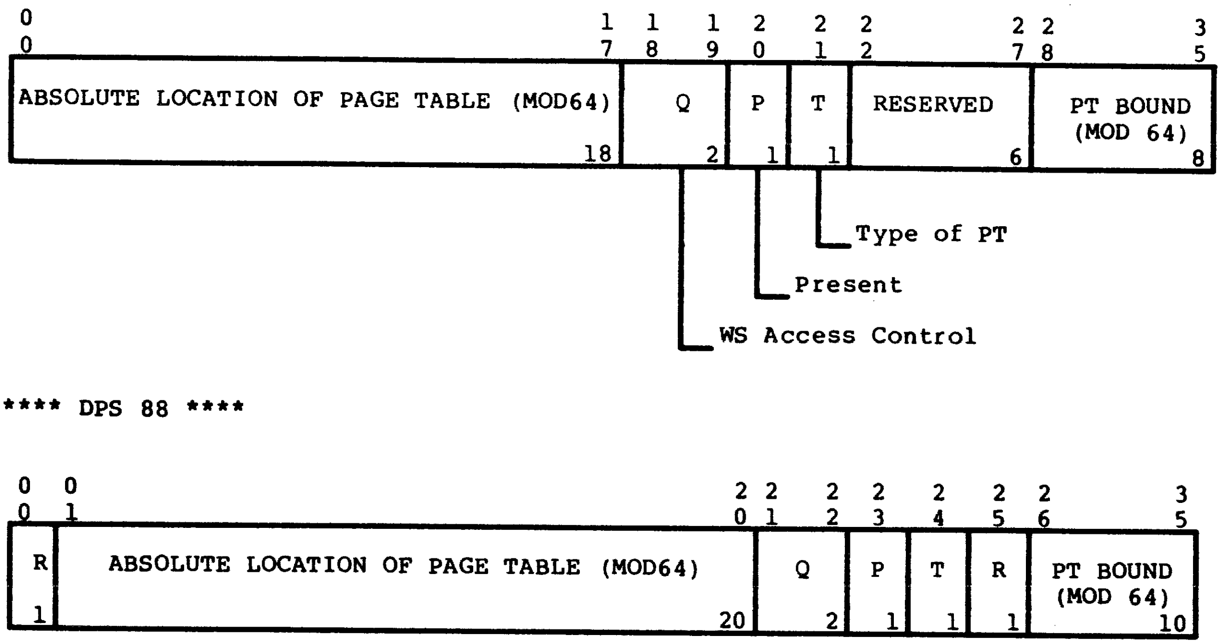


Figure 5-8. Working Space Page Table Directory Format

DPS 8 Bits	DPS 88 Bits	Description
0-17	1-20	Absolute location of page table.
18,19	21,22	WS access control provides a hardware method to force the isolation of working spaces. When one or more working spaces is allocated to a process, software will record in these bit positions of the associated PTDW, the two bits that will be checked against the first two bits of EA+LOC+BASE. This check can result in a fault.
20	23	= 0, the page table is not present. = 1, the page table is present.

DPS 8 Bits	DPS 88 Bits	Description
21	24	= 0, the page table is dense. = 1, the page table is fragmented.
22-27	0,25	Reserved to enable future increase in page table size.
28-35	26-35	Modulo 64 size of a dense page table. All zeros means size is 64 words. Has no meaning for a fragmented page table.

When the page table directory word (PTDW) is accessed and bit 20 = 0 (DPS 88: bit 23=0), a Missing Working Space fault is generated.

PAGE TABLE WORD FORMAT

The format of the page table word is given in Figure 5-9.

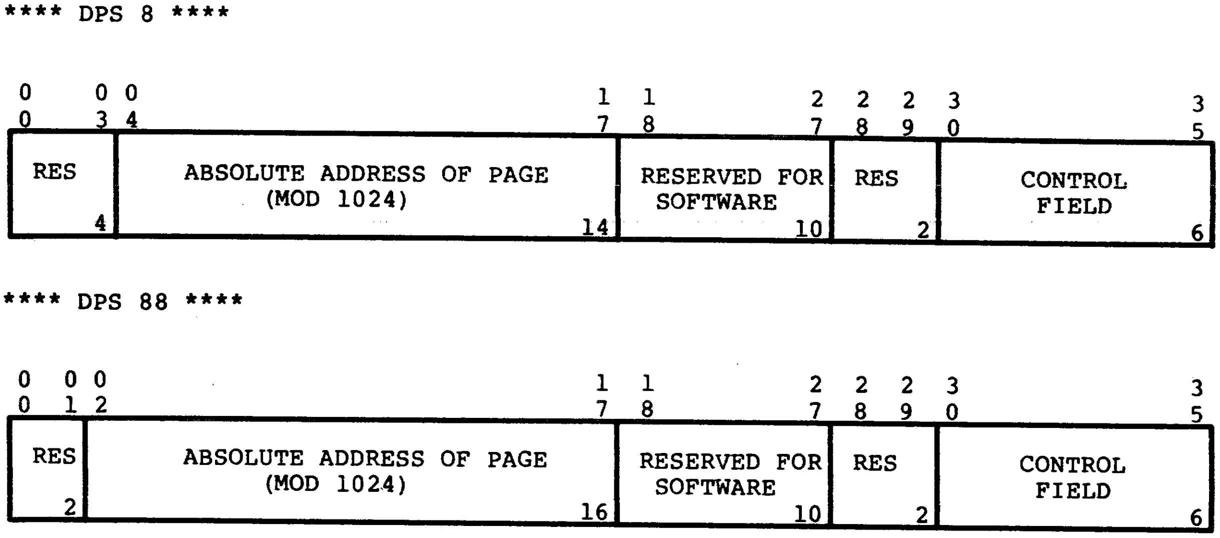


Figure 5-9. Page Table Word Format

Bits	Description
0-3 0-1 (DPS 88)	Reserved for future increase in memory size.
4-17 2-17 (DPS 88)	Absolute address of page.
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 vit = 0, page may not be written by processor = 1, page may be written by processor	Bit 31 is interpreted by processor and IOX (DPS 88), but not by IOM.
32	- Housekeeping bit = 0, nonhousekeeping page = 1, housekeeping page	Interpreted only by processor
33	- IOM (DPS 88 : IOX) 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 been 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" later in this document.

Note that the processor and the IOM (DPS 88: IOX) have separate bits to indicate a missing page. Thus, during I/O, a page may be present to the IOM (DPS 88: IOX) but missing to the processor or vice-versa. When a page is accessed, 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 the paging associative memory or paging buffer 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 the paging associative memory or paging buffer 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 or paging buffer 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 ≠ bits 0-1 of the relative virtual address, then the mapping is not completed, and a Security Fault, Class 2, is generated.

If PTDW.T=0, then the page table is a Dense Page Table.

If PTDW.T=1, then the page table is a Fragmented Page Table.

Regardless of which type of page table is used, the virtual address can be interpreted as shown in Figure 5-10. More detailed interpretations of the virtual address are also shown in Figures 5-12 and 5-16.

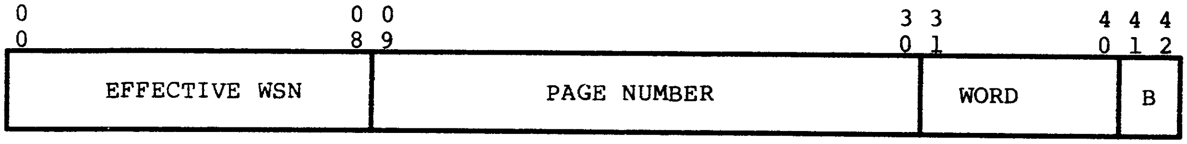
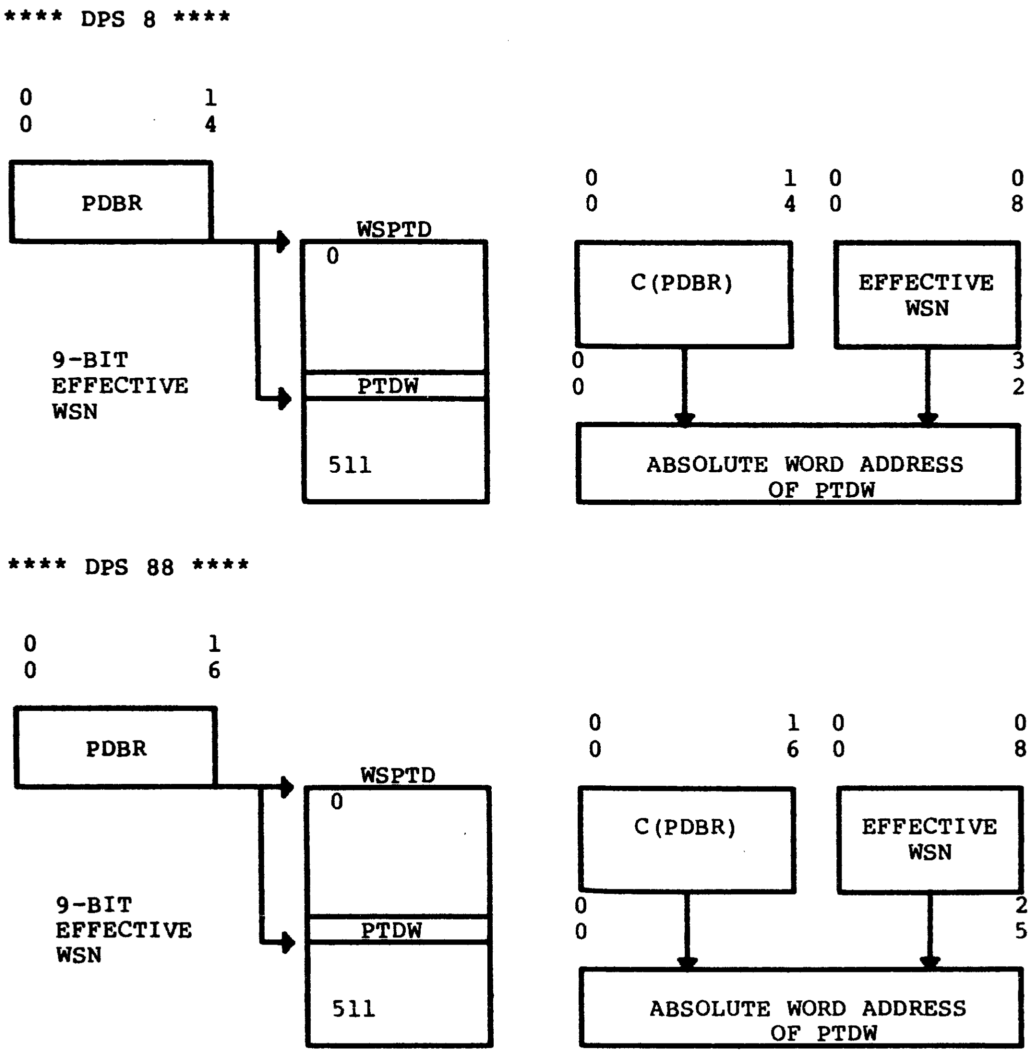


Figure 5-10. Virtual Address

LOCATING THE PAGE TABLE DIRECTORY WORD

The Page Directory Base Register (PDBR) contains the 0 modulo 512 word address of the Working Space Page Table Directory (WSPTD). Figure 5-11 shows how the hardware uses the effective WSN from the virtual address as an offset into the WSPTD to obtain the Page Table Directory Word (PTDW) for the particular working space.


**** DPS 8 ****

**** DPS 88 ****

Figure 5-11. Locating The PTDW

DENSE PAGE TABLE

The Dense Page Table that supports a particular working space must have the entire table in real memory, one word (PTW) per page. The location and size of the page table (PT) is defined by the Page Table Directory Word (PTDW). The maximum size of a Dense PT is 16K (DPS 88: 64K) words.

When the PTDW specifies a Dense PT, the virtual address is interpreted as shown in Figure 5-12.

**** DPS 8 ****

00–08	09–16	17–30	31–40	41–42
EFFECTIVE WSN	MBZ	PAGE NUMBER	WORD	B

**** DPS 88 ****

00–08	09–14	15–30	31–40	41–42
EFFECTIVE WSN	MBZ	PAGE NUMBER	WORD	B

Figure 5-12. Virtual Address, Dense Page Table

FIELD	INTERPRETATION
EFFECTIVE WSN	The working space to be accessed.
MBZ	Must be zero for a Dense PT. Thus, the upper 2^8 x 16K (DPS 88: 2^6 x 64K) pages of a working space are not addressable via a Dense PT. If these bits are not zero an STR or BND fault shall occur.
PAGE#	This page number is used as the offset, or Index, into the PT for this working space to locate the PTW. The page number is relative to the PT base address, which comes from the PTDW.
WORD	Locates the word within the 1024 word page that is being accessed.
B	The byte position within the word.

Virtual to real mapping through a Dense PT is shown in Figure 5-13 for DPS 8, and is shown in Figure 5-14 for DPS 88.

The PTDW contains the base address (0 modulo 64) of the PT. The address of the PTW is equal to the base address plus the 14-bit (DPS 88: 16-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 or paging buffer. 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.

**** DPS 8 ****

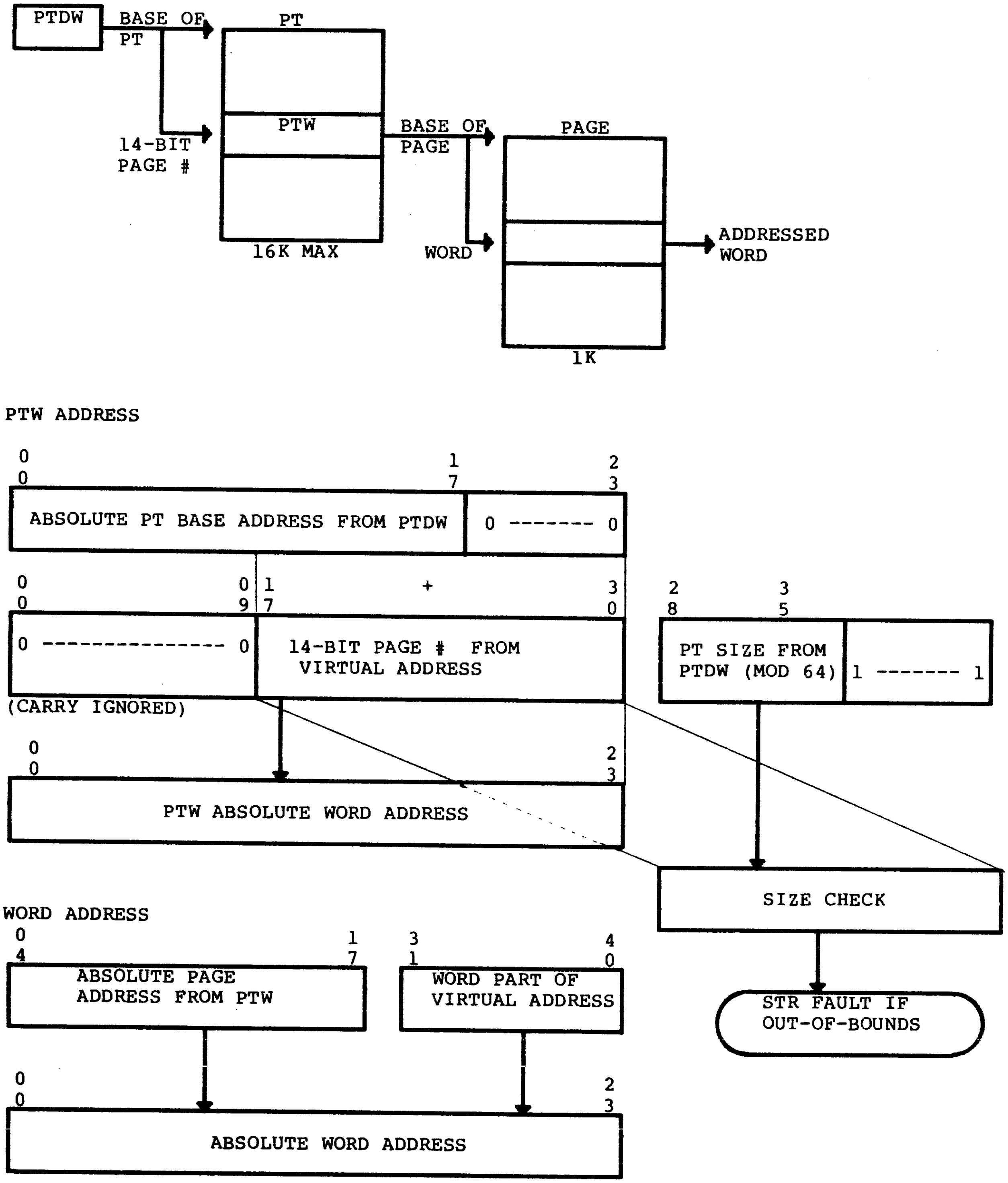


Figure 5-13. Dense Page Table Mapping DPS 8

**** DPS 88 ****

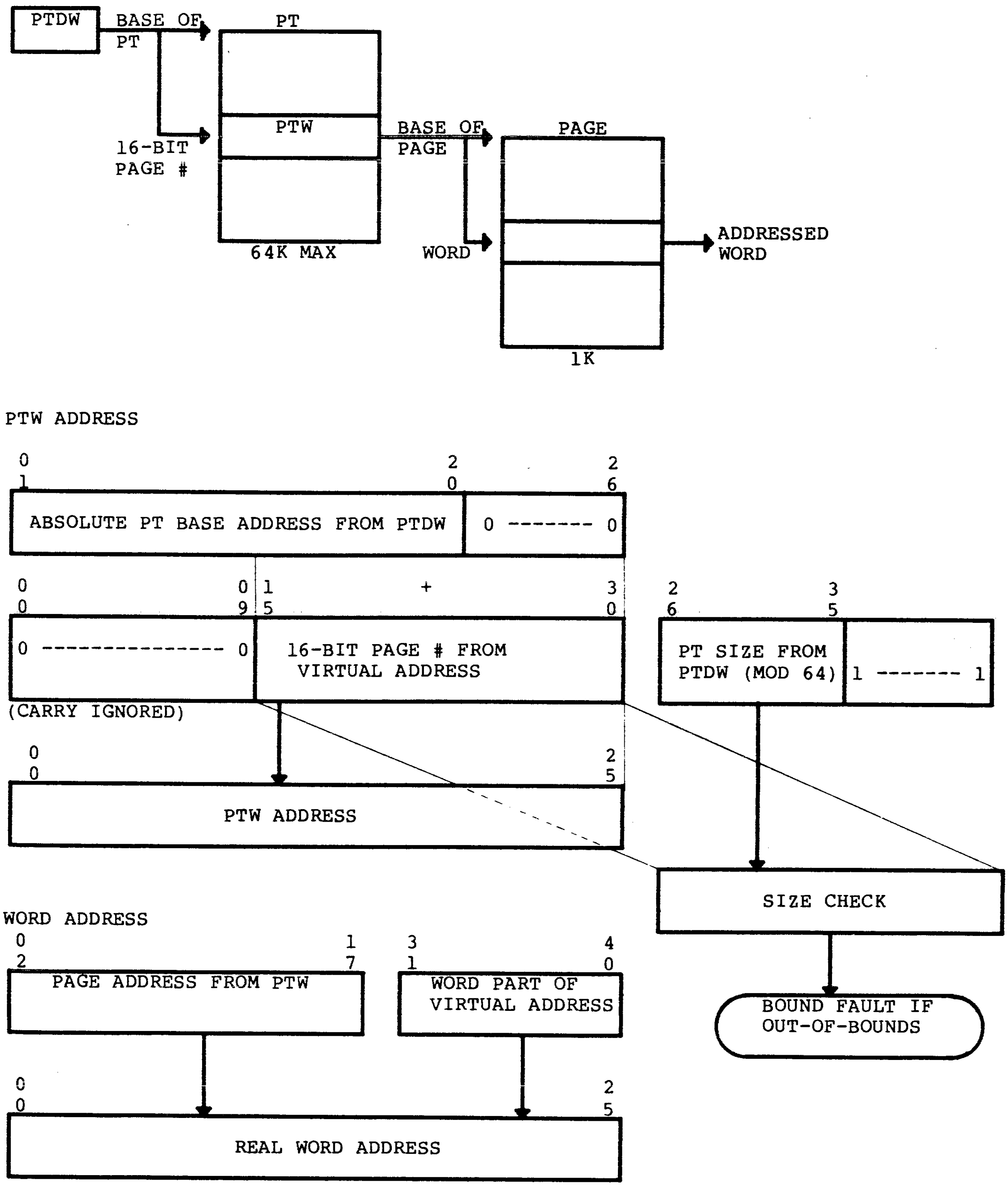


Figure 5-14. Dense Page Table Mapping DPS 88

FRAGMENTED PAGE TABLE

The Fragmented PT provides a special way for accessing pages in a large working space without requiring a large, contiguous page table to be present in real memory. The algorithm is similar to a directory set associative cache memory addressing scheme. The maximum size of a Fragmented PT is 384 words. The first 128 words are a directory containing page keys that correspond to up to 256 PTWs in the last 256 words of the PT. See Figure 5-15. This allows for mapping of up to 256K words of memory with one setting of the PT. These 256 pages can be noncontiguous virtual pages, and are a subset of the total working space. The only difference in virtual to real memory mapping when a Fragmented PT is used is the method of locating the PTW. As was the case with the Dense PT, the base address of the Fragmented PT is contained in the PTDW, obtained from the WSPTD.

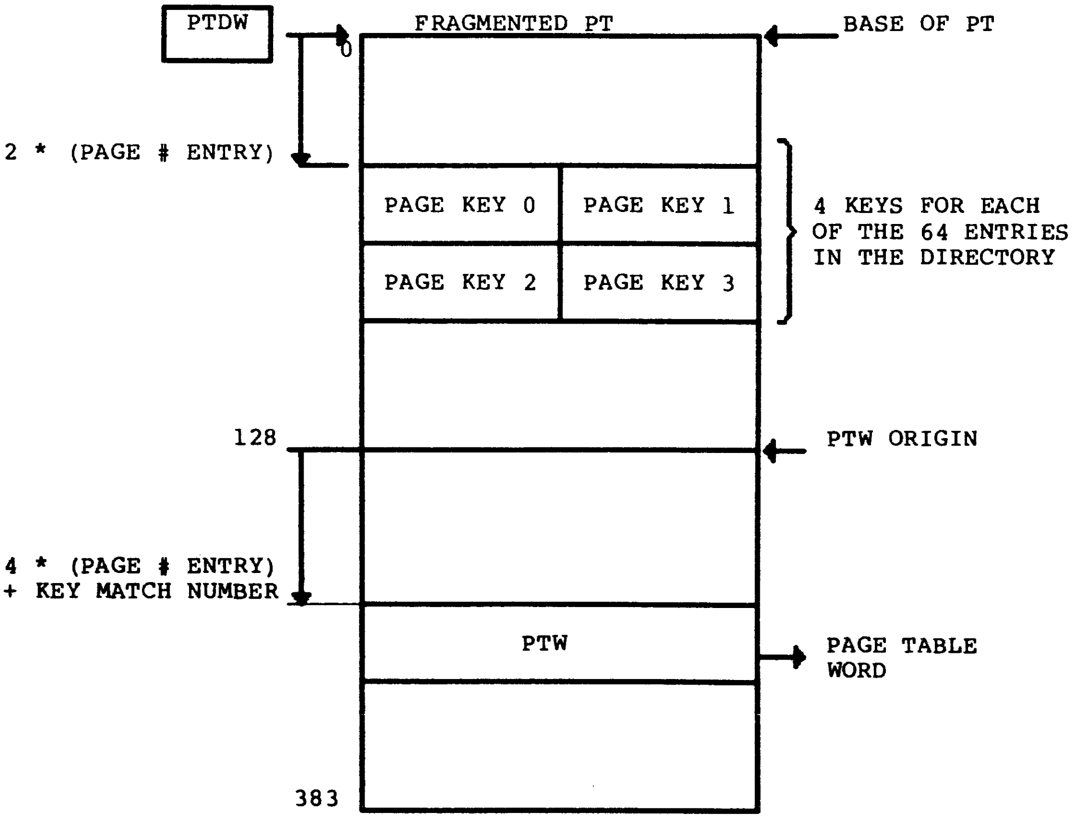


Figure 5-15. Fragmented Page Table

When the PTDW specifies a Fragmented PT, the virtual address is interpreted as shown in Figure 5-16.

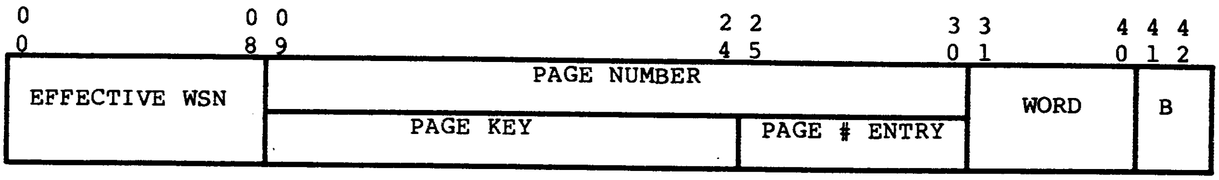
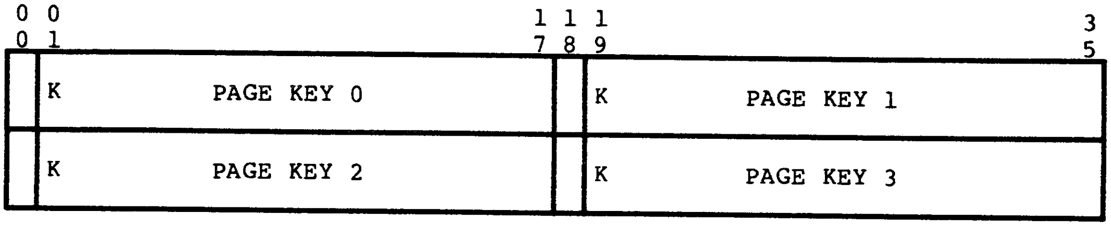


Figure 5-16. Virtual Address, Fragmented Page Table

The directory in the first 128 words of the Fragmented PT consists of 64 word pairs (directory entries), each containing four (i = 0, 1, 2, 3) 16-bit page keys with an associated bit (K) to indicate if the corresponding key is valid. See Figure 5-17.



BITS 0,18 NOT INTERPRETED BY HARDWARE

K = 0, PAGE KEY NOT VALID
K = 1, PAGE KEY VALID

Figure 5-17. Fragmented Page Table, Directory Entry

The address of a particular directory entry is determined by multiplying the 6-bit page number entry from virtual address bits 25-30 by 2, and adding this quantity to the modulo 64 base address for the page table, obtained from the PTDW. See Figures 5-18 and 5-19. The 16-bit page key field from virtual address bits 9-24 is compared with each of the valid page key fields in the selected directory entry. If the page key from the virtual address matches none of the valid page keys in the selected directory entry (or if there are not valid page keys), then the operation terminates with a Missing Page fault. If the page key from the virtual address matches more than one valid page key in the selected directory entry, then the operation ends with a Missing Page fault (DPS 88, DPS 8/20 and 8/44: the first matching page key is used). After a match is found, the address of the PTW, in the last 256 words of the PT, is equal to the modulo 64 base address for the PT, obtained from the PTDW, plus 128, plus 4 times the 6-bit page number entry from virtual address bits 25-30, plus i, where i identifies the matching page key (i = 0, 1, 2, 3). See Figures 5-18 and 5-19.

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 or paging buffer.

No hardware size check is performed when accessing the fragmented page table. It is the responsibility of systems software to ensure that fragmented page tables are always allocated in a contiguous block of 384 words.

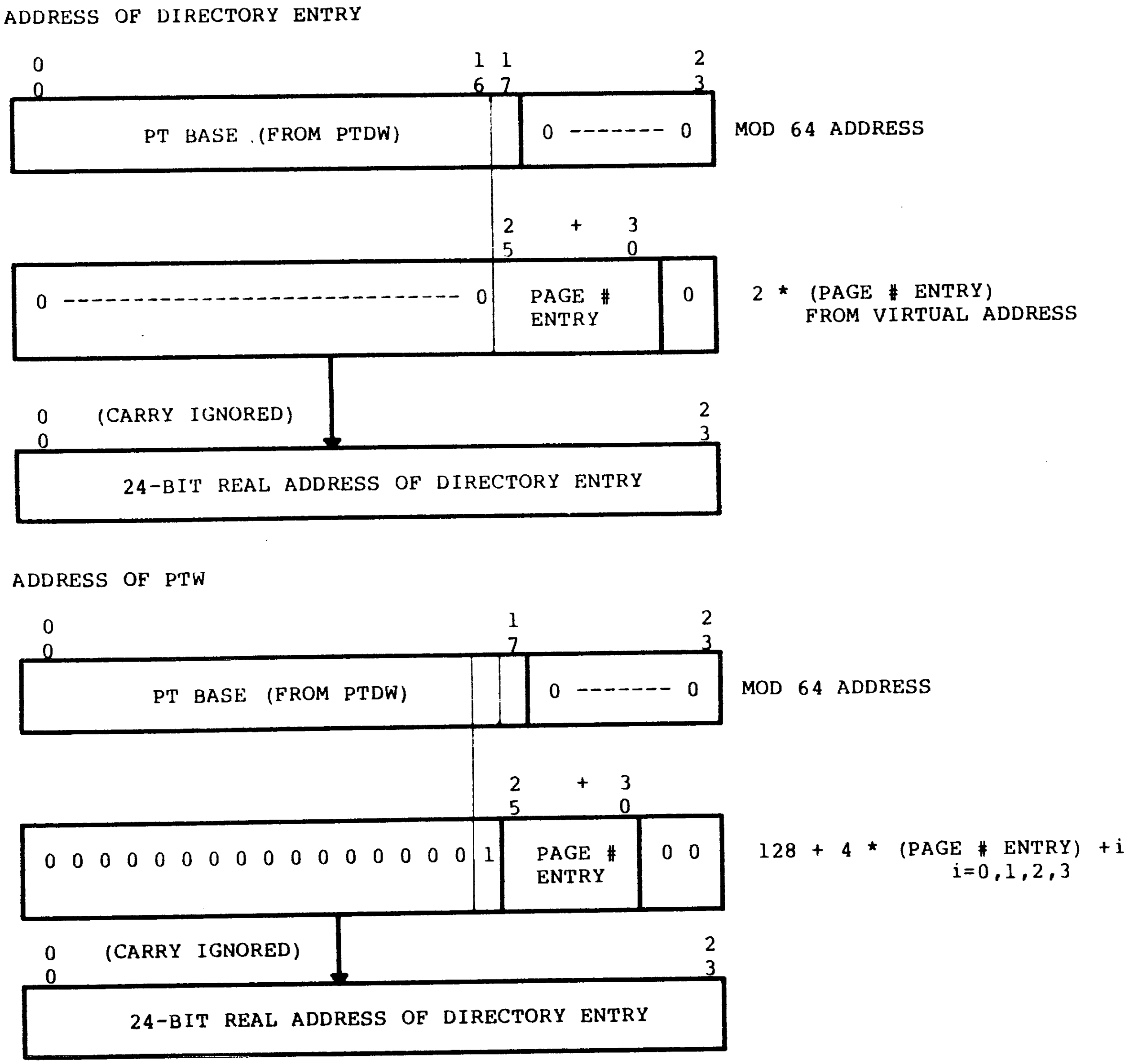


Figure 5-18. Fragmented Page Table Addressing (DPS 8)

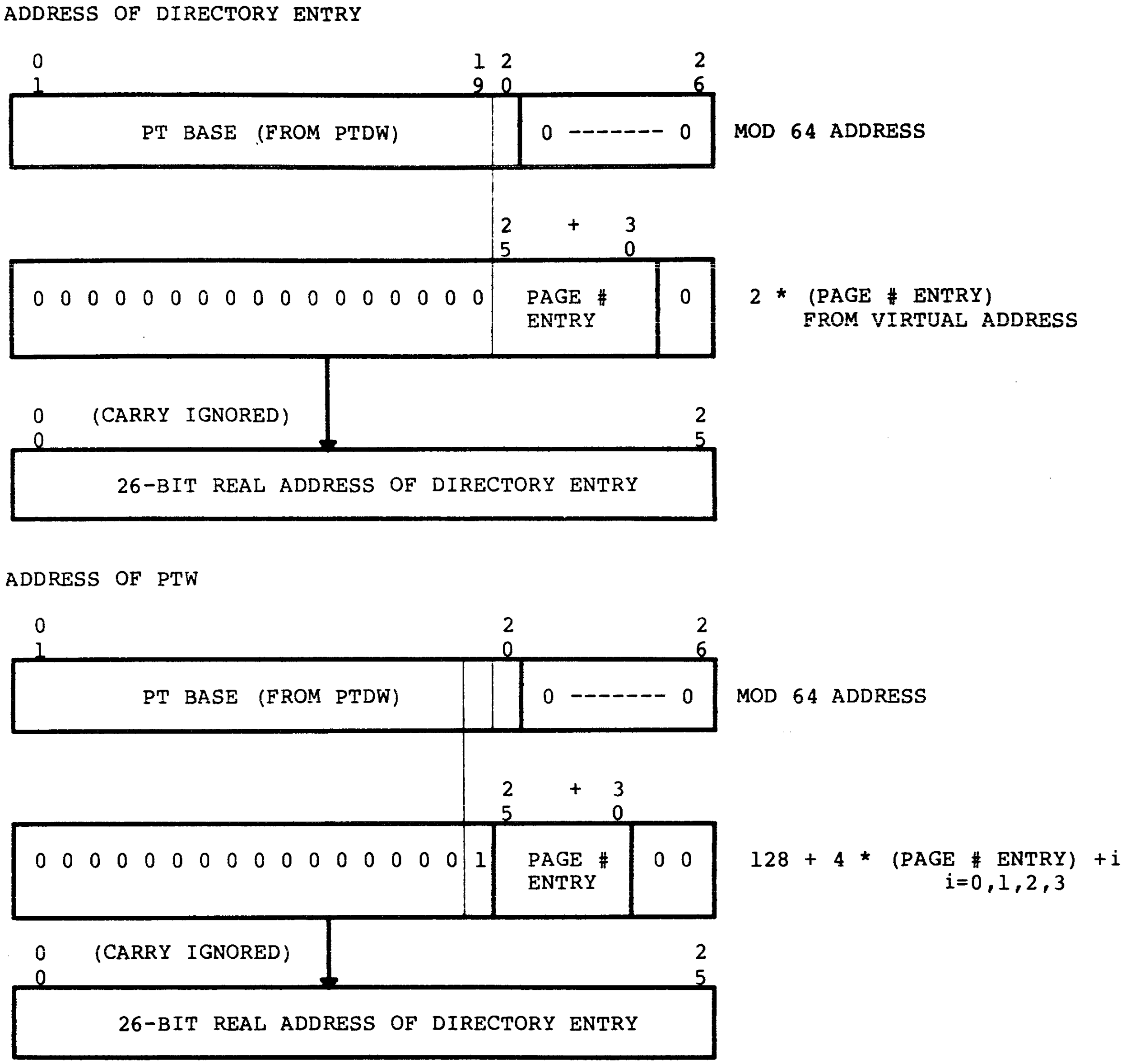


Figure 5-19. Fragmented Page Table Addressing (DPS 88)

**** DPS 8 ****

After a virtual address has been mapped to a real address as described in the previous paragraphs, this information is stored in the associative memory (AM) so that a subsequent reference to this page can be mapped in one step. The data stored in the associative memory is shown below.

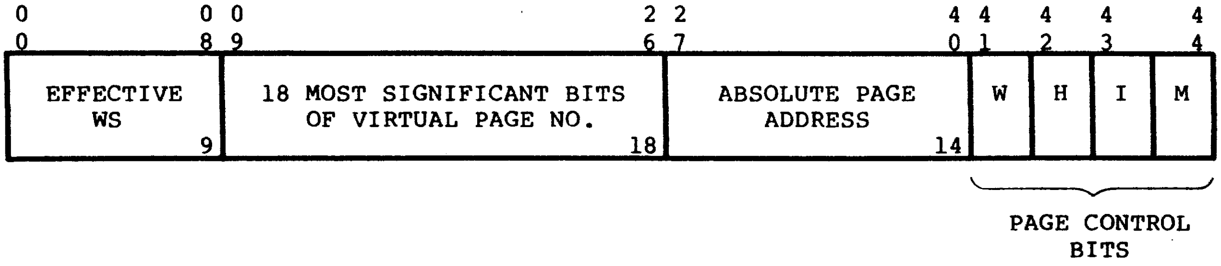


Figure 5-20. Associative Memory Word

Bits Description

0-26 The first 27 bits of the virtual address. (Note: bits 27-30 of the virtual address are used as the entry to the associative memory.)

27-40 The absolute page address from the page table word.

41-44 Page control bits:

W - write
H - housekeeping
I - IOM page present/missing
M - modified

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 16 rows by four columns (DPS 8/20 and 8/44: 64 by two columns). Each intersection of a row and a column contains a 45-bit entry as shown above. In the first phase of virtual to real memory mapping, bits 37-40 of the virtual address are used to select one of 16 (DPS 8/20 and 8/44: 64) rows. Then, bits 0-26 of the effective virtual address are compared against bits 0-26 of each of the four (DPS 8/20 and 8/44: two) row entries. If a match is found, the accompanying 14-bit absolute page address (modulo 1024) is obtained. If two or more matches are found, an STR fault is generated and the associative memory is disabled. The 10-bit word part of the virtual address is appended to form the absolute memory word address. Note that the two-bit comparison with bits 18 and 19 of the page table directory word is not made for PTWs mapped in the associative memory.

When a new address not contained in the associative memory has been mapped and the associative memory is full, the new entry replaces the oldest entry in the row (round-robin algorithm).

The associative memory may be disabled (any further comparisons or matches are ignored) by:

a. Setting the "PTW-AM Control" switch on the VU Maintenance Display and Control panel to the OFF position.

b. Executing a CAMP instruction with effective address bits 16-17 = 01.

c. Encountering an address compare of two or more columns in one of the 16 rows.

The associative memory is enabled and cleared when the "PTW-AM Control" switch is in the ON position and a CAMP instruction with effective address bits 16-17 = 10 is executed.

The associative memory is cleared whenever:

a. The processor is manually initialized.

b. It is enabled, and the CAMP instruction is executed with effective address bits 16-17 equal to 00, 10, or 11. If EA bits 16-17 = 01, the associative memory is disabled but not cleared.

c. It is disabled, and the CAMP instruction is executed with effective address bits 16-17 = 10.

d. It is enabled, and the LPDBR instruction is executed.

****

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.

Correctness of operation of an instruction as influenced by such address truncation is the responsibility of the user.

Bounds Checking

One of the capabilities provided by virtual memory is that of specifying the base and bound of a segment to the 9-bit byte level, enabling a higher level of security control and more efficient use of main memory. Since 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 (extended with 26 one bits) ≥ LOCATION + EFFECTIVE ADDRESS. The following information applies only to standard 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 an STR (DPS 88: BND) 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; that is, the character position in the base is not added to the character position held in the indirect and tally word.

NOTE: The preceding information is included to provide a warning for 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).

**** DPS 88: SSR, DSR

```
base  = 0 mod 32 bytes
bound = 31 mod 32 bytes ****
```

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 (except for DPS 8/20 8/44 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 attempted access is made to a segment with a base or bound not on a double-word boundary, an STR (DPS 88: BND) fault is generated.

BIT STRINGS AND INDEX 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, an STR (DPS 88: BND) 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. An STR (DPS 88: BND) 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 Lower	$(\text{BASE} + \text{EA})_{0-32} \| \| 111 \leq \text{BASE} + \text{BOUND}$ $(\text{BASE} + \text{EA})_{0-32} \| \| 000 \geq \text{BASE}$
Single-Word	Upper Lower	$(\text{BASE} + \text{EA})_{0-33} \| \| 11 \leq \text{BASE} + \text{BOUND}$ $(\text{BASE} + \text{EA})_{0-33} \| \| 00 \geq \text{BASE}$
Byte (includes 9-bit byte, 4-bit byte)	Upper Lower	$\text{EA}_{0-19} \leq \text{BOUND}$ 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.

Bound checking is also performed on page table sizes for dense page tables. The page number from the virtual address is bounded by:

**** DPS 8: page number$_{17-30}$ $\leq$ WSPTD PT Bound $_{28-35}$ $||$111111
and page number $_{9-16}$ must be zero ****

**** DPS 88: Page number $_{15-30}$ $\leq$ WSPTD PT BOUND $_{26-35}$ $||$111111
and page number $_{9-14}$ must be zero ****

In the absolute addressing mode, the virtual address is checked for the 26-bit (DPS 88: 28-bit) range of byte address.

**** DPS 8: Virtual address $_{9-16}$ must be zero ****

**** DPS 88: Virtual address $_{9-14}$ must be zero ****

ADDRESS WRAPAROUND

The execution of a multiword instruction that develops addresses at both the upper and lower boundaries of a maximum size segment is not permitted. This restriction is required due to the address wraparound development of the effective address (EA). For each 9-bit byte (each effective address byte), checks are made as follows:

a. For left-to-right instructions: following the calculation of the first effective address, bits 0-19 of all subsequent effective addresses are greater than those of the first effective address.

b. For right-to-left instructions: following the calculation of the first effective address, bits 0-19 of all subsequent effective addresses are less than those of the first effective address.

If these checks are violated, an STR (DPS 88: BND) fault is generated.

Multiprocessor Memory Management

The virtual memory option permits base and bound segments to be located on a byte boundary, both as a virtual address and a real address. Normal software multiprocessor protection does not exist across a segment boundary. Therefore, data may be lost when:

o two processors simultaneously refer to and change the same double word in memory,

o the double word contains a segment boundary, and

o one or both processors are executing a multiword instruction, unless the segment bouldary is modulo two words.

This condition may occur since the processor always reads a double-word from memory, changes the character(s) involved in the operation, and writes the double-word back to memory. Thus, between the reading of the double-word for a multiword instruction on one processor and the subsequent double-word store, a second processor could change that part of the double-word not affected by the multiword instruction, and the changed data would be destroyed when the double-word is stored.

SECTION VI

MACHINE INSTRUCTIONS

BASIC FEATURES

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 bytes, BCD characters, packed decimal data, and bit strings.

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:

- Boolean Operations
- Comparison Operations
- Data Movement Instructions
- Data Shifting Instructions
- Effective Address to Register Instructions
- Fixed-Point Arithmetic Instructions
- Floating-Point Arithmetic Instructions
- Special Processor Instructions

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.

Data Movement Instructions

Character handling and manipulation are facilitated by the "indirect and tally" (IT) address modification option, 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 twos complement (negative) of the contents of the memory location specified.

Data Shifting Instructions

Shifting is accomplished using a "gear-shifting" algorithm, so that 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, integral 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, 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.

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 DPS 88 instructions STAC and STACQ provide for conditional storing in the referenced memory cell, depending on the current contents of the memory cell.

The slave mode instructions providing rounded floating-point results include: DFSTR, FRD, DFRD, and FSTR.

Four master mode instructions provide system information and control for DPS 8: LCPR, SCPR, RSCR, and SSCR.

MULTIWORD INSTRUCTIONS

Multiword instructions fall into four general categories:

1. Alphanumeric instructions
2. Numeric instructions
3. Bit string instructions
4. Conversion 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 two- or three-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 a comparison to be made between two bit strings on a bit-by-bit basis and provide a capability for performing Boolean operations to combine strings and set indicators.

Conversion Instructions

Conversion instructions provide for decimal/binary and binary/decimal conversion.

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

   a. Four native data modes - ASCII, BCD, packed decimal (numeric only), and bit string. (DPS 88: A fifth data mode - EBCDIC)

3. Data Movement Capability

   a. Alphanumeric movement from left or right with character-fill.

   b. Numeric move with fill and/or rounding and scale change.

   c. Bit string manipulation using any of 16 different Boolean operations.

   d. 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

   a. Eight address registers providing for second-level indexing for all instructions (including single-word instructions).

Edited Move Micro-Operations

Both alphanumeric and numeric edited move instructions (MVE and MVNE; DPS 88: 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 character and numeric 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 (DPS 88: MVNEX)" later in this section for detailed information.

Instruction Repertoire

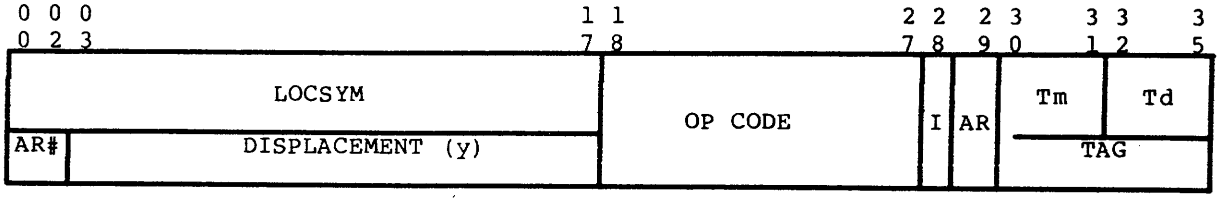
The processor interprets a 10-bit field of the instruction word as the operation code. This field size yields 1024 possible instructions of which over half are implemented.

Functional Classifications

Detailed below are the processor instructions and operation codes sorted alphabetically on the mnemonic by function. Under each category, the mnemonic, the operation code, and a brief description are given.

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 the same machine instruction word format:



AR# - One of eight address registers (0-7).

LOCSYM - Represents either address of operand or displacement from a base.

DISPLACEMENT (y) - A 15-bit displacement from the address register address (twos 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 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 Load

AARn	56n (1)	Alphanumeric Descriptor To Address Register n
LARn	76n (1)	Load Address Register n
LAREG	463 (1)	Load Address Registers
NARn	66n (1)	Numeric Descriptor to Address Register n

Address Register Store

ARAn	54n (1)	Address Register n to Alphanumeric Descriptor
ARn	64n (1)	Address Register n to Numeric Descriptor
SARn	74n (1)	Store Address Register n
SAREG	443 (1)	Store Address Registers

Address Register Special Arithmetic

This set of instructions provides 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 same instruction format:

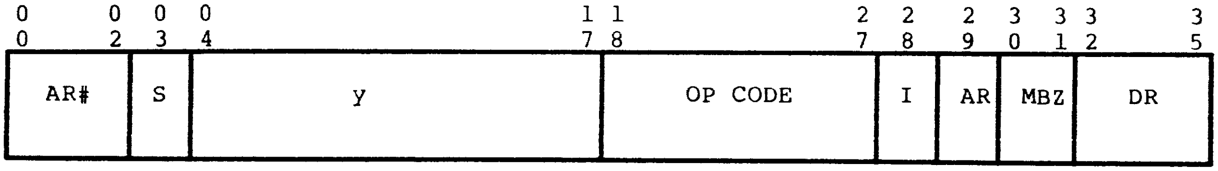


Figure 6-1. Address Register Special Arithmetic

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 (twos complement) or positive word displacement.

OP CODE - 10-bit operation code field.

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 above described sum or its twos 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. See Table 5-2.

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 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.

Indicators are unaffected by these instructions.

A4BD	502 (1)	Add 4-Bit Displacement to Address Register
A6BD	501 (1)	Add 6-Bit Displacement to Address Register
A9BD	500 (1)	Add 9-Bit Displacement to Address Register
ABD	503 (1)	Add Bit Displacement to Address Register
AWD	507 (1)	Add Word Displacement to Address Register
S4BD	522 (1)	Subtract 4-Bit Displacement from Address Register
S6BD	521 (1)	Subtract 6-Bit Displacement from Address Register
S9BD	520 (1)	Subtract 9-Bit Displacement from Address Register
SBD	523 (1)	Subtract Bit Displacement from Address Register
SWD	527 (1)	Subtract Word Displacement from Address Register

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.

Boolean Expressions

A Boolean expression is defined similarly to an algebraic expression except that the operators *, /, +, and - are interpreted as Boolean operators. The meaning of these operators is 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
*	AND, intersection	0 * 0 = 0 0 * 1 = 0 1 * 0 = 0 1 * 1 = 1
/	1s 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 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

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.

Descriptor Register Load

LDDn	67n (1)	Load Descriptor Register n

Descriptor Register Save

SDRn	11n (1)	Save Descriptor Register n

Descriptor Register Store

STDn	05n (1)	Store Descriptor Register n

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

SREG	753 (0)	Store Registers
STA	755 (0)	Store A-Register
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

Data Movement Shift

ALR	755 (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 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 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. Compare instructions are repeatable using the RPT, RPD, or RPL instruction.

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

Data Movement Load

DFLD	433 (0)	Double-Precision Floating Load
FLD	431 (0)	Floating Load
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
DUFS	537 (0)	Double-Precision Unnormalized Floating Subtract
FSB	575 (0)	Floating Subtract
UFS	535 (0)	Unnormalized Floating Subtract

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

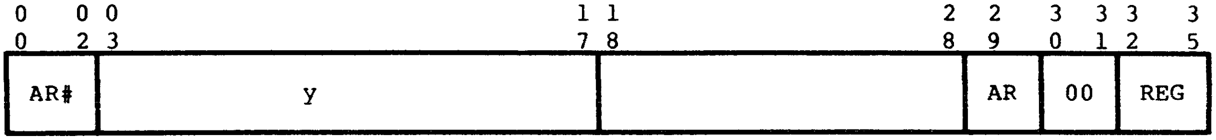
MULTIWORD INSTRUCTIONS

Operand Descriptors And Indirect Pointers

The words following a multiword instruction word are either operand descriptors or indirect pointers 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 POINTER FORMAT

An indirect pointer to an operand descriptor is interpreted as shown below (also see "Indirect Word" earlier in this manual):



AR# - A 3-bit pointer register number.

y - An 18-bit main memory address or a 15-bit word offset.

AR - Indirect via pointer register flag that controls the interpretation of the y field of the indirect pointer.

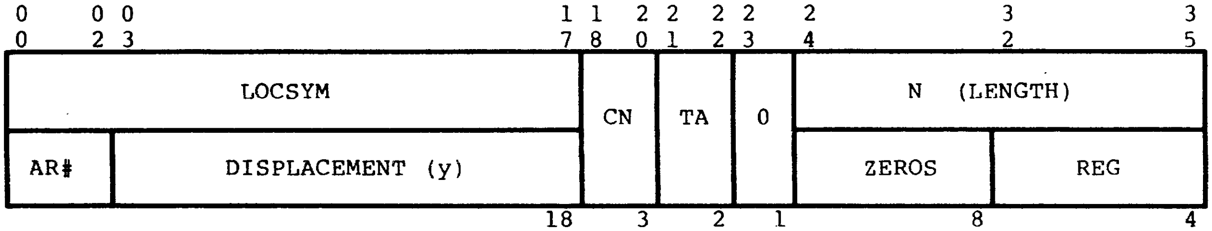
REG - The address modifier for the y field.

Alphanumeric Instructions

Alphanumeric instructions permit moving, transliteration, editing, and comparing of alphanumeric data.

ALPHANUMERIC OPERAND DESCRIPTOR FORMAT

For any operand of a multiword instruction that requires alphanumeric data, the operand descriptor is interpreted as shown below (also see "Alphanumeric Operand Descriptors" documented earlier in this manual):



AR# - A 3-bit address register number.

LOCSYM - 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 6-1 shows the interpretation of the field. A digit in the table indicates the corresponding character position (see Section II for data formats) and an "x" indicates an invalid code for the data type. Invalid codes cause IPR faults.

Table 6-1. Alphanumeric Character Number (CN) Codes

C(CN)	Data type		
	4-bit	6-bit	9-bit
000	0	0	0
001	1	1	x
010	2	2	1
011	3	3	x
100	4	4	2
101	5	5	x
110	6	x	3
111	7	x	x

TA - Type alphanumeric. This is the data type code for the operand. The interpretation of the field is shown in Table 6-2. The code shown as Invalid causes an IPR fault.

Table 6-2. Alphanumeric Data Type (TA) Codes

C(TA)	Data type
00	9-bit
01	6-bit
10	4-bit
11	Invalid

N - Operand length. If RL = 0, 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 (see "Register Codes", Table 5-1).

The alphanumeric operand descriptor is coded as follows:

```
1      8       16
    ⎧ ADSC9 ⎫ LOCSYM,CN,N,AM
    ⎨ ADSC6 ⎬
    ⎩ 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
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
CMPCT	166	(1)	(DPS 88 only)	Compare Characters and Translate

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
MMF	364	(1)	(DPS 88 only)	Move to Memory Format
MRF	360	(1)	(DPS 88 only)	Move to Register Format

Numeric Instructions

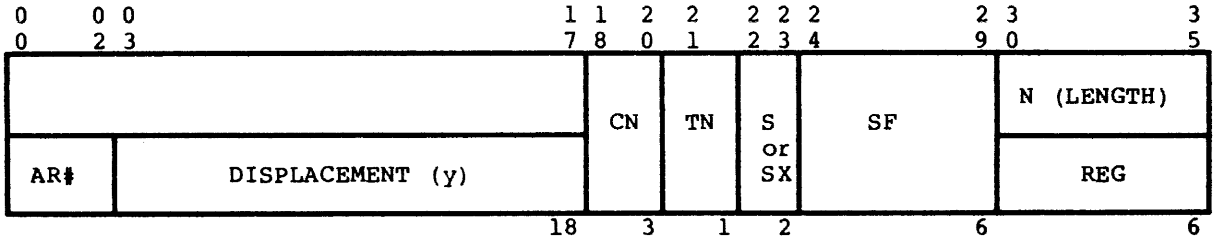
The set of numeric instructions deals with sign and magnitude operands. Floating-point decimal zero is represented as + 0 * 10**127. If any computation is performed that would result in a zero representation other than this, the hardware forces the zero representation to this format, thus preventing 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; octal 053 forms the plus sign and octal 055 forms the minus sign.

NUMERIC OPERAND DESCRIPTOR FORMAT

For any operand of a multiword instruction that requires numeric data, the operand descriptor is interpreted as shown below (also see "Numeric Operand Descriptors" documented earlier in the manual):



AR# - A 3-bit address register number.

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(TN)	Data type
0	9-bit
1	4-bit

S - Sign and decimal type of data. The interpretation of the field is shown in Table 6-3.

Table 6-3. 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, 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. This field contains the twos complement value of the base 10 scaling factor; that is, 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 move the decimal point to the left; positive values, to the right. The range of m is (-32,31).

N - Operand length. If RL = 0, 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.

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 or 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

SF - The scaling factor for scaled decimal numbers; range is +31 to -32 treated as the powers of ten.

AM - Address register containing the base.

NUMERIC COMPARE

CMPN	303 (1)		Compare Numeric
CMPNX	343 (1)	(DPS 88 only)	Compare Numeric Extended

NUMERIC MOVE

MVN	300 (1)		Move Numeric
MVNX	340 (1)	(DPS 88 only)	Move Numeric Extended
MVNE	024 (1)		Move Numeric Edited
MVNEX	004 (1)	(DPS 88 only)	Move Numeric Edited Extended

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

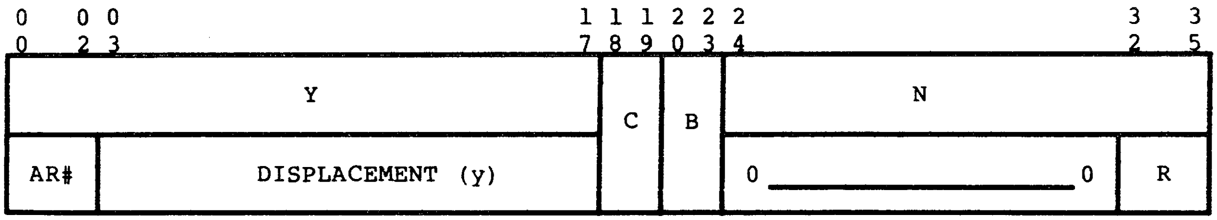
Boolean operations most commonly used are:

	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 below (also see "Bit String Operand Descriptor" documented earlier in this manual):



AR# - Address register containing the base.

Y - Nominal address of data.

y - Displacement from base.

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, 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.

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.

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.

DATA CONVERSION

BTD	301 (1)	Binary-to-Decimal Convert
DTB	305 (1)	Decimal-to-Binary Convert

Arithmetic Instructions

DECIMAL ADDITION

AD2D	202	(1)		Add Using Two Decimal Operands
AD2DX	242	(1)	(DPS 88 only)	Add Using Two Decimal Operands Extended
AD3D	222	(1)		Add Using Three Decimal Operands
AD3DX	262	(1)	(DPS 88 only)	Add Using Three Decimal Operands Extended

DECIMAL SUBTRACTION

SB2D	203	(1)		Subtract Using Two Decimal Operands
SB2DX	243	(1)	(DPS 88 only)	Subtract Using Two Decimal Operands Extended
SB3D	223	(1)		Subtract Using Three Decimal Operands
SB3DX	263	(1)	(DPS 88 only)	Subtract Using Three Decimal Operands Extended

DECIMAL MULTIPLICATION

MP2D	206	(1)		Multiply Using Two Decimal Operands
MP2DX	246	(1)	(DPS 88 only)	Multiply Using Two Decimal Operands Extended
MP3D	226	(1)		Multiply Using Three Decimal Operands
MP3DX	266	(1)	(DPS 88 only)	Multiply Using Three Decimal Operands Extended

DECIMAL DIVISION

DV2D	207	(1)		Divide Using Two Decimal Operands
DV2DX	247	(1)	(DPS 88 only)	Divide Using Two Decimal Operands Extended
DV3D	227	(1)		Divide Using Three Decimal Operands
DV3DX	267	(1)	(DPS 88 only)	Divide Using Three Decimal Operands Extended

MICRO-OPERATIONS FOR EDIT INSTRUCTIONS MVE, MVNE, AND (DPS 88: 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 of the micro-operations require special characters for insertion into the string of characters being edited. These special characters are shown under "Edit Insertion Table" below.

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 following format:

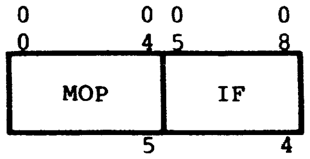


Figure 6-2. Micro-Operation (MOP) Character Format

MOP 5-bit code specifying the micro operator.

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 Table

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 6-4. For MVE and MVNE, the ASCII code is used for each initial value. (DPS 88: For MVNEX, the EIT field in the instruction word determines the character set (ASCII, BCD, or EBCDIC) to be used for the initial values.)

Table 6-4. Default Edit Insertion Table Characters

Table Entry Number	Character
1	blank
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.

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.

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.

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.

MVNE, MVE, And (DPS 88: MVNEX) Differences

The processor executes MVNE and MVNEX in a slightly different manner than it executes MVE. This is due to the 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 Operations

CHT	021	Change Table
ENF	002	End Floating Suppression
IGN	014	Ignore Source Characters
INSA	011	Insert Asterisk on Suppression
INSB	010	Insert Blank on Suppression
INSM	001	Insert Table Entry One Multiple
INSN	012	Insert On Negative
INSP	013	Insert On Positive
LTE	020	Load Table Entry
MFLC	007	Move With Floating Currency Symbol Insertion
MFLS	006	Move With Floating Sign Insertion
MORS	017	Move and OR sign
MSES	016	Move and Set Sign
MVC	015	Move Source Characters
MVZA	005	Move With Zero Suppression and Asterisk Replacement
MVZB	004	Move With Zero Suppression and Blank Replacement
SES	003	Set End Suppresion

POINTER REGISTER INSTRUCTIONS

Pointer Register Load

LDPn	47n (1)	Load Pointer Register n

Pointer Register Store

STPn	45n (1)	Store Pointer n

Pointer Register Miscellaneous

EPPRn	63n (1)	Effective Pointer to Pointer Register n
LDEAn	61n (1)	Load Extended Address n

PRIVILEGED INSTRUCTIONS

Privileged instructions are comparable to Master mode instructions. However, 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 absolute 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 (DPS 88: IPR fault) occurs.

When virtual memory is installed in the processor and is enabled, all of the former Master mode only instructions become Privileged Master mode instructions and require all of the above three conditions before they can be executed.

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 Stack Register
LDSS	773 (1)		Load Safe Store Register
LDWS	772 (1)		Load Working Space Registers
LPDBR	171 (1)		Load Page Table Directory Base Register

Register Store

SPDBR	151 (1)		Store Page Table Directory Base Register
STDSA	150 (1)		Store Data Stack Address Register
STDSD	551 (1)		Store Data Stack Descriptor Register
STPDW	155 (1)	(DPS 8 only)	Store PTWAM Directory word
STPTW	157 (1)	(DPS 8 only)	Store PTWAM Register
STSS	753 (1)		Store Safe Store Register
STTA	553 (1)		Store Test Address Registers
STTD	550 (1)	(DPS 8 only)	Store Test Descriptor Registers
STWS	752 (1)		Store Working Space Registers

Clear Associative Memory Pages

CAMP	532	(1)	(DPS 8 only)	Clear Associative Memory Pages
CAMPn	53n	(1)	(DPS 88 only)	Clear Paging Associative Memory

Clear Cache

CCAC	011	(1)	(DPS 8 only)	Clear Cache
CCACn	376	(1)	(DPS 88 only)	Clear Cache and Flush
	377	(1)	(DPS 88 only)	

Memory Control (DPS 8 Only)

RMCM	233	(0)		Read Memory Controller Mask Register
SMCM	553	(0)		Set Memory Controller Mask Register
SMIC	451	(0)		Set Memory Controller Interrupt Cells

System Control

ABSA	212	(0)	(DPS 88 only)	Absolute Address to A Register
CIOC	015	(0)		Connect Input/Output Channel
DIS	616	(0)		Delay Until Interrupt Signal
LCCL	057	(0)	(DPS 88 only)	Load Calendar Clock
LCPR	674	(0)	(DPS 8 only)	Load Central Processor Register
LDAT	336	(1)	(DPS 88 only)	Load Address Trap Register
LDO	172	(1)	(DPS 88 only)	Load Option Register
LDT	637	(0)		Load Timer Register
LIMR	553	(0)	(DPS 88 only)	Load Interrupt Mask Register
RIMR	233	(0)	(DPS 88 only)	Read Interrupt Mask Register
RIW	412	(0)	(DPS 88 only)	Read Interrupt Word Pair
RRES	231	(9)	(DPS 88 only)	Read Reserved Memory
RSCR	413	(0)		Read System Controller Register (Any Mode)
RSW	231	(0)	(DPS 8 only)	Read Switches (Any mode)
SCPR	452	(0)	(DPS 8 only)	Store Central Processor Register
SFR	452	(0)	(DPS 88 only)	Store Fault Register
SSCR	057	(0)	(DPS 8 only)	Set System controller Register
TTES	531	(0)	(DPS 88 only)	Transfer Table Entry Store
TTEZ	524	(0)	(DPS 88 only)	Transfer Table Entry Zero
TTTL	552	(0)	(DPS 88 only)	Transfer Trace Table Lock
TTTU	523	(0)	(DPS 88 only)	Transfer Trace Table Unlock

TRANSFER INSTRUCTIONS

The program transfer instructions permit the instruction counter to be stored in index registers X0 through X7 and also permit conditional and unconditional transters. 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.

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 two-word instruction has five versions and performs the functions of call, return, and co-routine invocations for intra- and inter-instruction segments and intra- and inter-domain references.

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
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

Domain Transfer (CLIMB)

CLIMB	713 (1)	Domain Transfer

MISCELLANEOUS OPERATIONS

All Mode Instructions

EPAT	412 (1)	Effective Pointer and Address to Test
LDO	172 (1)	Load Option Register (DPS 88: LDO is a privileged instruction)
PAS	176 (1)	Pop Argument Stack
STAS	750 (1)	Store Argument Stack Register
STO	152 (1)	Store Option Register
STPS	751 (1)	Store Parameter Stack 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. 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.

GTB	774 (0)	Gray-to-Binary Convert

Programmed Fault

DRL	002 (0)	Derail
MME	001 (0)	Master Mode Entry

No Operation

NOP	011 (0)	No Operation
PULS1	012 (0)	Pulse One
PULS2	013 (0)	Pulse Two

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. 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

SECTION VII

PROCESSOR INSTRUCTIONS

FORMAT OF INSTRUCTION DESCRIPTION

Each instruction in the repertoire is described in this section. The descriptions are presented in the format shown below.

MNEMONIC	INSTRUCTION NAME	OPCODE

FORMAT:	Figure or figure reference
CODING FORMAT:	Text
PROCESSOR MODE:	Text
SUMMARY:	Text and/or bit transfer equations
ILLEGAL ADDRESS MODIFICATIONS:	Text
ILLEGAL REPEATS:	Text
INDICATORS:	Text and/or logic statements
NOTES:	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: PROCESSOR MODE

The mode the processor should be in to execute the instruction.

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: 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 7: ILLEGAL REPEATS

A list of the repeat instructions that cannot be used with the instruction.

Line 8: 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 9: NOTES

Further explanations for those cases where the summary may not be sufficient to understand the operation.

Line 10: EXAMPLE(S)

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
AND	The Boolean connective AND (symbol ^)
AM	Address register modification
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) is defined as 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
CN	The original character number within the data word referred to by the original data word address
DR	Displacement register (bits 32-35)
EA	Character set definition, EBCDIC (0) or ASCII (1)
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)
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 — Bits used in the 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

MFn — 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)

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.

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

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

RL — Register or length indicator

RM — Register modification

S — Sign and decimal type

SF — Scaling factor

SX — Sign and scaling

Symbol	Meaning
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 which type of alphanumeric character is 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
y	A 15-bit displacement from the address register address
Y	The effective word address (18 bits) 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
XOR	The Boolean connective EXECLUSIVE OR
≠	Not equal

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 described in Figure 7-1.

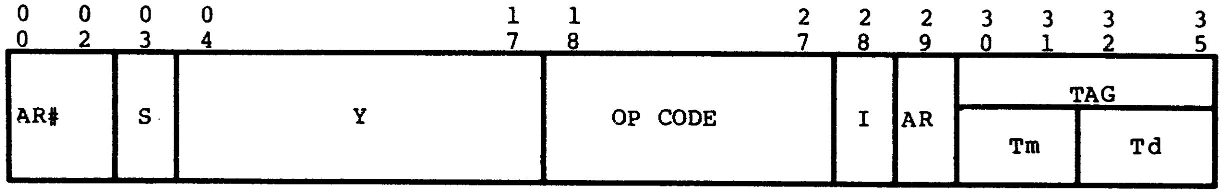


Figure 7-1. Single-Word Instruction Format

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.

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.

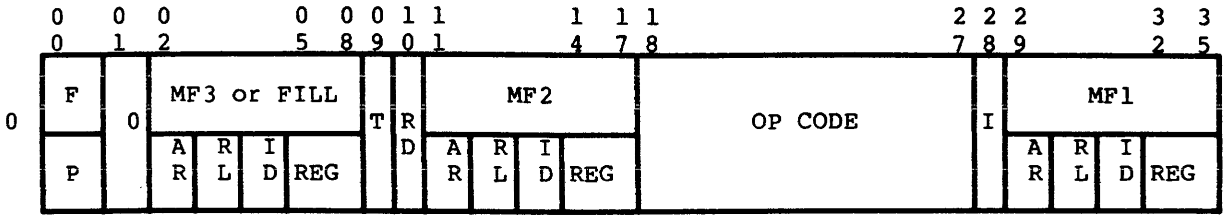
Td - (Bits 32-35) Index Register or modification variation designator.

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. 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. X0 is used to control terminate conditions and tally.

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 7-2.



The number of words and fields within the words will vary by instruction, but use the following general format:

1	Operand Descriptor or Indirect Pointer to Operand Descriptor 1
2	Operand Descriptor or Indirect Pointer to Operand Descriptor 2
3	Operand Descriptor or Indirect Pointer to Operand Descriptor 3 (When Required By the Instruction)

Figure 7-2. Multiword Instruction Format

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" documented earlier in this manual.)

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, X<u>n</u>).

Address Register Special Arithmetic Instructions

This set of instructions provides 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 7-3:

00 - 02	03	04 - 17	18 - 27	28	29	30 - 31	32 - 35
AR#	S	y	OP CODE	I	AR	MBZ	DR

Figure 7-3. 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 twos 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 fcr 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 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.

No indicators are affected by these instructions.

List Of Instructions

The following pages will detail in alphabetical order the machine instructions.

A4BD A4BDX	Add 4-Bit Displacement to Address Register	502 (1)

FORMAT: Special arithmetic instruction format (see Figure 7-3)

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.

PROCESSOR MODE: Any

SUMMARY: Description is the same as for A6BD except that the register specified by the DR field contains a count of 4-bit characters that must be effectively divided by 8. The AR is forced to point to a 4-bit character boundary prior to addition.

ILLEGAL ADDRESS MODIFICATIONS: All except N, AU, QU, AL, QL, and index registers

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

EXAMPLES:

```
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	Add 6-Bit Displacement to Address Register	501 (1)

FORMAT: Special arithmetic instruction format (see Figure 7-3)

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.

PROCESSOR MODE: Any

SUMMARY: 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) and 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. With 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.

ILLEGAL ADDRESS MODIFICATIONS: All except N, AU, QU, AL, QL, and index registers

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

EXAMPLES:

```
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	Add 9-Bit Displacement to Address Register	500 (1)

FORMAT: Special arithmetic instruction format (see Figure 7-3)

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.

PROCESSOR MODE: Any

SUMMARY: 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.

ILLEGAL ADDRESS MODIFICATIONS: All except N, AU, QU, AL, QL, and index registers

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

EXAMPLES:

```
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 7-1)

CODING FORMAT:

```
1      8       16
_________________________________
       AARn    LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: For n = 0, 1, ... or 7 as determined by op code

$C(Y)_{0-17} \dashrightarrow C(ARn)_{0-17}$

$C(Y)_{18-20} \xrightarrow{\text{translated}} C(ARn)_{18-23}$

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

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used or if the descriptor TA field contains code 11.

**** DPS 88: An Illegal Procedure fault occurs if descriptor CN field contains xx1 for TA = 00, or 11x for TA = 01. ****

EXAMPLES:

```
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	Add Bit Displacement to Address Register	503 (1)

FORMAT: Special arithmetic instruction format (see Figure 7-3).

CODING FORMAT

```
1      8      16
_______________________________
       {ABD }
       {ABDX}  word displacement,R,AR
```

When the mnemonic is coded with an X (ABDX), bit 29 is forced to zero.

PROCESSOR MODE: Any

SUMMARY: The bit count contained in the register specified by the DR field is converted into a word, character, and bit address, and the word portion is added to the y field (bit 3 extended). If bit 29 = 0, the resulting word address from the add and the character and bit values from the divide operation replaces bits 0-23 of the specified AR. If bit 29 = 1, these values are added to bits 0-23 of the specified AR and this result replaces bits 0-23 of the specified AR.

ILLEGAL ADDRESS MODIFICATIONS: All except N, AU, QU, AL, QL, and index registers

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

EXAMPLES:

```
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
```

**** DPS 88 ONLY ****

ABSA	Absolute Address to A Register	212 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode.

SUMMARY: Final main memory address, Y --> $C(A)_{0-25}$
00-->0 $C(A)_{26-35}$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS:

Zero	- If C(A) = 0, then ON; otherwise, OFF
Negative	- If $C(A)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. The use of this instruction in other than the Privileged Master mode causes an IPR fault.
2. An Illegal Procedure fault occurs if illegal address modification is used.

****

AD2D	Add Using Two Decimal Operands	202 (1)

FORMAT:

00	01 08	09	10	11 17	18 Op Code 27	28	29 35
P	0----------0	T	RD	MF2	202(1)	I	MF1

00 17	18 20	21	22 23	24 29	30 35
Y1	CN1	TN1	S1	SF1	N1

00 17	18 20	21	22 23	24 29	30 35
Y2	CN2	TN2	S2	SF2	N2

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
```

PROCESSOR MODE: 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 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-       0123450-
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)
```

**** DPS 88 ONLY ****

AD2DX	Add Using Two Decimal Operands Extended	242 (1)

FORMAT:

00	01	02 08	09	10	11 17	18 Op Code 27	28	29 35
EA	NS	0-------0	T	RD	MF2	242 (1)	I	MF1

00 17	18 20	21	22 23	24 29	30 35
Y1	CN1	TN1	SX1	SF1	N1

00 17	18 20	21	22 23	24 29	30 35
Y2	CN2	TN2	SX2	SF2	N2

PROCESSOR MODE: Any

SUMMARY: C(string 2) + C(string 1) --> C(string 2)

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. If SX2 indicates a scaled 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. 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. The character set is defined by EA. Placement of an overpunched sign in the output is controlled by NS. If RD is 1, rounding takes place prior to storage. 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.

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 fault same as for MVN.

3. Independent 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 calculated result after it has been aligned.

4. All notes for AD3D apply to AD2DX.

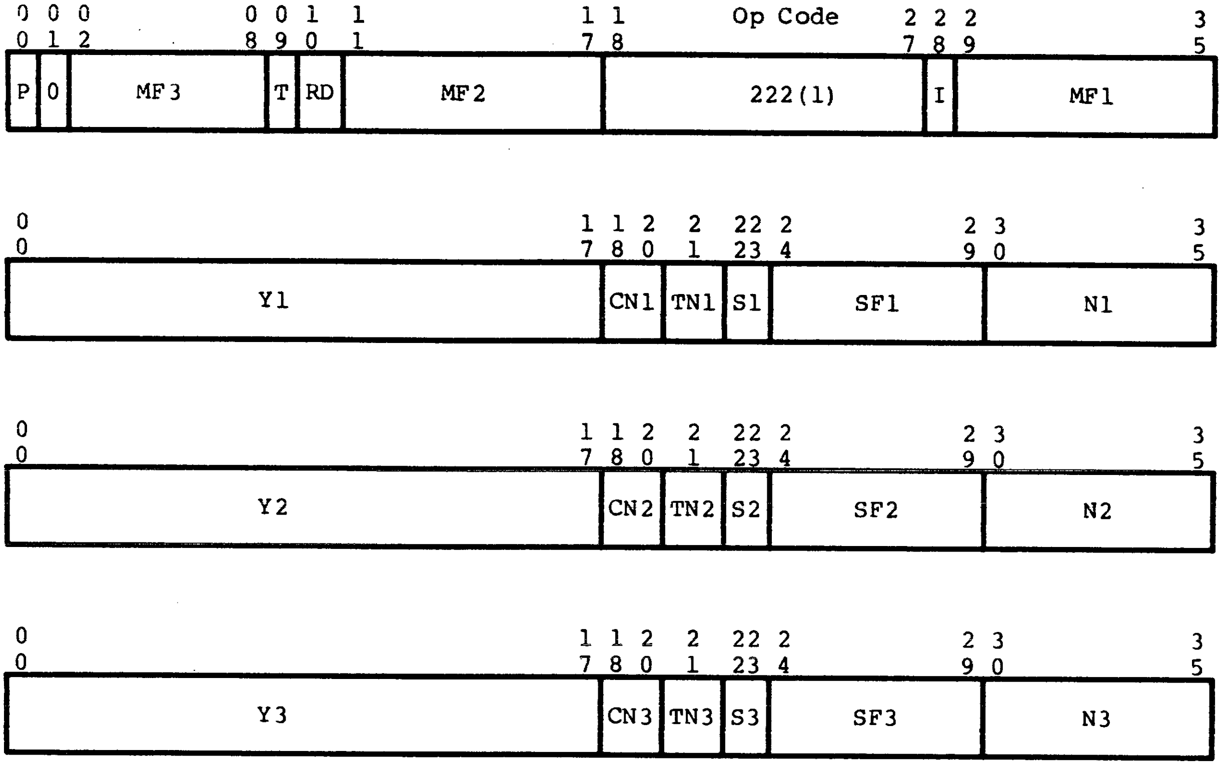
5. For coding of overpunched signs, see MVNX.

6. 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
```

PROCESSOR MODE: 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 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 scaled 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. 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 is specified), OFF

Exponent Overflow - If exponent of floating-point result is greater than 127, then ON; otherwise, unchanged

Exponent Underflow - If exponent of floating-point result is less than -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 a 1.

2. Illegal Procedure fault same as for MVN.

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.

      ****DPS 88: Note that DPS 88 accommodates all possible intermediate results without loss of significant digits.****

   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. ****DPS 88: If an illegal digit or sign is detected, part or all of the receiving field may be changed before the IPR fault occurs.****

EXAMPLE:

```
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
```

****DPS 88 ONLY****

AD3DX	Add Using Three Decimal Operands Extended	262 (1)

FORMAT:

00	01	02 – 08	09	10	11 – 17	18 Op Code 27	28	29 – 35
EA	NS	MF3	T	RD	MF2	262 (1)	I	MF1

00 – 17	18 – 20	21	22 23	24 – 29	30 – 35
Y1	CN1	TN1	SX1	SF1	N1

00 – 17	18 – 20	21	22 23	24 – 29	30 – 35
Y2	CN2	TN2	SX2	SF2	N2

00 – 17	18 – 20	21	22 23	24 – 29	30 – 35
Y3	CN3	TN3	SX3	SF3	N3

PROCESSOR MODE: Any

SUMMARY: C(string 2) + C(string 1) --> C(string 3)

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 scaled 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 EA. Placement of overpunched sign in the output is controlled by NS. If RD is 1, rounding takes place 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 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, OFF.

NOTES:

1. Truncation fault occurs if the truncation indicator is set and the truncation fault enable (T) bit is a 1.

2. Illegal Procedure fault same as for MVN.

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. An Illegal Procedure fault occurs if illegal address modification is used.

ADA	Add to A-Register	075 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: Any

SUMMARY: $C(E) + C(Y)_{0-7}$ --> C(E)

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - Set OFF

Negative - Set OFF

Exponent Overflow - If exponent is greater than +127, then ON

Exponent Underflow - If exponent is less than -128, then ON

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

ADL	Add Low to AQ-Register	033 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(AQ) + C(Y, right-adjusted) --> C(AQ)

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

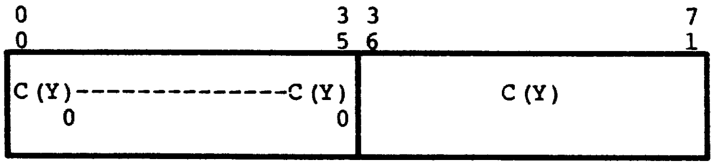
Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

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

NOTES:

1. This instruction forms the following 72-bit number:



That is, bits 0-35 are each identical to bit 0 of C(Y). This number is added to C(AQ).

2. 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 7-1)

PROCESSOR MODE: 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

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

NOTES:

1. 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.

2. An Illegal Procedure fault occurs if illegal address modification is used.

ADLAQ	Add Logical to AQ-Register	037 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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

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.

NOTES:

1. This instruction is identical to ADAQ 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.
2. 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 7-1)

PROCESSOR MODE: 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

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.

NOTE: This instruction is identical to ADQ 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.

ADLXn	Add Logical to Index Register n	02n (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0,1..., 7 as determined by op code
$C(Xn) + C(Y)_{0-17}$ --> $C(Xn)$; C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of ADLX0

INDICATORS:

Zero - If C(Xn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(Xn) is generated, then ON; otherwise, OFF. When the Carry indicator is ON, the range of Xn has been exceeded

NOTES:

1. 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.
2. DL modification executes with all zeroes for data.
3. An Illegal Procedure fault occurs if illegal address modification is used.

ADQ	Add to Q-Register	076 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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

ADXn	Add to Index Register n	06n (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0, 1..., or 7 as determined by op code
$C(Xn) + C(Y)_{0-17}$--> C(Xn); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of ADX0

INDICATORS:

Zero - If C(Xn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of Xn is exceeded, then ON

Carry - If a carry out of bit 0 of C(Xn) is generated, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged as illegal but is executed with all zeros for data.

2. An Illegal Procedure fault occurs if illegal address modification is used.

ALR	A-Register Left Rotate	775 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: 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 into bit position 35.

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 rotate count in the instruction must be a decimal number. To ´right-rotate´ n bits, use ALR 36-n.
2. An Illegal Procedure fault occurs if illegal address modification is used.

ALS	A-Register Left Shift	735 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Shift C(A) left the number of positions indicated by bits 11-17 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

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

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modification is used.

ANA	AND to A-Register	375 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 7-1)
PROCESSOR MODE:	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 7-1)
PROCESSOR MODE:	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

ANSA	AND to Storage from A-Register	355 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 is used.

ANSQ	AND to Storage from Q-Register	356 (0)

FORMAT:	Single-word instruction format (see Figure 7-1)
PROCESSOR MODE:	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 is used.

ANSXn	AND to Storage from Index Register n	34n (0)

FORMAT:	Single-word instruction format (see Figure 7-1)
PROCESSOR MODE:	Any
SUMMARY:	For n = 0, 1, ..., or 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
ILLEGAL ADDRESS MODIFICATIONS:	DU, DL, CI, SC, SCR
ILLEGAL REPEATS:	RPT or RPD of ANSX0; RPL
INDICATORS:	Zero - If bits $C(Y)_{0-17}$ = 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 is used.

ANXn	AND to Index Register n	36n (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0, 1 ..., or 7 as determined by op code
For i = 0 to 17, $C(Xn)_i$ AND $C(Y)_i$ --> $C(Xn)_i$;
C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of ANX0

INDICATORS: Zero - If C(Xn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)_0$ = 1, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

AOS	Add One to Storage	054 (0)

FORMAT:	Single-word instruction format (see Figure 7-1)
PROCESSOR MODE:	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 is used.

ARAn	Address Register n to Alphanumeric Descriptor	54n (1)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
 1      8       16
 ___________________________________
        ARAn    LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: For n = 0, 1..., or 7 as determined by op code

$C(ARn)_{0-17} \rightarrow C(Y)_{0-17}$

$C(ARn)_{18-23} \xrightarrow{\text{translated}} C(Y)_{18-20}$

$C(Y)_{21-35}$ unchanged

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: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used, if the descriptor TA field contains code 11, or if descriptor bit 23 = 1.

****DPS 88, DPS 8/30 and DPS 8/44: IPR occurs if the descriptor TA field contains code 11, if descriptor CN field contains xx1 for TA = 00, or if descriptor CN field contains 11x for TA = 01.****

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 7-1)

PROCESSOR MODE: Any

SUMMARY: Shift C(A) right the number of positions indicated by bits 11-17 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 is used.

ARNn	Address Register n to Numeric Descriptor	64n (1)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1      8      16
       ARNn   LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

For n = 0, 1 .., or 7 as determined by op code

$C(ARn)_{0-17}$ --> $C(Y)_{0-17}$

translated

$C(ARn)_{18-23}$ --> $C(Y)_{18-20}$

Bits 21-35 of C(Y) unchanged

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

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

****DPS 88, DPS 8/20 and 8/44: An Illegal Procedure fault occurs if descriptor CN field contains xx1 for TN = 0.****

ARS	A-Register Right Shift	731 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: 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).

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 is used.

ASA	Add To Storage From A-Register	055 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 is used.

ASQ	Add To Storage From Q-Register	056 (0)

FORMAT:	Single-word instruction format (see Figure 7-1)
PROCESSOR MODE:	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 is used.

ASXn	Add To Storage From Index Register n	04n (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0, 1,..., or 7 as determined by op code
$C(Xn) + C(Y)_{0-17}$ --> C(Y); C(Xn) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT or RPD of ASX0

INDICATORS:

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

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

AWCA	Add with Carry to A-Register	071 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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) + 0...01 --> 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

NOTES:

1. 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.

2. 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.

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 7-1)

PROCESSOR MODE: 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) + 0...1 --> 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

NOTES:

1. 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.
2. 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.

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	Add Word Displacement to Address Register	507 (1)

FORMAT: Special arithmetic instruction format (see Figure 7-3)

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.

PROCESSOR MODE: Any

SUMMARY:

If bit 29 = 1: $C(AR\underline{n})_{0-17} + y + C(DR) \dashrightarrow AR\underline{n}_{0-17}$

If bit 29 = 0: $y + C(DR) \dashrightarrow AR\underline{n}_{0-17}$

In either case, zeros $\dashrightarrow AR\underline{n}_{18-23}$

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.

ILLEGAL ADDRESS MODIFICATIONS: All except N, AU, QU, AL, QL, and index registers

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

EXAMPLES:

```
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 7-1)

PROCESSOR MODE: Any

SUMMARY: Shift C(A) left 3 positions

$|C(A)| \div C(Y)$ --> 4-bit quotient;
C(A) - (C(Y) * quotient) --> remainder

Shift C(Q) left 6 positions;
4-bit quotient --> CQ_{32-35}
remainder --> C(A)

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. Restrictions

   o The largest number that can be converted with the BCD instruction is that represented by 33 bits.

   o One 6-bit character is produced each time the BCD instruction is executed.

   o The character produced represents a decimal digit from 0 to 9.

   o One full 36-bit word cannot be directly converted by the BCD instruction.

   o An Illegal Procedure fault occurs if illegal address modification is used.

2. 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.

   $c_i = 8^i * 10^{n-i}$ (for i = 1, 2,...),

   with n being defined by $10^{n-1} \leq |\text{ number }| \leq 10^n - 1$.

For base K other than 10:

$c_i = 8^i * K^{n-1}$, where $K^{n-1} \leq |\text{ number }| \leq K^n-1$.

EXAMPLE:

```
1      8      16
_________________________
       LDA    =15,DL
       LDQ    0,DL
       BCD    =80,DL
       BCD    =64,DL
```

Conversion Constants

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.

The values in the table below 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 7-1. Binary-To-BCD Conversion Constants

Starting Range of C(AR) Conversion Step	$-10^{10} + 1$ --> $10^{10} - 1$	$-10^{9} + 1$ --> $10^{9} - 1$	$-10^{8} + 1$ --> $10^{8} - 1$	$-10^{7} + 1$ --> $10^{7} - 1$
1	8^1 X 10^9	8 X 10^8	8 X 10^7	8 X 10^6
2	8^2 X 10^8	8^2 X 10^7	8^2 X 10^6	8^2 X 10^5
3	8^3 X 10^7	8^3 X 10^6	8^3 X 10^5	8^3 X 10^4
4	8^4 X 10^6	8^4 X 10^5	8^4 X 10^4	8^4 X 10^3
5	8^5 X 10^5	8^5 X 10^4	8^5 X 10^3	8^5 X 10^2
6	8^6 X 10^4	8^6 X 10^3	8^6 X 10^2	8^6 X 10^1
7	8^7 X 10^3	8^7 X 10^2	8^7 X 10^1	8^7
8	8^8 X 10^2	8^8 X 10^1	8^8	
9	8^9 X 10^1	8^9		
10	8^{10}			

Table 7-1 (cont). Binary-To-BCD Conversion Constants

$-10^6 + 1$ --> $10^6 - 1$	$-10^5 + 1$ --> $10^5 - 1$	$-10^4 + 1$ --> $10^4 - 1$	$-10^3 + 1$ --> $10^3 - 1$	$-10^2 + 1$ --> $10^2 - 1$	$-10^1 + 1$ --> $10^1 -1$
8 X 10^5	8 X 10^4	8 X 10^3	8 X 10^2	8 X 10^1	8
8^2 X 10^4	8^2 X 10^3	8^2 X 10^2	8^2 X 10^1	8^2	
8^3 X 10^3	8^3 X 10^2	8^3 X 10^1	8^3		
8^4 X 10^2	8^4 X 10^1	8^4			
8^5 X 10^1	8^5				
8^6					

BTD	Binary-to-Decimal Convert	301 (1)

FORMAT:

			Op Code		
00	01 10	11 17	18 27	28	29 35
P	0---------------0	MF2	301(1)	I	MF1

00	02	03 17	18 20	21 29	30	32 35
Y1			CN1	0------------------0	N1	
a1		Y1			00	R1

00	02	03 17	18 20	21	22 23	24 29	30	32 35
Y2			CN2	TN2	S2	0---------0	N2	
a2		Y2					00	R2

CODING FORMAT:

```
1      8          16
       BTD        (MF1),(MF2),P
       NDSC9      LOCSYM,CN,N,,,AM
       NDSCn      LOCSYM,CN,N,S,,AM
```

PROCESSOR MODE: Any

SUMMARY:

$$C(\text{string }1) \xrightarrow{\text{converted}} C(\text{string }2)$$

The twos 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. 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 resultant number generated 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

NOTE: An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2, if L1 is less than one or greater than eight, 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
```

****DPS 8 ONLY****

CAMP	Clear Associative Memory Pages	532 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode.

SUMMARY: When the PTWAM is ON:

If EA_{16-17} = 00, 10 or 11; the PTWAM is cleared as described below.

If EA_{16-17} = 01, the PTWAM is turned OFF but is not cleared.

When the PTWAM is OFF:

If EA_{16-17} = 10, the PTWAM is cleared. It is also turned ON if the PTWAM Control switch on the VU Maintenance Display and Control panel is in the ON position.

If EA_{16-17} = 00, 01 or 11, the PTWAM is not affected.

ILLEGAL ADDRESS MODIFICATIONS: Ignored

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Refer to "Cache Mode Register (CMR)" and "Associative Memory" documented earlier in this manual.

2. When the Page Table Word Associative Memory (PTWAM) is enabled, the full or empty bit (PTW.F) of each of the 18 sets of associative memory registers is reset and the "round-robin" counter (RRO) is cleared to zero in each set.

   When the PTWAM is ON and effective address bits 16-17 = 00, 10, or 11, the PTWAM is cleared. If effective address bits 16-17 = 01, the PTWAM is turned OFF but is not cleared.

   When the PTWAM is OFF and effective address bits 16-17 = 10, the PTWAM is cleared. It is also turned ON if the PTWAM control switch on the VU control panel is ON. If effective address bits 16-17 = 00, 01, or 11, the PTWAM is not affected.

This instruction must be issued twice to enable and clear the PTWAM after it has been disabled.

3. An Illegal repeat causes an Illegal Procedure (IPR) fault.

4. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

****DPS 88 ONLY****

CAMPn	Clear Paging Associative Memory	53n (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode.

SUMMARY: CAMP0: In the executing processor, the PTW at the Paging Buffer entry location corresponding to bits 0-9 of the effective address is marked invalid.

CAMP1: In all processors having the same Control CIU and the same ICR, the PTW at the paging buffer entry location corresponding to bits 0-9 of the effective address is marked invalid.

CAMP2: All PTWs in the paging buffer of the executing processor are marked invalid.

CAMP3: All PTWs in the paging buffers of all processors having the same Control CIU and the same ICR are marked invalid.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The use of this instruction in other than Privileged Master mode causes an IPR fault.
2. For CAMP1 and CAMP3, the issuing processor must wait for an acknowledgement from the CIU. Because of the effect on performance, use of CAMP1 and CAMP3 should be limited to situations requiring this function.
3. This instruction must be gated under software control to ensure that no more than one processor allocated to the same operating system can execute a CAMP1 or CAMP3 at the same time.
4. For CAMP1 and CAMP3, processor port selection (which CIU) is determined by bit 23 (Control CIU) of the Option Register. The Interrupt Cell Register (ICR) is determined by bits 27-32 of the Option Register.
5. An Illegal Procedure fault occurs if illegal address modification is used.

CANA	Comparative AND with A-Register	315 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: 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 7-1)
PROCESSOR MODE:	Any
SUMMARY:	For n = 0, 1, ..., or 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
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 but executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification is used.

****DPS 8 ONLY****

CCAC	Clear Cache	011 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: 0 --->full/empty bits of Cache Directory

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Illegal repeats cause an IPR fault.
2. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.
3. Operand address development is allowed to proceed but has no effect on operation.

EXAMPLE:

```
        1      8       16              32
        ________________________________________________
               INHIB   ON
        *
        PINIT  TZE     DIS             not configured, park it
               LDQ     .CRCMC,AL,P.CR  is RLSEP active at this time
               CANQ    .FBT3,DU
               TNZ     WHOA            yes, park processor
               CANQ    .FBT4,DU        is it returning to service
               TZE     DIS             no, stop processing
               EAX7    0,AL            yes, set processor number
               REM     INITIALIZE CACHE
               LDP     P0,SD.RMS,DL
               LDQ     SD.PTD*2-SD.SLS*2+1,,P0
               QLS     10
               STQ     AQSAV
               LPDBR   AQSAV
               LCPR    NMODE,04
               XEC     .CRCSH,7,P.CR
               LCPR    .CRLUF,02,P.CR
               SCPR    RGSV0,01
               XEC     .CRCSH,7,P.CR
               CAMP                    clear associative memory
               CCAC                    clear cache
               LDP     P0,SD.PID,DL
               LDO     =0204000,DL     enable safe store and cache
               LDSS    .KLKPS,7*,KLS   load safe store for processor process
               LDWS    4,,P0           load WSR 0-3 and 4-7
               LDWS    5,,P0
               LXL1    .CRNPC,,P.CR
****
```

****DPS 88 ONLY****

CCAC0 CCAC1	Clear Cache and Flush	376 (1) 377 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode.

SUMMARY: For CCAC1
In the processor executing this instruction, all cache blocks which have the valid bit and the written bit ON are written to main memory.

For CCAC0 and CCAC1
0 ---> Valid bits of instruction and operand cache directories in the processor executing this instruction.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The use of this instruction in other than Privileged Master mode causes an IPR fault.
2. Operand address development is allowed to proceed but has no effect on operation.
3. Normally the software cache clearing will not be used; however, some mechanism is required to flush cache under exception conditions.
4. Attempted execution of the instruction CCAC, op code 011 (1), will be treated as a no-op.
5. If the memory hierarchy is operational, any use of CCAC0 must be preceded by the CCAC1 instruction in order to preserve system integrity.
6. An Illegal Procedure fault occurs if illegal address modification is used.

****

****DPS 8 ONLY****

CIOC	Connect Input/Output Channel	015 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Master Mode

SUMMARY: Sends a connect pulse through the channel that is specified by C(Y)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The CIOC instruction is the only input-output instruction in the system repertoire. The processor, having set up the I/O commands in memory, issues a CIOC instruction to the input/output multiplexer, which then assumes I/O responsibility.

2. The effective address Y is used to access a memory location. The memory module uses bits 33-35 of C(Y) to select one of its eight ports, sends a connect pulse to the unit on this port.

3. If the use of this instruction is attempted by a processor in the Slave mode, a Command fault occurs.

4. An Illegal Procedure fault occurs if illegal address modification is used.

EXAMPLE:

```
1      8       16              32
IOMCON STZ     IOMST,3         clear status word
       CIOC    .KLCIC,1,P5     connect to IOM
       EAX3    ,3              see if regular or alternate I/O
       TNZ     RESTA           if alternate, return
       EAX7    IOMST
       STX7    ISIRG           set up to return status at I/O complete
       STZ     IOSTS           reset important flags in case of fault
       STZ     INTWS
       TSX7    STARTA          start alternate I/O if possible
       ARG     TAPE2I
       TDCW    TPDCW
       TRA     IODIS           go wait for interrupt
```

****DPS 88 ONLY****

CIOC	Connect Input/Output Channel	015 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: The processor uses $C(A)_{18-35}$ with the contents of two internal registers, the Reserved Memory Base Register (RMBR) and the Connect Base Register (CBR), to develop the address of a Connect Table Word (CTW). See notes 1, 2, and 3.

If the coding in the CTW does not cause a fault (see notes 4 and 5) the processor forms two words of information to transmit to its control CIU. A Command Address Word (CAW) is transmitted, followed by a Connect Control Word (CCW).

Development of the CIOC instruction´s address field proceeds normally (effective address, virtual address, real address, physical address). The resulting physical address is placed in the CAW. When the IOX is the destination of the connect, this address becomes the Channel Mailbox address. See note 6.

The CIU action varies, depending on destination port steering information contained in the CCW. See note 7.

For an IOX central or channel-destined connect, a connect signal accompanies the CCW to the destination, followed by the CAW. See notes 8, 9, and 10.

For a processor-destined connect, only a connect signal is sent to the destination. See note 11.

For an SSF-destined connect, the CIU sends an Alarm signal to the SSF and buffers the CCW and CAW for subsequent transfer to the SSF. See note 12.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

BOUND FAULT: If $C(A)_{18-35}$ + C(Connect Base Register) > Reserved Memory Bound, a Bound fault shall occur.

COMMAND FAULT: If bit 0 of the Connect Table Entry is ON ($C(CTW)_0 = 1$), a Command fault occurs in the processor that executes the CIOC instruction.

HYPERMODE FAULT: If $C(CTW)_{0-1} = 01$ and HFER25 = 1, a CIOC Hypermode Entry fault occurs, causing entry into Hypermode.

NOTES:

1. The content of the A register is used in executing the CIOC instruction. C(A) must have the following format:

00 – 08	09 – 17	18 – 35
BECOMES CCW 0-8		OFFSET FROM CONNECT BASE TO CONNECT TABLE WORD

   See note 3 for more information about how bits 18-35 are used. See notes 9-12 for more information about how bits 0-8 are used.

2. The connect table provides dynamic determination of the set of processors and I/O channels with which the operating system communicates. If the destination is not configured to the OS issuing the connect, the entry in the connect table will be coded to cause either a Command fault or a CIOC Hypermode Entry fault. See notes 4 and 5.

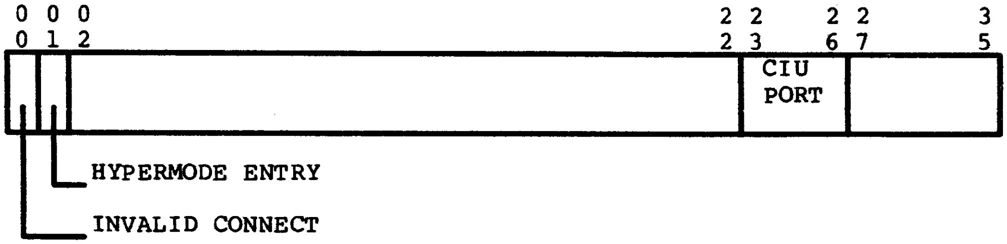
   The connect table also allows connects intended for a configured destination to be intercepted and redirected to another destination that is capable of receiving the connect.

   The connect table is located in reserved memory. The connect base register specifies the offset in words from the reserved memory base to the first word of the connect table. The connect table is initialized and maintained by SMAS software in the SSF. When multiple operating systems are configured on a DPS 88 system, each OS has its own reserved memory, and its own connect table.

Format of Connect Table:

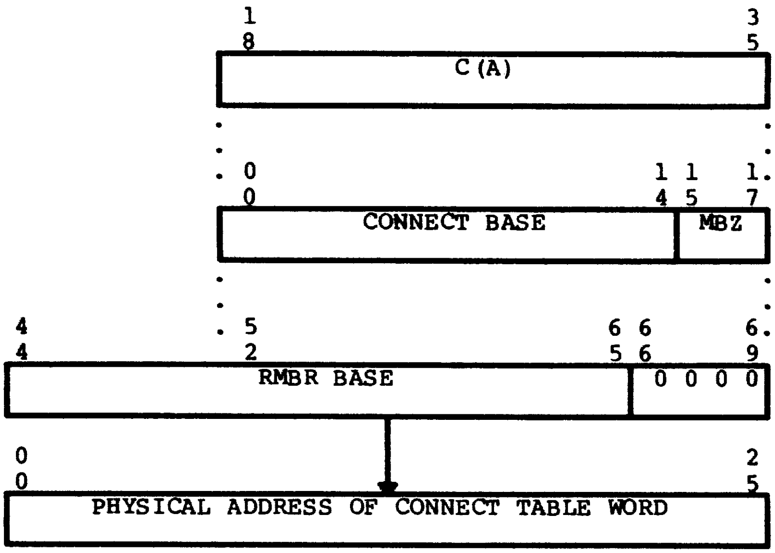
	00	35
0	HYPERSWITCHER ENTRY CONNECT	
1 2		
3	SSF CONNECT (PRIMITIVE COMMUNICATION)	
4	CONNECT TO CPU-0	
5	CONNECT TO CPU-1	
6	CONNECT TO CPU-2	
7	CONNECT TO CPU-3	
8 . . . 135	IOX-0 LOGICAL CHANNELS 0-127	
136 . . . 263	IOX-1 LOGICAL CHANNELS 0-127	
264 . . . 391	IOX-2 LOGICAL CHANNELS 0-127	
392 . . . 519	IOX-3 LOGICAL CHANNELS 0-127	

Each CTW has the following format:



The format of CTW bits 2-22 and 27-35 varies according to the destination of the connect. See notes 9-12.

3. The processor selects the appropriate CTW from the connect table by using $C(A)_{18-35}$ as an offset into the connect table from the connect base.



4. A 1 in bit 0 of the addressed CTW indicates that the processor or I/O channel at the destination has not been configured to the operating system (see note 2); the CIOC instruction terminates with a Command fault in the originating processor, and no other action is taken.

5. The hypermode fault enable register (HFER) provides one bit for each type of fault, and enables entry into the Hypervisor when any of the selected faults occurs. Bit 25 of the HFER allows the CIOC instruction to cause entry into hypermode via the CIOC Hypermode Entry fault. If bits 0-1 of the addressed CTW = 01 and bit 25 of the HFER = 1, then hypermode is entered through the hypervisor fault vector or entry descriptor, associated with the operating system´s reserved memory.

6. If $C(CTW)_0 = 0$, and either $HFER_{25} = 0$ or $C(CTW)_1 = 0$, the processor transmits the CAW and CCW to the control CIU, using bit 23 (Control CIU) of the processor option register to select the control CIU. The CAW and CCW are formed as shown:

00 – 08	09	10 – 35
CIU COMMAND 540_8	0	PHYSICAL ADDRESS FROM CIOC INSTRUCTION

CAW: COMMAND ADDRESS WORD

00 – 08	09 – 35
$C(A)_{0-8}$	$C(\text{CONNECT TABLE WORD})_{9-35}$

CCW: CONNECT CONTROL WORD

The address in bits 10-35 of the CAW is the physical address determined from the address field of the CIOC instruction, using tag and AR fields. Transposition of address bits to reflect interlacing is not performed. Subsequent use of this address by the IOX as the address of the channel mailbox will cause appropriate interlacing to be performed by the hardware.

7. If the ports are not masked between the originating processor and the control CIU, the CIU uses bits 23-26 of the CCW to select the port to which is sends the connect signal:

CCW Bits 23-26	2-Port CIU Port Selection	4-Port CIU Port Selection
0000	CIU Processor Port 0	CIU Processor Port 0
0001	CIU Processor Port 1	CIU Processor Port 1
0010	invalid	CIU Processor Port 2
0011	invalid	CIU Processor Port 3
01XX	invalid	invalid
1000	CIU CMPA Port 0 (IOX)	CIU CMPA Port 0 (IOX)
1001	CIU CMPA Port 1 (IOX)	CIU CMPA Port 1 (IOX)
1010	invalid	CIU CMPA Port 2 (IOX)
1011	invalid	CIU CMPA Port 3 (IOX)
1100	invalid	CIU CMPA Port 4 (SSF)
1101	invalid	invalid
1110	CIU CMPA Port 2 (SSF)	invalid
1111	invalid	invalid

CMPA = CIU Multiplex Port Adapter

If any port is masked between the originating processor and the destination, the destination port will not be notified of the connect, and the originating processor will receive no abnormal indication, except for lack of response from the destination.

8. If CCW bits 23-26 = 10XX, the destination is an IOX (see note 7). The CIU sends a connect signal to the IOX, along with the CCW and CAW. When the IOX receives a connect signal with the CCW and CAW, it examines bits 0, 1, and 9 of the CCW to determine the format of the CCW and the action to be taken. See notes 9 and 10.

9. When the IOX receives a connect signal with the CCW and CAW, in which bits 0, 1, and 9 of CCW ≠ 101, the CCW received by the IOX has the following format:

```
0 0 0 0 0 0 0 0 0 0 1 1           1 1       2 2       2 2 2 2 3 3 3     3
0 1 2 3 4 5 6 7 8 9 0 1           7 8       2 3       6 7 8 9 0 1 2     5
+-------------------+---------------------------------------------------+
|      C(A)         |                     C(CTW)                        |
|        0-8        |                      9-35                         |
+--+-+--------+-----+-+-+-----------+---------+-------+---+-+-+---------+
|  | |   TD   | 0 0 | | | CHANNEL   |         |       |   |X|I|         |
|TT|P+----+---+-----+H|0| NUMBER    |         |1 0 X X|0 0|P|O| SYS ID  |
|  | | 0 0|   SWP   | | |           |         |       |   |T|X|         |
+--+-+----+---------+-+-+-----------+---------+-------+---+-+-+---------+
```

TT	H	Action
00	X	Connect will be ignored by IOX
01	X	Channel-only connect. TD defines action. No status or interrupt.
10	0	IOXC-only. Read scratchpad word pair (SWP) of channel. CAW_{10-35} is the physical address where word pair from scratchpad gets stored.
11	X	Channel connect. CAW_{10-35} is the physical address of the Channel Mailbox. TD defines channel action.

TD Channel Action (TT = 01, 11):

0000	Standard connect
0001	Mask logical channel
0011	Mask adapter
0110	Mask adapter and reset controller
1000	Initiate DI channel

SWP Scratchpad Word Pair (TT = 10):

$SWP_5 = 0$

SWP_{6-8} = 000-111, specifying 1 of 8 zero-modulo-2 addresses in the IOXC's 16-word Command Word Processor (CWP) scratchpad for the logical channel specified by CCW_{10-17}. Each connect of this type reads the CWP scratchpad word pair specified by SWP_{6-8} and stores the word pair in the main memory location specified by CAW_{10-35}.

On completion of the two-word data transfer to main memory, no status is stored and no interrupt is sent.

This register command can be issued to a logical channel regardless of its busy state. If it is busy, the command will not interfere with the operation under way. Word 15 of the CWP scratchpad is used by this command and contains CAW_{10-35} and CCW_{29-35}. The IOX sets bit 26 = 0 to indicate non-hypermode.

P Paged (0) or nonpaged (1) operation

XPT This bit is saved by the IOX for use in selecting one of its two CIU ports for program interrupts. This is not a CIU number.

IOX These two bits are used only as a software reference, and are not used by DPS 88 hardware. The bits are saved in the IOX scratchpad and returned in the interrupt word when the I/O terminates.

SYS ID The SYS ID field is used by the IOX as an index into its hyperbase and hyperbound tables to obtain the hyperbase and hyperbound of the OS in forming physical addresses. The hyperbase and hyperbound tables are initialized and maintained by SMAS software in the SSF using the mechanism described in note 10.

The SYS ID field is saved in the IOX scratchpad for inclusion in the first interrupt word, where it is used by the CIU to select the interrupt cell register (ICR) that has been allocated to the OS.

10. When the IOX receives a connect signal with CCW and CAW in which bits 0, 1, and 9 of the CCW = 101, the CCW received by the IOX has the following format:

<table>
<tr><th>0 1
0 1</th><th>2 0
2 4</th><th>0 0
5 8</th><th>0
9</th><th>1 1
0 8</th><th>1 2
9 2</th><th>2 2
3 6</th><th>2 3
7 5</th></tr>
<tr><td colspan="3">C (A)
0-8</td><td colspan="5">C (CTW)
9-35</td></tr>
<tr><td>1 0</td><td>X X X</td><td>TI
0-3</td><td>1</td><td>TI
4-12</td><td>TD
0-3</td><td>1 0 X X</td><td>TD
4-12</td></tr>
</table>

CCW
Bits
2-4

000 Disable hyperaddressing. TI, TD not used.
100 Enable hyperaddressing. TI, TD not used.
X01 Hyperpage table change.

TI_{0-1} = 00 (reserved for future use).

TI_{2-12} specifies the address of a hyperPTW entry in the IOXC's 2K word hyperpage table.

TD_{0-12} is the data which is loaded into the hyperPTW entry addressed by TI_{2-12}. TD_0 = 1 indicates a missing hyperpage; if the hyperPTW is accessed with bit 0 = 1, the IOX reports an error.

TD_{1-12} specifies the high order 12 bits of the hyperpage address to be used in IOXC address development.

X10 Hyperbase table change.

TI_{0-8} = 0

TI_{9-12} = 0000-1111; in effect, this is the SYS ID field. It specifies the address of 1 of 16 entries in the IOXC's hyperbase table.

TD_{0-12} is the data which is loaded into the entry addressed by the TI field. TD_0 is set to zero.

TD_{1-12} specifies the high order 12 bits of the hyperbase address to be used in IOXC address development.

X11 Hyperbound table change.

TI_{0-8} = 0

TI_{9-12} = 0000-1111; in effect this is the SYS ID field. In specifies the address of 1 of 16 entries in the IOXC's hyperbound table.

TD_{0-12} is the data which is loaded into the entry addressed by the TI field. TD_0 is set to zero.

TD_{1-13} specifies the high order 12 bits of the hyperbound address which is checked against the high order 12 bits of the original "real memory address". An error occurs if the high order bits of the real memory address > hyperbound.

11. If CCW bits 23-26 = 00XX, the destination is a processor (see note 7). The CIU sends a connect signal to the destination processor, which causes a Connect Received fault to occur in the destination processor. The CIU does not send the CAW or the CCW to the destination processor. The CCW received by the CIU has the following format:

<table>
<tr><td>00 – 08</td><td>09 – 22</td><td>23 – 26</td><td>27 – 35</td></tr>
<tr><td>C(A)
0-8</td><td colspan="3">C(CTW)
9-35</td></tr>
<tr><td>0</td><td>0</td><td>0 0 X X</td><td>0</td></tr>
</table>

12. If CCW bits 23-26 = 11X0, the destination is an SSF attached to the SSF port on the CIU (not to be confused with an SSF attached to a DI channel of an IOX). See note 7. The CIU sends an Alarm signal to the SSF, and buffers the CCW and CAW:

00 08	09	10 22	23 26	27 31	32 35
C(A) 0-8	C(CTW) 9-35				
0	1	0	1 1 X 0	0	SYS ID

Bit 9 of the CCW is coded with a 1 to provide the SSF with a way to differentiate between connects through the SSF port of the CIU and connects through a DI channel of an IOX.

13. The use of this instruction in other than Privileged Master mode causes an IPR Fault.

14. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

****

CLIMB	Domain Transfer	713 (1)

FORMAT:

00 17	18 Op Code 27	28	29	30 35
ADDRESS	713(1)	I	AR	TAG

First Word

00	01 09	10 17	18 23	24 25	26 35
E	P	UNUSED	C	S	D

Second Word

The first word has the standard single-word instruction format (see Figure 7-1). The second word of the CLIMB instruction contains four control fields: C22-23, C18-19, E and P, and S and D. Bits 10-17 and 20-21 are not interpreted. The four control fields are defined as follows:

- $C_{22,23}$. Instruction Version

  This field determines one of the four 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. The processor state is saved (safe stored) if the SSBF flag of the option register = 1. If S,D = 0,1760, this is the PMME version (System Entry). If S,D ≠ 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.

  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 safestore frame off the safestore 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.

o C_{18}, X0 Control

The C_{18} bit allows the caller to load the effective address of the CLIMB instruction into X0 if C_{18} = 1 and if an entry descriptor is referenced during execution of the CLIMB. For OCLIMB, only the condition C_{18} = 1 is required to cause X0 to be loaded with the effective address of the CLIMB. If C_{18} = 0, X0 is not loaded, regardless of CLIMB version.

o C19, Slave Mode

For a CALL, LTRAS, or LTRAD, the C_{19} bit allows Slave mode to be set. For 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, 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 shall occur.

o E and P Argument Passing

The E and P fields are interpreted only for the CALL and 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 they are 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.

If E = 0, no parameters are passed.

In both cases the ASR is updated such that it locates the next available even-word location of the descriptor stack. The updated ASR bound field is set to zero and marked "not valid".

The E and P fields are not interpreted for the RET and LTRAS versions of the CLIMB instruction.

- S,D

  For CALL, LTRAS, or LTRAD, this field indicates from where the descriptor that determines the destination of the CLIMB is to come from (SEGID), or that the CLIMB is a System Entry (PMME).

  For Outward CLIMB (RET) this field is ignored.

PROCESSOR MODE: Any

SUMMARY: The CLIMB instruction provides a highly secure hardware mechanism for transferring control from one software function to another. This instruction performs the functions of call, return, and co-routine (goto) invocation for intra- and inter-instruction segments and intra- and inter-domain references.

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" above

NOTES:

1. Versions of the CLIMB instruction include:

Mnemonic	Meaning
ICLIMB (Inward CLIMB)	Change of domain call
PCLIMB (Lateral Transfer- LTRAD)	Change of domain transfer
GCLIMB (Lateral Transfer-LTRAS)	Change of domain transfer with passed arguments and parameters
OCLIMB (Outward CLIMB)	Change domain return
PMME (System Entry CLIMB)	Privileged Master mode entry

2. Interpretation of the fields 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 that identifies a descriptor specifying a new linkage segment and instruction segment or the same linkage segment and an instruction segment.

count - Decimal expression representing a value in the range $0 \le \text{count} \le 512$. This value indicates the number of vectors or descriptors (one for each argument). 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.

flags - 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.

The keyword EAX0 indicates that the effective address field is to be loaded in index register 0.

No flags are used for the OCLIMB version.

If both keywords are needed, the field must be enclosed by parentheses with a comma separating the keywords: (EAX0, SLAVE).

3. The instruction version to be executed is determined by bits 22 and 23 of the C field which are defined as follows:

00 ICLIMB or PMME - Functions as a call; that is, one segment transfers control to another segment to accomplish a task and upon completion of the task regains control. Additional descriptors may be prepared (shrunken) and placed in the argument segments; new parameter and argument segments are framed. The processor state is saved (safe stored) if bit 19 of the option register is 1.

11 PCLIMB - Functions in the same manner as ICLIMB, with new parameter and argument segments, except that the processor state is not saved.

10 GCLIMB - Functions as an unconditional transfer with the same parameters and arguments. The processor state is not saved.

01 OCLIMB - Functions as a return to the caller. The processor state is restored from the last frame of the safe store stack (see Figure 8-3).

NOTE: All versions of the CLIMB instruction make use of stack segments. ICLIMB means push the safe store stack, frame a new parameter segment, and open a new argument segment. OCLIMB means pop the safe store stack, and restore the parameter segment and argument segment to previous settings. PCLIMB and GCLIMB mean leave the safe store stack unchanged. GCLIMB leaves the parameter segment and argument segment registers unchanged, whereas PCLIMB activates a new parameter segment and argument segment in the same manner as ICLIMB.

4. Each version of the CLIMB instruction is described below:

ICLIMB (Inward CLIMB) - 00

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 $\bar{S}$ = 0 and D = 1760 (octal) define a PMME. If S = 0 and D = 1761 or 1763-1767 (octal), an IPR fault is generated.

If the CLIMB is a result of a fault or interrupt, this is an inter-domain transfer, requiring an entry descriptor, which is obtained from locations in the operating system as follows:

Interrupt:	30-31 octal
Fault:	32-33 octal

The referenced descriptor must be one of the following types:

a. Standard Descriptor (T = 0)

b. Descriptor Segment Descriptor (T = 1 or 3)

c. Entry Descriptor (T = 8, 9, or 11)

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 intra-domain 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, an STR fault occurs. If the obtained descriptor is not an entry descriptor nor 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 inter-domain transfer. The type of entry descriptor determines how much data (register contents) will be safe stored.

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.

The size of the safe store frame to be created and the data to be stored is determined by the referenced descriptor (the T field of the descriptor pointed to by the S,D field of the Inward CLIMB). The base of the safe store register (SSR) points to the starting address of the previous frame (the most recently stored frame that has not been returned to). Thus, before storing a new frame, the base address is incremented by the last frame size. A code is contained in the 2-bit stack control register (SCR) associated with the SSR that denotes the size of the last frame on the stack. (Note that the SCR is initialized to 11 binary when the LDSS instruction is executed.) The base and bound fields of the SSR are adjusted as follows:

SCR (binary)	Change to Base of SSR	Change to Bound of SSR
00_2	+ 16 words	- 16 words
01_2	+ 24 words	- 24 words
11_2	+ 64 words	- 64 words

The SSR base now points to the start of the new safe store frame to be created for the CLIMB instruction. The contents of the SCR are stored in the new safe store frame and the TEMP SCR is loaded according to the referenced descriptor as follows:

Field of Entry Descriptor (T Field)	SCR (binary)
0 or 8	00_2
9	01_2
11	11_2

The new SCR value determines the amount of data stored in the new safe store frame (16, 24, or 48-56 words). When the frame size is 64 words, the actual number of words stored is 48, unless indicator register bit 30 = 1 (multiword instruction interrupt). If the indicator register bit 30 = 1, the actual number of words stored is: ****DPS 8/70, 8/50, 8/52, and 8/62: 56 words; **** DPS 8/10 and 8/44: 52 words; **** DPS 88: 50 words. The following describes the contents of the safe store frame where all values are as at the beginning of the CLIMB instruction:

**** DPS 8 And Level 66:
Words 0-1 - Undefined ****

**** DPS 88:

Word 0

Bits 0-13 - Undefined

Bits 14-17 - ITC. Count of the number of indirect tally words updated in an indirect chain. If the count is 15_{10}, the processor cannot recover from a Missing Page fault; the operating system must handle the recovery or terminate the process.

Bits 18-27 - Opcode of the faulting instruction for faults other than Startup, Execute, ONC, Lockup, or MEMSYS. Not defined for interrupts, programmed CLIMBs, and the above five fault types.

Bit 28 - Interrupt inhibit bit of the faulting instruction. Not defined for interrupts, programmed CLIMBs, and Startup, Execute, ONC, Lockup, and MEMSYS faults.

Bit 29 - Address register bit of the faulting instruction. Not defined for conditions listed in Bit 28.

Bits 30-35 - Tag field of the faulting instruction. Not defined for conditions listed in Bit 28.

Word 1

Bits 0-15 - Undefined

Bits 16-17 - Zero

Bit 18 - IPR fault was caused by illegal opcode.

Bit 19 - IPR fault was caused by illegal tag, illegal execute, or illegal repeat.

Bit 20 - IPR fault was caused by illegal mode (i.e., attempted execution of Privileged Master Mode instruction while in Slave mode).

Bit 21 - Fault occurred during an instruction fetch.

Bit 22 - Zero

Bit 23 - Fault occurred during an attempted read of a descriptor from a linkage, argument or parameter segment; Opcode and tag are invalid.

Bit 24 - Undefined

Bit 25 - IPR fault was caused by illegal EIS digit, sign, MOP, etc.

Bits 26-29 - Undefined

Bits 30-34 - Indicators used by the processor in conjunction with words 2 and 3 to restart from certain faults which occurred during the execution of RPT, RPD, RPL, XEC, and XED. If equal to zero the Return CLIMB instruction ignores words 2 and 3 and no restart functions are performed.

Bit 35 - Demand Paging Recovery Flag (DPRF)
0 = Missing Page fault (not recoverable)
1 = Missing Page fault (recoverable)

****
**** DPS 8/70, 8/50, 8/52, and 8/62:

Word 2 - Undefined ****

**** DPS 8/20 and 8/44:

Word 2 - The faulting instruction when fault occurs ****

**** Level 66

Words 2-3 - The even/odd instruction pair when fault occurs ****

**** DPS 8/20, 8/44, 8/70, 8/50, 8/52, and 8/62:

Word 3 - Undefined ****

Word 4

Bits 0-17 - Programmed CLIMB
Instruction counter (IC) value for CLIMB plus 2
- Interrupt during multiword instruction
IC of first word of multiword instruction
- Interrupt after completed multiword or single word instruction
IC of next instruction
- Fault while attempting to fetch "Transferred To" instructions resulting from CLIMB
IC of "Transferred To" instruction
- Safestore stack fault on a programmed CLIMB
IC of "Transferred To" instruction
- Startup or Execute fault
IC undefined
- Op not complete, Lockup, Memory System faults
**** DPS 8: IC of faulting instruction + 1 ****
**** DPS 88: IC undefined ****
- Connect, Timer runout, or Shutdown faults during multiword instruction
IC of first word of multiword instruction
- Connect, Timer runout, or Shutdown faults after completed multiword or single word instruction
IC of next instruction

- Any other fault
  **** DPS 8: IC of faulting instruction + 1 ****
  **** DPS 88: IC of faulting instruction ****

Bits 18-32 - Contents of indicator register (IR)
Bits 33-35 - Undefined

Word 5

Bit 0 - Undefined
**** DPS 8/20 and 8/44: FRTRY - If zero, instruction cannot be retried, if 1, the instruction can be retried ****
Bits 1-7 - Undefined
Bit 8 - Undefined
**** DPS 88: RVA - Valid flag (1=valid) ****
Bit 9 - Undefined
**** DPS 8/70, 8/50, 8/52, 8/62: MRT - if 1, safe store is the result of an address trap, an opcode trap or a CU history register overflow ****
Bit 10 - Set to 1 if new SSR bound field is less than 192; Safe Store Stack fault occurs
Bit 11 - If CLIMB was due to a fault or programmed ICLIMB, bit 11 is 0; if due to interrupt, bit 11 is 1
Bits 12-16 - Fault or interrupt codes
- **** DPS 88: For interrupts and programmed CLIMB bits 12-16 = 0 ****
Bit 17 - Zero
Bit 18 - Undefined
Bits 19-21 - Processor number
Bits 22-23 - Stack control register (old value)
Bits 24-35 - Instruction segment identity register - SEGID(IS)

Word 6

Bits 0-16 - The value stored is the content of the data stack address register (DSAR) at the beginning of the CLIMB instruction **** DPS 88: Bits 0-14 ****
Bits 17-26 - Undefined
**** DPS 88: Bits 15-26 undefined ****
Bits 27-35 - Effective working space number when the fault occurred

Word 7 - Upon occurrence of an instruction fault, the relative virtual address when the fault occurred

Words 8-9 - Instruction segment register (ISR)

Words 10-11 - Argument stack register (ASR)

Words 12-13 - Linkage segment register (LSR)

Words 14-15 - Parameter stack register (PSR)

Words 16-23 - Address register (ARn) and segment identities

Words 24-39 - Descriptor registers (DRn)

Words 40-47 - SREG registers (index, accumulator, quotient, exponent, and timer registers)

NOTE: All register values stored reflect the contents of the register at the beginning of the CLIMB. If descriptors are pushed onto the argument stack during the CLIMB, the bound value of the safe-stored ASR is the value before the push occurred.

Words 48-55 - Pointer and length registers are stored in response to a fault or interrupt
**** DPS 88: Words 48-49 ****
**** DPS 8/20, 8/44: Words 48-51; mid instruction interrupt recovery data for firmware ****

Words 56-63 - Undefined
**** DPS 88: Words 50-63 undefined ****
**** DPS 8/20, 8/44: Words 52-63 undefined ****

Refer to Figure 8-3, Safe Store Stack, for a detailed diagram of the storage.

If field E of the second word of the CLIMB instruction is zero, then no descriptors are to be passed. If field E is 1, descriptors are to be passed and the action depends on the descriptor type contained in DR0 as described below (if DR0 contains a type T = 3, 5, or 7-15 descriptor, an IPR fault occurs):

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 will not prepare any descriptors but will frame these last P+1 descriptors with the parameter stack register. However, an STR fault occurs at this point if P+1 > DR0 bound field. ****DPS 88**** A BND fault occurs at this point if P+1 > DR0 bound field.

b. Descriptor Type in DR0 = 0, 2, 4, or 6

If the descriptor type contained in DR0 is 0, 2, 4, or 6, 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 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 = 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.

Although generating a data stack descriptor during the CLIMB instruction is permissible, clearing of the framed stack space is not allowed. Thus, 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 ignores the clear function.

With the state saved in the safe store stack, the registers are changed as follows:

a. Load the Linkage Segment Register (LSR)

(1) For an intra-domain transfer, the linkage segment does not change.

(2) For an inter-domain transfer, a standard descriptor from the entry descriptor is placed in the LSR as follows:

(a) Base = Linkage base (LBASE) with zeros in the 10 most significant bit positions

(b) Size = Linkage bound (LBOUND) extended with three 1 bits on the right and with zeros in the 7 most significant bit positions

(c) WSR = WSR (working space register)

(d) T = 1

(e) Flags - Bits 20, 22, 23, 27, and 28 = 1
Bits 21, 24, 25, and 26 = 0

****DPS 88: Although bit 23 is never interpreted by the hardware, it will be forced to 1 for software compatibility.****

b. Load the Instruction Segment Register (ISR)

(1) For an intra-domain transfer, the standard descriptor referenced by the S and D fields of the instruction is placed in the ISR. If S and D fields referenced a DRn (177n), then SEGIDn --> SEGID(IS); otherwise, S and D --> SEGID(IS).

(2) For an inter-domain 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.

c. Load the Instruction Counter (IC)

(1) For an intra-domain 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.

(2) For an inter-domain transfer, the 18-bit entry location contained in the entry descriptor is placed in the instruction counter.

d. Adjust the Argument Stack Register (ASR) and the Parameter Stack Register (PSR) as follows:

(1) If E bit = 0 (pass no parameters)

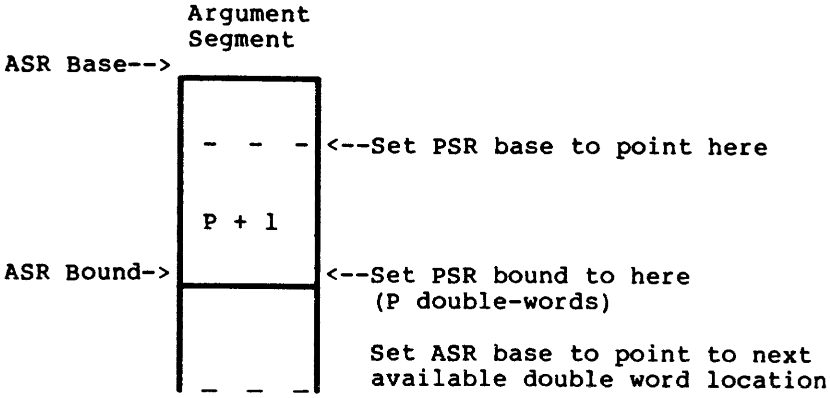
(a) Set PSR flag bit 27 to 0 to indicate bound not valid.

(b) If the current ASR flag bit 27 is 0, no change to ASR; the bound is not valid and the base does not change.

(c) If the current ASR flag bit 27 is 1, set the ASR base to point to the next available location by replacing ASR base with ASR base + (ASR bound + 1), and the ASR bound and flag bit 27 to zero.

(2) If E bit = 1 (pass parameters) and DR0 type = 0, 2, 4, or 6

(a) Using the current values from the ASR base and bound fields (bound field will have increased if descriptors were prepared by hardware; ASR flag bit 27 is 1), set the ASR and PSR as shown below:



(b) Copy the ASR flag field (with the exception of bit 27) to the PSR flag field. (This makes the PSR subordinate to the ASR; i.e., the PSR frames a portion of the space that was framed by the ASR and should not be allowed to grant additional privileges to the space control.) This also sets bit 27 of the PSR to 1, thereby indicating that it is not empty.

The new base and bound for the ASR are formed identically as described above for the case in which E bit is 0 and ASR flag bit 27 is 1.

(c) Set ASR flag bit 27 to 0 to indicate that the segment is empty and zero the bound field.

(3) If E bit = 1 and DR0 type = 1

(a) The descriptors to be framed by the PSR are the last P+1 descriptors in the descriptor segment pointed to by DR0.

(b) The ASR base and bound are adjusted exactly as described for the above case when the E bit is 0. Also, ASR flag bit 27 is set to zero.

(c) The new PSR base is set to the value DR0 base + DR0 bound - 2P

(d) The new PSR bound is set to (2P-1)

(e) The new base and bound values formed above 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.

e. Selectively Load Pointer Registers

If type 11 entry descriptor was referenced by the S and D fields of the CLIMB instruction, set all pointer registers to the value of the target IS as follows:

```
      ISR --> DR0 through DR7
SEGID (IS) --> SEGID0 through SEGID7
   0 - - 0 --> AR0 through AR7
```

f. Load SSR

```
If SCR = $11_2$
then
  SSR Base+$100_8$ words --> SSR Base
  SSR Bound-$100_8$ words --> SSR Bound

If SCR = $01_2$
then
  SSR Base+$30_8$ words --> SSR Base
  SSR Bound-$30_8$ words --> SSR Bound

If SCR = $00_2$
then
  SSR Base+$20_8$ words --> SSR Base
  SSR Bound-$20_8$ words --> SSR Bound
```

g. Load SCR: TEMP SCR --> SCR

h. Selectively Set Index Register 0

If bit 18 of the C field is 1 and if this is an inter-domain CLIMB instruction (an entry descriptor is involved), then the effective address for the instruction as generated is loaded into index register 0.

PCLIMB (Lateral Transfer - LTRAD) - 11

The execution of the PCLIMB version is identical with that of ICLIMB, except that the processor state is not saved in the safe store stack, the SCR remains unchanged, and the pointer registers are not set to the state of the instruction segment.

GCLIMB (Lateral Transfer - LTRAS) - 10

In the GCLIMB version of the CLIMB instruction, the safe store register and the parameter stack register remain unchanged. Also, the base and bound of the argument stack register remain unchanged.

The bit in the E field is not interpreted and the SCR remains unchanged.

The GCLIMB may be an inter- or intra-domain 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.

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 and the address registers are zero-filled.

OCLIMB (Outward CLIMB) - 01

In the OCLIMB version of the CLIMB instruction, a return occurs according to the last frame stored in the safe store stack.

The E, P, S, and D fields, and bits 19, 20, and 21 of the C field are ignored. The value of the stack control register (SCR) at the beginning of the OCLIMB determines the number and type of registers to be restored in addition to the following registers which are always restored.

a. Instruction counter (IC)

b. Indicator register (IR)

c. Stack control register (SCR)

d. Instruction segment identity register - SEGID (IS)

e. Data stack address register (DSAR)

f. Instruction segment register (ISR)

g. Linkage segment register (LSR)

h. Argument stack register (ASR)

i. Parameter stack register (PSR)

NOTE: If the Safe Store Bypass Flag, bit 19 (DPS 88: bit 3) in the option register, is zero, an IPR fault occurs.

When SCR = 00 (binary), all the normal checks are made before loading the listed registers from the safe store stack. If any test fails, the appropriate fault occurs.

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 eight descriptor registers, the eight index registers, and the A, Q, and E registers are restored. If bit 30 of the indicator register is 1, the pointer and length registers are also restored. (DPS 88: The pointer and length registers are restored unconditionally.)

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 (binary)	Base of SSR	Bound of SSR
00	-16	+16
01	-24	+24
10 or 11	-64	+64

If bit 18 of the C field is 1, the effective address loaded in index register 0. Control is transferred to the instruction pointed to by the instruction counter and the instruction segment register (ISR). When restoring the indicator register, the Master mode indicator bit may be turned ON.

If a fault occurs during execution of the OCLIMB the saved state will be the same as at the beginning of the OCLIMB, except that the values of the IR stored in the new safe store stack frame for the fault may be the values of the instruction being returned instead of the state of the indicators at the start of the OCLIMB.

PMME (System Entry CLIMB) - 00

In the PMME version of the CLIMB instruction, the system protected entry is activated for S = 0 and D = 1760 (octal). An entry descriptor is obtained from operating system location 34-35$_8$. The Master mode indicator bit is always set ON and bit 19 is ignored.

All modifications are allowed except DU, DL, CI, SC, and SCR.

The illegal repeats and executes are RPT, RPD, RPL, and XEC, XED.

Any of the following conditions cause an IPR fault:

a. If illegal repeats and executes precede modifications.

b. If the base and bound fields of the instruction segment descriptor are not modulo 32 bytes.

c. If the S and D fields are S = 0 and D = 1760 (octal), and the descriptor from the System Entry location is not an entry descriptor.

d. 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).

e. ·If the S and D fields of the vector or instruction are S = 0 and D = 1761 (octal).

A Command fault occurs 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.

Missing Segment and Missing Page faults may also occur.

The following conditions may cause an STR fault:

a. The ICLIMB version of the instruction if field E = 1 and either P + 1 > ASR bound field or P + 1 > descriptor register 0 bound field (the bound field to test P + 1 depends on descriptor type in descriptor register 0).

b. If flag bit 27 of the instruction segment descriptor is 0 (empty).

c. If a carry occurs in forming a new argument stack register (ASR) or parameter stack register (PSR) base.

A Security Fault, Class 2 occurs if flag bit 25 of the instruction segment descriptor is 0 (no execute permission).

Summary of CLIMB Instruction Format

00 02	03 17	18 26	27	28	29	30 35
	ADDRESS	713(1)	1	I	AR	TAG

Op Code (bits 18–26)

First Word

00	01 09	10 17	18	19	20 21	22 23	24 25	26 35
E	P	UNUSED	X0	SLV		TYP	S	D
36								72

Second Word

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 = 0 - Climb will not affect X0

X0 = 1 - If entry descriptor (T = 8, 9, or 11) is referenced or OCLIMB is executed, X0 is loaded with the effective address designated by the address tag and AR fields of the CLIMB instruction

SLV = 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 processcr state

TYP = 11 - PCLIMB (LTRAD) - Transfer with new ASR and PSR; Do not save processor state

S,D - Target SEGID

Coding Format:

```
 1      8       16
________________________________________________
        ICLIMB  entry,count,(ea),(flags)
        PCLIMB  entry,count,(ea),(flags)
        GCLIMB  entry,(ea),(flags)
        OCLIMB  (ea)
        PMME    (ea),count,(flags)
```

entry - SEGID - Least significant 12 bits are used

count - Number of parameters to pass, pointed to by PR0; If count field is specified, the assembler sets bit 0 of the second word of the instruction

ea - Effective address to be transferred to or loaded into X0; On OCLIMB, ea sets bit 18 of the second word

flags - EAX0 - Sets bit 18 of the second word.

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)

MASTER - Sets bit 19 of the second word

NOTE: PMME is synonymous with ICLIMB with 1760_8 coded in the entry field.

EXAMPLES:

```
1      8       16              32
________________________________________________________________
*                              ICLIMB
       INHIB   OFF
ODDF   NULL
NEPR1  LDD     P0,DSTKS        shrink data stack (64 words)
       SDR     P0
       LDD     P1,ODRSH        ODRS...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/,O9/713,1/1,1/0,1/0,6/M.
*      VFD     1/1,9/3-1,8/0,1/.N,1/.O,2/0,2/0,12/.DR+4
       .
       .
*                              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,O9/713,1/1,1/0,1/0,6/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/,O9/713,1/1,1/0,1/0,6/M.
*      VFD     1/0,9/0,8/0,1/.N,1/.O,2/0,2/0,12/.DR6
       TRA     0,,P0
```

CMG	Compare Magnitude	504 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: |C(A)| - |C(Y)| --> C(Z); C(A), 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

NOTES:

Zero	Negative	Relation
0	0	$\lvert C(A)\rvert > \lvert C(Y)\rvert$
1	0	$\lvert C(A)\rvert = \lvert C(Y)\rvert$
0	1	$\lvert C(A)\rvert < \lvert C(Y)\rvert$

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.

CMK	Compare Masked	211 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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

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

NOTES:

1. This instruction compares for identity those corresponding bit positions of A and Y that are not masked by a 1 in the corresponding bit position of Q.

2. The Zero indicator is set ON if the comparison is successful for all bit positions; i.e., if for all i = 0,1,...,35 there is

   either $C(A)_i = C(Y)_i$
   or
   $C(Q)_i = 1$

   Otherwise, the zero indicator is set OFF.

   The Negative indicator is set ON if the comparison is unsuccessful for bit position 0; i.e., if

   $C(A)_0 \neq C(Y)_0$
   and
   $C(Q)_0 = 0$

   Otherwise, the Negative indicator is set 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 ones 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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(A) - C(Y) --> C(Z); 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

Carry - If a carry out of bit 0 of C(Z) is generated, then ON; otherwise, OFF

NOTES:

1. Algebraic comparison (Signed Binary Operands)

Zero	Neg	Relation
0	0	C(A)>C(Y)
1	0	C(A)=C(Y)
0	1	C(A)<C(Y)

2. Logical comparison (Unsigned Positive Binary Operands)

Zero	Carry	Relation
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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(AQ) - C(Y-pair) --> C(Z); 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

Carry - If a carry out of bit 0 of C(Z) is generated, then ON; otherwise, OFF

NOTES: 1. Algebraic comparison (Signed Binary Operands)

Zero	Neg	Relation
0	0	C AQ)>C(Y-pair)
1	0	C(AQ)=C(Y-pair)
0	1	C(AQ)<C(Y-pair)

2. Logic comparison (Unsigned Positive Operands)

Zero	Carry	Relation
0	1	C(AQ)>C(Y-pair)
1	1	C(AQ)=C(Y-pair)
0	0	C(AQ)<C(Y-pair)

3. An Illegal Procedure fault occurs if illegal address modification is used.

CMPB	Compare Bit Strings	066 (1)

FORMAT:

00	01–10	11–17	18–27 Op Code	28	29–35
F	0---------------0	MF2	066(1)	I	MF1

<table>
<tr><th>00–02</th><th>03–17</th><th>18–19</th><th>20–23</th><th>24–32</th><th>32–35</th></tr>
<tr><td colspan="2">Y1</td><td rowspan="2">C1</td><td rowspan="2">B1</td><td colspan="2">N1</td></tr>
<tr><td>a1</td><td>Y1</td><td>0-------------------0</td><td>R1</td></tr>
</table>

<table>
<tr><th>00–02</th><th>03–17</th><th>18–19</th><th>20–23</th><th>24–32</th><th>32–35</th></tr>
<tr><td colspan="2">Y2</td><td rowspan="2">C2</td><td rowspan="2">B2</td><td colspan="2">N2</td></tr>
<tr><td>a2</td><td>Y2</td><td>0-------------------0</td><td>R2</td></tr>
</table>

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
```

PROCESSOR MODE: Any

SUMMARY: C(string 1) :: C(string 2)

The string of bits starting at location YCB1 is logically compared with the string of bits starting at location YCB2 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. ****DPS 88: If L1=L2=0, both the Zero and Carry indicators are turned ON****

2. An Illegal Procedure fault occurs if DU or DL modifications are used for MF1 or MF2.

EXAMPLES:

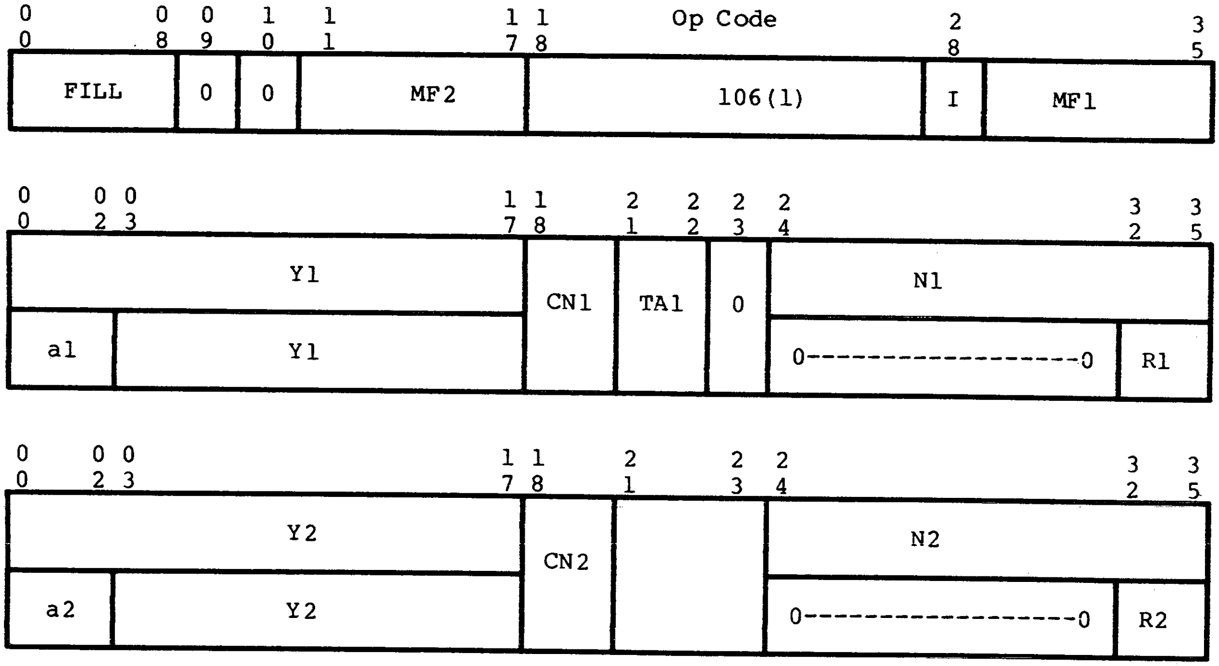
```
1      8       16              32
_________________________________________________________________
       CMPB    ,,1             fill bit 1 option
       BDSC    FLD1,45,0,0     FLD1 operand descriptor
       BDSC    FLD2,48         FLD2 operand descriptor
       TRC     EQU.GR          FLD1 equal/greater 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
```

PROCESSOR MODE: Any

SUMMARY: C(string 1) :: C(string 2)

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.

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=L2=0 both the Zero and Carry indicators are turned ON.

2. An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2.

3. **** DPS 88, DPS 8/20, and DPS 8/44: Depending on TA1, Bits 0-8, 3-8, or 5-8 of the FILL character are used to pad the least significant characters of the shorter string.****
**** DPS 8/70: Bits 0-8 (independent of TA1) of the FILL character are used to pad the least significant characters of the shorter string.****

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           ABCDb̸b̸
FLD2   BCI     2,XXXXABCDXXXX   ABCDb̸b̸
       USE                      Result - FLD1   =   FLD2
```

****DPS 88 ONLY****

CMPCT	Compare Characters and Translate	166 (1)

FORMAT:

00 08	09	10	11 17	18 Op Code 27	28	29 35
FILL	d1	d2	MF2	166(1)	I	MF1

00 17	18 20	21 22	23	24 35
Y1	CN1	TA1	0	N1

00 17	18 20	21 22	23	24 35
Y2	CN2	0	0	N2

00 17	18 28	29	30 31	32 35
Y3	0-------------------------0	AR3	0-0	REG

PROCESSOR MODE: Any

SUMMARY: 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. For the case 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, bits 0-8, 3-8, or 5-8 of the FILL character (depending on TA1) are used to pad the least significant characters of the shorter string. The contents of both strings remain unchanged.

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	
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. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CMPN	Compare Numeric	303 (1)

FORMAT:

<table>
<tr><td>0
0</td><td>1
0</td><td>1
1</td><td>1 1
7 8</td><td>Op Code</td><td>2
7</td><td>2
8</td><td>2
9</td><td>3
5</td></tr>
<tr><td colspan="2">0--------------0</td><td colspan="2">MF2</td><td colspan="2">303(1)</td><td>I</td><td colspan="2">MF1</td></tr>
</table>

<table>
<tr><td>0
0</td><td>0 0
2 3</td><td>1 1
7 8</td><td>2
0</td><td>2
1</td><td>22
23</td><td>2
4</td><td>2 3
9 0</td><td>3
5</td></tr>
<tr><td colspan="3">Y1</td><td rowspan="2">CN1</td><td rowspan="2">TN1</td><td rowspan="2">S1</td><td rowspan="2" colspan="2">SF1</td><td rowspan="2">N1</td></tr>
<tr><td>a1</td><td colspan="2">Y1</td></tr>
</table>

<table>
<tr><td>0
0</td><td>0 0
2 3</td><td>1 1
7 8</td><td>2
0</td><td>2
1</td><td>22
23</td><td>2
4</td><td>2 3
9 0</td><td>3
5</td></tr>
<tr><td colspan="3">Y2</td><td rowspan="2">CN2</td><td rowspan="2">TN2</td><td rowspan="2">S2</td><td rowspan="2" colspan="2">SF2</td><td rowspan="2">N2</td></tr>
<tr><td>a2</td><td colspan="2">Y2</td></tr>
</table>

CODING FORMAT: 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
```

PROCESSOR MODE: Any

SUMMARY: C(string 2) :: C(string 1)

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	$\lvert C(\text{number } 1)\rvert > \lvert C(\text{number } 2)\rvert$
1	1	$\lvert C(\text{number } 1)\rvert = \lvert C(\text{number } 2)\rvert$
0	1	$\lvert C(\text{number } 1)\rvert < \lvert C(\text{number } 2)\rvert$

NOTES:

1. An IPR fault occurs if any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.

   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.

2. 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
```

****DPS 88 ONLY****

CMPNX	Compare Numeric Extended	343 (1)

FORMAT:

00	01	02 10	11 17	18 Op Code 27	28	29 35
EA	NS	00-------------00	MF2	343(1)	I	MF1

00 17	18 20	21 22	23 24	29 30	35

00 17	18 20	21	22 23	24 29	30 35
Y1	CN1	TN1	SX1	SF1	N1

00 17	18 20	21	22 23	24 29	30 35
Y2	CN2	TN2	SX2	SF2	N2

PROCESSOR MODE: Any

SUMMARY: C(string 1) :: C(string 2)

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.

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 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. See 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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Q) - C(Y) --> C(Z); 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
Carry	- If a carry out of bit 0 of C(Z) is generated, then ON; otherwise, OFF

NOTES:

1. Algebraic comparison (Signed Binary Operands)

Zero	Neg	Relation
0	0	C(Q)>C(Y)
1	0	C(Q)=C(Y)
0	1	C(Q)<C(Y)

2. Logical comparison (Unsigned Positive Binary Operands)

Zero	Carry	Relation
0	1	C(Q) > C(Y)
1	1	C(Q) = C(Y)
0	0	C(Q) < C(Y)

CMPX<u>n</u>	Compare with Index Register <u>n</u>	10<u>n</u> (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0,1, ..., or 7 as determined by op code
$C(Xn)$ :: $C(Y)_{0-17}$; C(Xn) and C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of CMPX0

INDICATORS:

Zero - If C(Z) = 0, then ON; otherwise, OFF

Negative - If $C(Z)_0$ = 1, then ON; otherwise, OFF

Carry - If a carry out of $C(Z)_0$ is generated, then ON; otherwise, OFF

NOTES:

1. Algebraic (signed binary) comparison:

Zero	Neg	Relation
0	0	$C(Xn)>C(Y)_{0-17}$
1	0	$C(Xn)=C(Y)_{0-17}$
0	1	$C(Xn)<C(Y)_{0-17}$

2. Logical comparison (Unsigned Positive Binary Operands)

Zero	Carry	Relation
0	1	$C(Xn) > C(Y)_{0-17}$
1	1	$C(X\underline{n}) = C(Y)_{0-17}$
0	0	$C(X\underline{n}) < C(Y)_{0-17}$

3. DL modification is flagged as illegal but executes with all zeros for data.

4. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CNAA	Comparative NOT AND with A-Register	215 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: Any

SUMMARY: For i = 0 to 35, $C(Z)_i = C(Q)_i$ AND $\overline{C(Y)_i}$

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

CNAX$\underline{n}$	Comparative NOT AND with Index Register $\underline{n}$	20$\underline{n}$ (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: 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

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 is used.

CSL	Combine Bit Strings Left	060 (1)

FORMAT:

00	01–04	05–08	09	10	11–17	18–27 (Op Code)	28	29–35
F	0000	BOLR	T	0	MF2	060(1)	I	MF1

00–02	03–17	18–19	20–23	24–32	32–35
Y1		C1	B1	N1	
a1	Y1			0-------------------0	R1

00–02	03–17	18–19	20–23	24–32	32–35
Y2		C2	B2	N2	
a2	Y2			0-------------------0	R2

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
```

PROCESSOR MODE: Any

SUMMARY: C(string 1) : (BOLR) : C(string 2) --> C(string 2)

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; otherwise, OFF
**** DPS 88: If L2=0, then ON.****

Truncation - If L1 is greater than L2, then ON; otherwise, OFF
****DPS 88: If L1>0 and L2=0, then ON. If L1=L2=0, then OFF.****

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2.

2. ****DPS 88,DPS 8/20 and 8/44: The Zero and Truncation indicators are affected even if L1 and/or L2=0.****

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:

00	01–04	05–08	09	10	11–17	18–27 Op Code	28	29–35
F	0000	BOLR	T	0	MF2	061(1)	I	MF1

00–02	03–17	18–19	20–23	24–31	32–35
Y1	Y1	C1	B1	N1	N1
a1	Y1			0-------------------0	R1

00–02	03–17	18–19	20–23	24–31	32–35
Y2	Y2	C2	B2	N2	N2
a2	Y2			0-------------------0	R2

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
```

PROCESSOR MODE: Any

SUMMARY: C(string 1) : (BOLR) : C(string 2) --> C(string 2)

Same as for 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

NOTE: Illegal Procedure fault same as for CSL.

EXAMPLES:

```
1     8      16               32
FLD1  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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Y) :: closed (algebraic) interval [C(A), C(Q)] and with number C(Q); C(Y), C(A), C(Q) unchanged

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

Neg.	Carry	Relation	Sign
0	0	$C(Q) > C(Y)$	$C(Q)_0 = 0$, $C(Y)_0 = 1$
0	1	$C(Q) \geq C(Y)$	$C(Q)_0 = C(Y)_0$
1	0	$C(Q) < C(Y)$	
1	1	$C(Q) < C(Y)$	$C(Q)_0 = 1$, $C(Y)_0 = 0$

NOTE: 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). This instruction is not recommended for logical (unsigned) comparisons.

DFAD	Double-Precision Floating Add	477 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 greater than +127, then ON

Exponent Underflow - If exponent is less than -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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

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 7-1)

PROCESSOR MODE: Any

SUMMARY: $|C(E,AQ_{0-63})|$ :: $|C(Y\text{-pair})|$; magnitude comparison
C(EAQ), C(Y-pair) unchanged

ILLEGAL ADDRESS
MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	Neg.	Relation
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. 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.

   ****DPS8/20 and 8/44: If the number of shifts equals or exceeds 72 and if $|C(E,AQ_{0-63})| < |C(Y\text{-pair})|$, the processor fails to turn on the Negative indicator.****

   b. Compare the absolute values of the mantissas and set the indicators accordingly.

2. The DFCMG instruction is identical to the DFCMP instruction except that the magnitudes of the mantissas are compared instead of the algebraic values.

3. 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:

   **** DPS 8: Mode register, bit 33 ****
   **** DPS 88: Option register, bit 0 ****

4. 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 7-1)

PROCESSOR MODE: Any

SUMMARY: $C(E,AQ_{0-63})$ :: C(Y-pair); C(EAQ), C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	Neg.	Relation
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. 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.

2. The DFCMP instruction is identical to the FCMP instruction except for the precision of the mantissas actually compared.

3. 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:

   **** DPS 8: Mode register, bit 33 ****
   **** DPS 88: Option register, bit 0 ****

4. 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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Y-pair) ÷ C(EAQ) --> C(EAQ); C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

	If division occurs:	If no division occurs:
Zero	- If C(AQ) = 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 greater than +127, then ON	
Exponent Underflow	- If quotient exponent is less than -128, then ON	

NOTES:

1. 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.

2. ****DPS 88: Dividend and divisor are normalized by the hardware prior to division.****

   ****DPS 8: Dividend and divisor are not normalized by the hardware prior to division.****

3. **** DPS 8: If $AQ_{64-71} \neq 0$ and $A_0 = 0$, 1 is added to AQ_{63}. 0 --> AQ_{64-71}, unconditionally. AQ_{0-63} is then used as the divisor mantissa. The dividend exponent and mantissa are placed in working registers (8 and 72 bits, respectively). The dividend mantissa is shifted right, and the dividend exponent is increased accordingly until: $|\text{Dividend mantissa}| < |C(AQ)_{0-63}|$. When such a shift occurs, significant bits from the dividend may be lost. ****

4. ****DPS 88: $C(AQ)_{0-71}$ is used as the divisor mantissa.****

   ****DPS 8**** $C(AQ)_{0-63}$ is used as the divisor mantissa.****

5. ****DPS 88: 72 bits of quotient mantissa are placed in AQ.****

   ****DPS 8: 64 bits of quotient mantissa are placed in AQ_{0-63}. Zeros are placed in AQ_{64-71}.****

6. 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:

   ****DPS 8: Mode Register, bit 33 ****
   ****DPS 88: Option Register, bit 0 ****

7. 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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(EAQ) ÷ C(Y-pair) --> C(EAQ); C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

	If division occurs:	If no division occurs:
Zero	- If C(AQ) = 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 greater than +127, then ON	
Exponent Underflow	- If quotient exponent is less than -128, then ON	

NOTES:

1. 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.

2. ****DPS 88: Dividend and divisor are normalized by the hardware prior to division.****

   ****DPS 8: Dividend and divisor are not normalized by the hardware prior to division.****

3. ****DPS 8: The dividend mantissa C(AQ) is shifted right and the dividend exponent is increased accordingly until:
   $|C(AQ)_{0-72}| < |C(Y\text{-pair})_{8-71}$ with zero fill$|$.
   When such a shift occurs, significant bits from the dividend may be lost.****

4. $C(AQ)_{0-71}$ are used by this instruction.

5. ****DPS 88: 72 bits of quotient mantissa are placed in AQ.****

   ****DPS 8: 64 bits of quotient mantissa are placed in AQ_{0-63}. Zeros are placed in AQ_{64-71}.****

6. 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:

   ****DPS 8: Mode register bit 33 ****
   ****DPS 88: Option register, bit 0 ****

7. 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 7-1)

PROCESSOR MODE: 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.

DFMP	Double-Precision Floating Multiply	463 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -128, then ON

NOTES: This multiplication is executed as follows:

1. C(E) + $C(Y\text{-pair})_{0-7}$ --> C(E).
2. C(AQ) * $C(Y\text{-pair})_{8-71}$ results in a 134-bit product plus sign, the leading 71 bits plus sign of which --> C(AQ).
3. C(EAQ) normalized --> C(EAQ).
4. The definition of normalization is located under the description of the FNO instruction.

   When indicator bit 32=1 and 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

5. 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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(EAQ) rounded to 64 bits and normalized --> C(EAQ)

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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -128, then ON

NOTES:

1. 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.

2. 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.

3. The definition of normalization is located under the description of the FNO instruction.

4. 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

DFSB	Double-Precision Floating Subtract	577 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -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.

   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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0****

2. 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 7-1)

PROCESSOR MODE: Any

SUMMARY:

$C(E) \rightarrow C(Y\text{-pair})_{0-7}$
$C(AQ)_{0-63} \rightarrow C(Y\text{-pair})_{8-71}$
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 is used.

DFSTR	Double-Precision Floating Store Rounded	472 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: $C(EAQ)_{0-71}$ rounded, normalized --> C(Y-pair);
C(EAQ) unchanged

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 is greater than +127, then ON

Exponent Underflow - If exponent is less than -128, then ON

NOTES:

1. 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.

2. Except for precision, this instruction is identical with the FSTR instruction.

3. See the FRD instruction for the definition of true round.

4. The definition of normalization is located under the description of the FNO instruction.

   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:

   ****DPS 8: Mode Register, bit 33 ****
   ****DPS 88: Option register, bit 0****

5. An Illegal Procedure fault occurs if illegal address modification is used.

DIS	Delay Until Interrupt Signal	616 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: No operation takes place, and the processor does not continue with the next instruction, but waits for a program interrupt signal.

ILLEGAL ADDRESS MODIFICATIONS: None. The modification specified will be performed including the modification of any indirect words specified. However, the effective address will have no effect on the operation.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The inhibit bit in this instruction only affects the recognition of a Timer Runout (TROF) fault as follows:

   a. Inhibit ON causes the recognition of a TROF to be delayed until the processor enters Slave mode.

   b. Inhibit OFF allows the TROF to interrupt the DIS state.

2. For all other faults and interrupts, the inhibit bit is ignored. The use of this instruction in the Slave mode causes a Command fault.

   ****DPS 88: The use of this instruction in other than Privileged Master Mode causes an IPR fault.****

DIV	Divide Integer	506 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Q) ÷ C(Y)
integral quotient -> C(Q), right adjusted
integral remainder -> C(A), right adjusted
C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: None

INDICATORS: If division takes place:

Zero - If C(Q) = 0, then ON; otherwise, OFF

Negative - If bit 0 of C(Q) = 1, then ON; otherwise, OFF

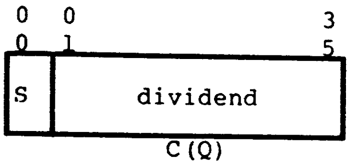
If no division takes place:

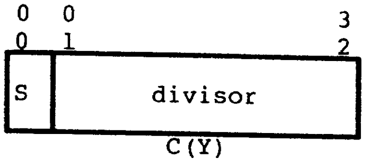
Zero - If divisor = 0, then ON; otherwise, OFF

Negative - If dividend < 0, then ON; otherwise, OFF

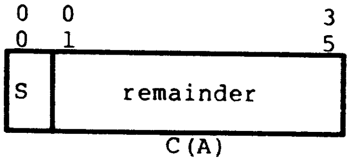
NOTES:

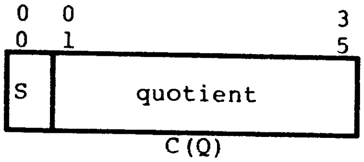
1. This instruction divides a 36-bit integral dividend (including sign) by a 36-bit integral divisor (including sign) to form a 36-bit integral quotient (including sign) and a 36-bit integral remainder (including sign). The remainder´s sign is equal to the dividend´s sign unless the remainder´s is zero.





yielding:





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.

2. ****DPS 88:
-2**35 (the most negative integer) divided by +1 results in the correct answer of A=0, Q=02**35.****

3. ****DPS 8:
If -2**35 (the most negative integer) is divided by +1 a Divide Check fault occurs, C(Y) remains unchanged, C(Q) contains the dividend magnitude, the Negative indicator reflects the dividend´s sign, and C(A) is set to zero.****

DRL	Derail	002 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Generates a DRL fault, which causes the processor to switch to Privileged Master mode to execute an Inward CLIMB instruction using the entry descriptor obtained from the word pair in

****DPS 8: Real memory location 32 octal ****
****DPS 88: Operating system memory location 32 octal ****

ILLEGAL ADDRESS MODIFICATIONS: None
**** DPS 8/70, 8/50, 8/52, 8/62: CI, SC and SCR generate an illegal condition that causes the history registers to be locked if mode register bit 31 = 1. No IPR fault occurs as the MME fault has higher priority. ****

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Master Mode - ON

NOTES:

1. 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.

2. The wired-in CLIMB instruction functions as though the second word of the CLIMB instruction had the following characteristics:

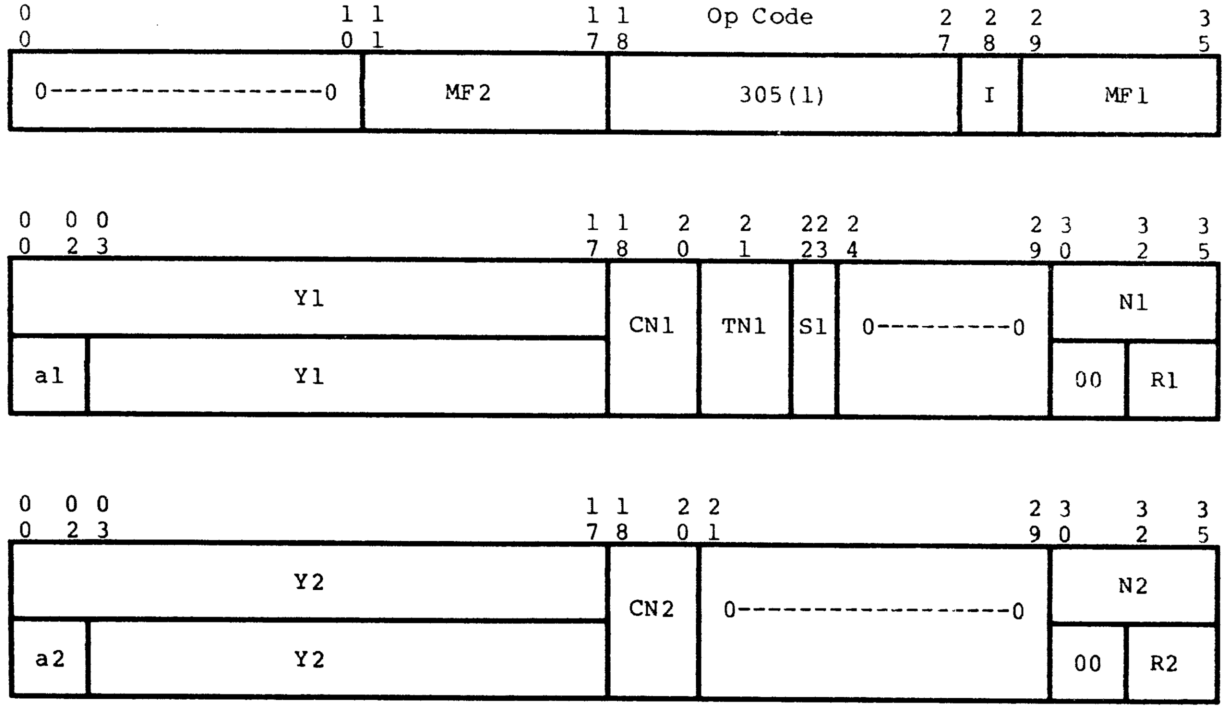
   E = 0 No parameters.
   C_{18} = 0 Do not load X0.
   C_{19} has no effect. Turn Master Mode indicator ON.
   C_{22-23} = 00 Inward CLIMB.
   S,D has no effect.

3. 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 (IC).

4. 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.

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
```

PROCESSOR MODE: Any

SUMMARY: $C(\text{string 1}) \xrightarrow{\text{converted}} C(\text{string 2})$

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 twos complement binary integer and stored, right-justified, as a character string of length L2 and 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. The length specified by L2 is given as the number of 9-bit segments that make up the length of the binary number to be stored and is equal to or is less than 8. Thus the stored binary number can be 9, 18, 27, 36, 45, 54, 63, or 72 bits long. 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 if DU or DL modifications are used for MF1 or MF2, if L2 is less than 1 or greater than 8, if CN2 does not contain a legal code, if S1 = 00, if illegal digit or sign is detected in string 1, or 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. ****DPS 8: If string 1 has the value -2**(9*L2-1), the result is zero and the overflow indicator is turned ON. ****

3. If string 1 contains more than 22 significant digits, an incorrect result is produced and the Overflow indicator is turned ON.

4. If the binary result is longer than L2 9-bit characters, the most significant nontruncated bit

   ****DPS 8: is forced to agree with the result sign ****

   ****DPS 88: may be different from 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 7-1)

PROCESSOR MODE: 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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****.

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 7-1)

PROCESSOR MODE: 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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -128, then ON

NOTES:

1. This multiplication is executed like the DFMP instruction, with the exception that the final normalization is performed only in the case of both factor mantissas being = -1.00...0.

2. Except for the precision of the mantissa of the operand from main memory, the DUFM instruction is identical to the UFM instruction.

3. 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

4. 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 7-1)

PROCESSOR MODE: 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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

2. An Illegal Procedure fault occurs if illegal address modification is used.

DV2D	Divide Using Two Decimal Operands	207 (1)

FORMAT:

00	01 – 09	10	11 – 17	18 – 27 Op Code	28	29 – 35
P	0------------0	RD	MF2	207(1)	I	MF1

00 – 17	18 – 20	21 – 22	23	24 – 29	30 – 35
Y1	CN1	TN1	S1	SF1	N1

00 – 17	18 – 20	21 – 22	23	24 – 29	30 – 35
Y2	CN2	TN2	S2	SF2	N2

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
```

PROCESSOR MODE: Any

SUMMARY: C(string 2) ÷ C(string 1) --> C(string 2)

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
```

****DPS 88 ONLY****

DV2DX	Divide Using Two Decimal Operands Extended	247 (1)

FORMAT:

00	01	02 09	10	11 17	18 Op Code 27	28	29 35
EA	NS	00---------00	RD	MF2	247(1)	I	MF1

00 17	18 20	21	22 23	24 29	30 35
Y1	CN1	TN1	SX1	SF1	N1

00 17	18 20	21	22 23	24 29	30 35
Y2	CN2	TN2	SX2	SF2	N2

PROCESSOR MODE: Any

SUMMARY: C(string 2) ÷ C(string 1) --> C(string 2)

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:

00	01	02–08	09	10	11–17	18–27 Op Code	28	29–35
P	0	MF3	0	RD	MF2	227(1)	I	MF1

00–17	18–20	21	22–23	24–29	30–35
Y1	CN1	TN1	S1	SF1	N1

00–17	18–20	21	22–23	24–29	30–35
Y2	CN2	TN2	S2	SF2	N2

00–17	18–20	21	22–23	24–29	30–35
Y3	CN3	TN3	S3	SF3	N3

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
```

PROCESSOR MODE: 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 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 scaled 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. The contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged. A Divide Check fault occurs under either of the following two conditions:

1. If the divisor is equal to zero. The divisor is the number starting at YC1.
2. If S3 specifies that the quotient be stored in scaled format and the calculated length required for the quotient is greater than 63 (see Note 2).

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

Exponent Overflow - If exponent of floating-point result is greater than 127, then ON; otherwise, unchanged

Exponent Underflow - If exponent of floating-point result is less than -128, then ON; otherwise, unchanged

Overflow - If fixed-point integer overflow, then ON; otherwise, unchanged

****DPS 8: If internal register overflow then ON; otherwise, unchanged ****

NOTES:

1. 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 described in Note 2. In floating-point operations the required number of quotient digits is determined as described in Note 3.

2. When the quotient descriptor specifies that the quotient is to be stored in scaled format, the necessary number of quotient digits to form is calculated as follows:

   #QD = (LD-#LZD+1)-(LDR-#LZR)+(ED-EDR-EQ)

   where:

   #QD = number of quotient digits to form

   LD = length of dividend

   #LZD = number of leading zeros in 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.

   ****DPS 88: If $\#QD \leq 0$, then zero is stored ****

3. ****DPS 8/70, 8/20, 8/44: In a floating-point divide operation 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.****

   ****DPS 88: In a floating-point divide operation with the divisor greater than the dividend, the algorithm generates a leading zero in the quotient. This characteristic of the algorithm is taken into account along with rounding requirements when determining the required number of digits for the quotient, so that the resulting quotient contains as many significant digits as specified by the quotient descriptor.****

4. An Illegal Procedure fault occurs if:

   a. DU or DL modification is specified for MF1 or MF2.

   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.

5. ****DPS 88: If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.****

   ****DPS 8: 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
```

****DPS 88 ONLY****

DV3DX	Divide Using Three Decimal Operands Extended	267 (1)

FORMAT:

						Op Code		
00	01	02 08	09	10	11 17	18 27	28	29 35
EA	NS	MF3	0	RD	MF2	267 (1)	I	MF1

00 17	18 20	21	22 23	24 29	30 35
Y1	CN1	TN1	SX1	SF1	N1

00 17	18 20	21	22 23	24 29	30 35
Y2	CN2	TN2	SX2	SF2	N2

00 17	18 20	21	22 23	24 29	30 35
Y3	CN3	TN3	SX3	SF3	N3

PROCESSOR MODE: Any

SUMMARY:

C(string 2) ÷ C(string 1) --> C(string 3)

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 scaled 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 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 EA. Placement of overpunched sign in the output is controlled by NS. If RD is a 1, the quotient is rounded prior to storage. The contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged. A divide check fault occurs under either of the following two conditions:

1. If the divisor is equal to zero. The divisor is the number starting at YC1.

2. If SX3 specifies that the quotient be stored in scaled format and the calculated length required for the quotient is greater than 63 (see Note 2 of DV3D).

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, or MF3

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.

****

DVF	Divide Fraction	507 (0)

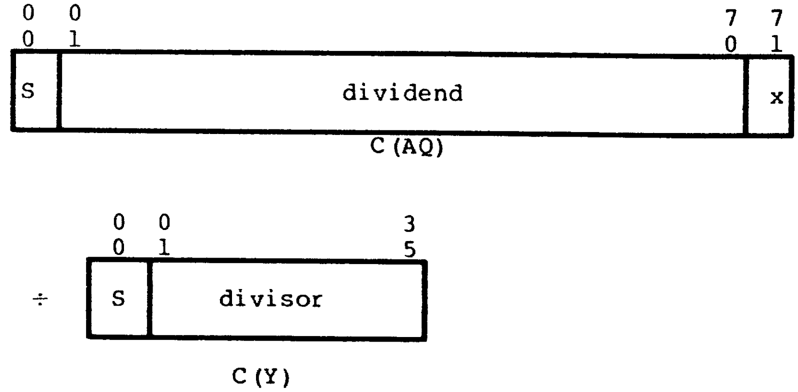
FORMAT:	Single-word instruction format (see Figure 7-1)
PROCESSOR MODE:	Any
SUMMARY:	C(AQ) ÷ C(Y) fractional quotient --> C(A), left adjusted fractional remainder --> C(Q), left adjusted C(Y) unchanged
ILLEGAL ADDRESS MODIFICATIONS:	None
ILLEGAL REPEATS:	None
INDICATORS:	If division takes place:

Zero	- If C(A) = 0, then ON; otherwise, OFF
Negative	- If $C(A)_0$ = 1, then ON; otherwise, OFF

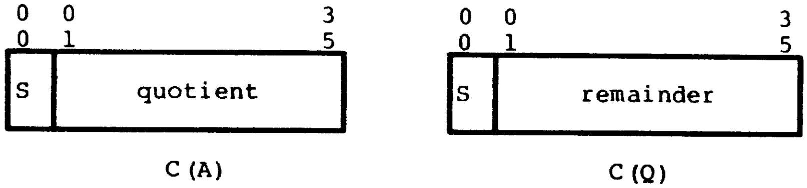
If no division takes place:

Zero	- If divisor = 0, then ON; otherwise, OFF
Negative	- If dividend < 0, then ON; otherwise, OFF

NOTE: 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 in absolute, and the Negative indicator reflects the dividend sign.

EAA	Effective Address to A-Register	635 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Y --> C(A)$_{0-17}$; 0...0 --> C(A)$_{18-35}$; C(Y) unchanged

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. This instruction facilitates inter-register data movement; the data source is specified by the address modification and the data destination by the operation code of the instruction.
2. An Illegal Procedure fault occurs if illegal address modification is used.

EAQ	Effective Address to Q-Register	636 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Y --> $C(Q)_{0-17}$; 0...0 --> $C(Q)_{18-35}$; C(Y) unchanged

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. This instruction facilitates inter-register data movement; the data source is specified by the address modification and the data destination by the operation code of the instruction.
2. An Illegal Procedure fault occurs if illegal address modification is used.

EAX<u>n</u>	Effective Address to Index Register <u>n</u>	62<u>n</u> (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0,1... or 7 as determined by opcode
Y --> C(X<u>n</u>); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL

ILLEGAL REPEATS: RPL
RPT or RPD of EAX0 cause IPR fault.

INDICATORS: Zero - If C(X<u>n</u>) = 0, then ON; otherwise, OFF

Negative - If $C(X\underline{n})_0$ = 1, then ON; otherwise, OFF

NOTES:

1. This instruction facilitates inter-register data movement; the data source is specified by the address modification and the data destination by the operation code of the instruction.

2. An Illegal Procedure fault occurs if illegal address modification is used.

EPAT	Effective Pointer and Address to Test	412 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: ****DPS 8: 000 --> C(test register 0)$_{0-2}$

Real memory address --> C(test register 0)$_{3-26}$

Effective working space number --> C(test register 0)$_{27-35}$

Relative virtual address --> C(test register 1)$_{0-35}$

C(Descriptor register (effective)) --> C(test registers 2 and 3)****

****DPS 88: 0 --> C(test register)$_{0}$

Real Memory Address --> C(test register)$_{1-26}$

Effective WSN --> C(test register)$_{27-35}$

Virtual Address$_{27-42}$ --> C(test register)$_{36-71}$ ****

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. This instruction provides the capability to test the real address preparation. All address preparation takes place in the normal sequence and the results are entered in the four special test registers instead of accessing memory. Refer to the STTA and STTD instructions for information concerning the special test registers.

   ****DPS 88: The EPAT instruction ignores the contents of the paging buffer and accesses the PTDW from memory.****

2. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, RPL cause an IPR fault.

3. ****DPS 88: The EPAT instruction has two functions. It can be used with the STTA instruction to test the address preparation and hardware in the processor. It can also be used for forming virtual addresses for the LDAT instruction. Both STTA and LDAT are privileged instructions.****

4. ****DPS 88: If WSN = 0, C(test register)$_{1-26}$ is undefined.****

5. **** DPS 8: An IPR fault occurs if the descriptor type is not 0, 2, 4, or 6. ****

6. An Illegal Procedure fault occurs if illegal address modification is used.

EPPRn	Effective Pointer to Pointer Register n	63n (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY:

```
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
    EA --> $C(ARn)_{0-23}$
    ****DPS 88:  IF $EA_{20-23}$ > 8 then 8 --> $C(ARn)_{20-23}$ ****
    $SEGID_m$ --> SEGIDn
    C(DRm) --> DRn
    m is selected by instruction bits 0,1,2
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
```

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. This set of eight instructions provides the capability of generating an effective pointer and storing it in a pointer register (a collective term referring to ARn, SEGIDn, and DRn).
2. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, RPL cause an IPR fault.

EXAMPLES:

```
1      8          16              32
NSGCHK XED        .CRNSG,,P.CR    test shut gates
       TZE        NSGOK           no
       LDA        .CRGID,7*,P.CR  get shut pointer
       CANA       =O3000,DL       is it system
       TNZ        SHSYS           yes
       LDP        P0,.SSR,DL      no, user
       LDD        P0,.WISR,,P0
       EPPR       P0,0,AU,P0      get gate pointer
       TRA        SHSYS+2


UNDO   EQU        *
       .CALL      .MSWAP,4        *if enabled, go unswap it
       INHIB      SAVE,ON
       EPPR0      *+3,$           sets return address
       TRA        .CRCAL,,P.CR
       ZERO       .MSWAP,4
       INHIB      RESTORE
```

ERA	EXCLUSIVE OR to A-Register	675 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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

EXAMPLE:

```
1      8      16
______________________________
       LDA    FLIP
       ERA    FLOP
       ERSA   FLIP
       ERSA   FLOP
```

ERAQ	EXCLUSIVE OR to AQ-Register	677 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: 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

ERSA	EXCLUSIVE OR to Storage with A-Register	655 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

ERSQ	EXCLUSIVE OR to Storage with Q-Register	656 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 is used.

EXAMPLE:

```
1         8         16
_____________________________________
          LDQ       =1,DL
          ERSQ      FLAG
```

ERSX<u>n</u>	EXCLUSIVE OR to Storage with Index Register <u>n</u>	64<u>n</u> (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: 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

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL
RPT or RPD of ERSX0

INDICATORS: Zero - If $C(Y)_{0-17} = 0$, then ON; otherwise, OFF

Negative - If $C(Y)_0 = 1$, then ON; otherwise, OFF

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

ERX<u>n</u>	EXCLUSIVE OR to Index Register <u>n</u>	66<u>n</u> (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: 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

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of ERX0

INDICATORS: Zero - If C(X<u>n</u>) = 0, then ON; otherwise, OFF

Negative - If $C(X\underline{n})_0$ = 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 is used.

FAD	Floating Add	475 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

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 7-1)

PROCESSOR MODE: Any

SUMMARY: $|C(E,AQ_{0-27})|$ :: $|C(Y)|$; magnitude comparison; C(EAQ), C(Y) unchanged.

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	Neg.	Relation
0	0	$\|C(E,AQ_{0-27}\| > \|C(Y)\|$
1	0	$\|C(E,AQ_{0-27}\| = \|C(Y)\|$
0	1	$\|C(E,AQ_{0027}\| < \|C(Y)\|$

NOTES:

1. 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 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.

      ****DPS 8/20 and 8/44: If the number of shifts equals or exceeds 72 and if $|C(E,AQ_{0-27})| < |C(Y)|$ the processor fails to turn on the Negative indicator ****

   b. Compare the absolute values of the mantissas and set the indicators accordingly.

2. The FCMG instruction is identical to the FCMP instruction except that the magnitudes of the mantissas are compared instead of the algebraic values.

3. 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

4. An Illegal Procedure fault occurs if illegal address modification is used.

FCMP	Floating Compare	515 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: $C(E,AQ_{0-27})$ :: C(Y); algebraic comparison

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	Neg.	Relation
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. 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 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.

   b. Compare the mantissas and set the indicators accordingly.

2. 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

3. An Illegal Procedure fault occurs if illegal address modification is used.

FDI	Floating Divide Inverted	525 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Y) ÷ C(EAQ) --> C(EA); 00...0 --> C(Q)

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 greater than +127, then ON

Exponent Underflow - If exponent is less than -128, then ON

NOTES:

1. 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.

2. ****DPS 88: Dividend and divisor are normalized by the hardware prior to division ****

   ****DPS 8: Dividend and divisor are not normalized by the hardware prior to division ****

3. ****DPS 8: If $AQ_{28-71} \neq 0$ 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 dividend exponent and mantissa are placed in working registers (8 and 72 bits, respectively).

   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. ****

4. ****DPS 88: $C(AQ)_{0-71}$ is used as the divisor mantissa ****

   ****DPS 8: $C(AQ)_{0-27}$ is used as the divisor mantissa ****

5. 36 bits of quotient mantissa are placed in A.

6. 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

7. An Illegal Procedure fault occurs if illegal address modification is used.

FDV	Floating Divide	565 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(EAQ) ÷ C(Y) --> C(EA); 00...0 --> C(Q); C(Y) unchanged

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 greater than +127, then ON	
Exponent Underflow	- If exponent is less than -128, then ON	

NOTES:

1. If the divisor mantissa (bits 8-35 of C(Y)) is zero, the division does not take place. Instead, a Divide Check fault occurs. The divisor C(Y) remains unchanged, C(AQ) contains the dividend's magnitude in absolute, and the Negative indicator reflects the dividend's sign.

2. ****DPS 8: This division is executed as follows:

   The dividend mantissa C(AQ) is shifted right and the dividend exponent C(E) is increased accordingly until

   $|C(AQ)_{0-71}| < |C(\text{Y-pair})_{8-35} \text{ with zero fill}|$.

   When such a shift occurs, significant bits from the dividend may be lost. ****

3. ****DPS 88: Dividend and divisor are normalized by the hardware prior to division ****

4. ****DPS 8: Dividend and divisor are not normalized by the hardware prior to division ****

5. ****DPS 88: $C(AQ)_{0-71}$ is used by this instruction ****

6. 36 bits of quotient mantissa are placed in A.

7. 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

8. An Illegal Procedure fault occurs if illegal address modification is used.

FLD	Floating Load	431 (0)

FORMAT:	Single-word instruction format (see Figure 7-1)
PROCESSOR MODE:	Any
SUMMARY:	$C(Y)_{0-7}$ --> C(E) $C(Y)_{8-35}$ --> $C(AQ)_{0-27}$ 00..0 --> $C(AQ)_{28-17}$
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.

FMP	Floating Multiply	461 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -128, then ON

NOTES:

1. 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).

2. The definition of normalization is located under the description of the FNO instruction.

3. 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

4. An Illegal Procedure fault occurs if illegal address modification is used.

FNEG	Floating Negate	513 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: -C (EAQ) normalized --> C(EAQ)

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 greater than +127, then ON

Exponent Underflow - If exponent is less than -128, then ON

NOTES:

1. This instruction changes the number in C(EAQ) to its normalized negative (if C(AQ) $\neq$ 0). The operation is executed by first forming the twos complement of C(AQ), and then normalizing C(EAQ).

2. Even if C(EAQ) is already normalized, an exponent overflow can still occur, namely when C(E) = +127 and C(AQ) = 100...0 which is the twos complement representation for the decimal value -1.0.

3. The definition of normalization is located under the description of the FNO instruction.

4. When indicator bit 32 = 1 and the Hex Permission Flag = 1, the floating-point alignment and normaliz@ation are hexadecimal. Otherwise, the floating-point alignment and normalization are binary. The Hex Permission Flag is:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

FNO	Floating Normalize	573 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(EAQ) normalized --> C(EAQ)

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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -128, then ON
Overflow	- Set OFF

NOTES:

1. The instruction normalizes the number in C(EAQ). If the Overflow indicator is ON, then 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.

   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).

2. 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 $C(AQ)_0 \neq C(AQ)_1$

3. 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 $C(AQ)_{1-4} \neq 1111$

4. 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.

5. If $C(AQ)=_0$, then C(E) is set to -128 and the zero indicator is set ON.

6. 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

FRD	Floating Round	471 (0)

FORMAT:	Single-word instruction format (see Figure 7-1)
PROCESSOR MODE:	Any
SUMMARY:	C(EAQ) rounded to 28 mantissa bits and normalized --> C(EAQ)
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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -128, then ON

NOTES:

1. 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.

2. If the mantissa overflows upon rounding, it is shifted right one place and a corresponding correction is made to the exponent.

3. If the mantissa does not overflow and is nonzero upon rounding, normalization is performed.

4. If the resultant mantissa is all zeros, the exponent is forced to -128 and the Zero indicator is set.

5. If the exponent resulting from the operation is greater than +127, the Exponent Overflow indicator is set.

6. If the exponent resulting from the operation is less than -128, the Exponent Underflow indicator is set.

7. The definition of normalization is located under the description of the FNO instruction.

8. 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

FSB	Floating Subtract	575 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option.register, bit 0 ****

3. An Illegal Procedure fault occurs if illegal address modification is used.

FST	Floating Store	455 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY:

$C(E) \text{ --> } C(Y)_{0-7}$
$C(A)_{0-27} \text{ --> } C(Y)_{8-35}$
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 is used.

FSTR	Floating Store Rounded	470 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(EAQ) rounded and normalized --> C(Y); C(EAQ) unchanged

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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -128, then ON

NOTES:

1. 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.

2. Upon completion of the rounding and normalization, the exponent and truncated mantissa are stored as follows:

   Exponent in bits 0-7 of C(Y)
   Bits 0-27 of mantissa in bits 8-35 of C(Y)

   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).

3. The rounding, then normalization operation of this instruction is identical with FRD.

4. The definition of normalization is located under the description of the FNO instruction.

5. 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:

   ****DPS 8: Mode register, bit 33 ****
   ****DPS 88: Option register, bit 0 ****

6. An Illegal Procedure fault occurs if illegal address modification is used.

FSZN	Floating Set Zero and Negative Indicators from Storage	430 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Test C(Y); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	Neg.	Relation
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.

GTB	Gray-to-Binary Convert	774 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(A) is converted from Gray code to a 36-bit binary number

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

NOTES:

1. 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:

   $S_0 = R_0$

   $S_1 = (R_i \text{ AND } \overline{S_{i-1}}) \text{ OR } (\overline{R_i} \text{ AND } S_{i-1})$

   where: i = 1,...,35.

2. 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 four-word, two-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.

LARn	Load Address Register n	76n (1)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1          8          16
_________________________________________________

           LARn       LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: For n=0,1.., or 7 as determined by op code
$C(Y)_{0-23}$ --> 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 7-1)

CODING FORMAT:

```
1      8      16
       LAREG  LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: $C(Y,Y+1,...,Y+7)_{0-23}$ --> C(AR0,AR1,...,AR7)

The hardware assumes bits 15-17 of Y = 000. No check is made.

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. 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.

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 7-1)

PROCESSOR MODE: 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
Overflow	- If range of A is exceeded, then ON

NOTE: This instruction changes the number to its negative (if ≠ 0) while moving it from Y to A. The operation is executed by forming the twos complement of the string of 36 bits. An overflow condition exists if C(Y) = 2**35.

LCAQ	Load Complement into AQ-Register	337 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: -C(Y-pair) --> (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

NOTES:

1. 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 twos complement of the string of 72 bits. An overflow condition exists if C(Y)-pair) = -2**71.

2. An Illegal Procedure fault occurs if illegal address modification is used.

****DPS 88 ONLY****

LCCL	Load Calendar Clock	674 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: $C(AQ)_{0-71}$ --> $C(\text{Calendar Clock})_{0-71}$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None

NOTES:

1. Address development is allowed to proceed but has no effect on the loading of the Calendar Clock.

2. Processor port selection (which CIU) is determined by bit 23 (Control CIU) of the Option Register. This Control CIU bit can be changed by the SSF, or by the LDHC instruction in Hyper mode, if reconfiguration requires the use of the Calendar Clock in the other CIU.

3. The LCCL instruction loads the Calendar Clock in the Control CIU. The 72-bit binary value covers a period of $2^{72}-1$ microseconds, which is more than 149 million years. The Calendar Clock increments by one every microsecond.

4. The Calendar Clock is initially loaded by the SSF (SMAS) with a value that is the number of microseconds that have elapsed since 00:00 hours, Greenwich Mean Time (GMT), January 1, 1901. When an operating system loads the Calendar Clock with the LCCL instruction, the value loaded should represent GMT since the SSF will resync its clock to this newly loaded value, and expects the value to represent GMT.

5. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

****DPS 8 ONLY****

LCPR	Load Central Processor Register	674 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(Y) --> C(register specified by tag field); C(Y) unchanged

Tag

02 C(Y)$_{0-17}$ Ignored
****DPS 8/20 and 8/44: C(Y)$_{18-24}$ --> Cache Mode Register****
C(Y)$_{25-33}$ Ignored
C(Y)$_{34-35}$ --> Lockup Fault Register

03 0...0 --> History Registers

04 C(Y) --> Mode Register

07 1...1 --> History Registers

11 ****DPS 8/20 and 8/44: C(Y)$_{0-17}$ --> Configuration Register****
C(Y)$_{18-35}$ Ignored

12 C(Y)$_{0-26}$ --> Address Trap Register
C(Y)$_{27-35}$ Ignored

13 C(Y)$_{0-5}$ Ignored
C(Y)$_{6-12}$ --> Fault Base Register
C(Y)$_{13-33}$ Ignored
C(Y)$_{34-35}$ --> C.P. Number Register

15 Bits 7-15 of Effective Address --> Cache Directory Address Register (Specifies Column and Level to be Loaded)
C(A) --> Cache Directory (Level OFF Flag is Set and RRO Counter is Cleared)

17 Bits 11-17 of Effective Address --> Associative Memory Address Register (Specifies Column and Level to be Loaded)
C(A) --> Associative Memory Directory

ILLEGAL ADDRESS MODIFICATIONS: Tag field defines register.

ILLEGAL REPEATS: RPT, RPD, RPL cause IPR fault

INDICATORS: None affected

NOTE: The use of this instruction in Slave mode causes a Command fault.

EXAMPLE:

```
1      8      16            32
       LDA    =3,DL         Set lockup timer to 16 ms on DPS 8/70
       STA    ***           .
       LCPR   ***,02        .
```

****

LCQ	Load Complement into Q-Register	336 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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
Overflow	- If range of Q is exceeded, then ON

NOTE: This instruction changes the number to its negative (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.

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 7-1)

PROCESSOR MODE: Any

SUMMARY: For n=0,1.... or 7 as determined by opcode
$-C(Y)_{0-17}$ --> (Xn); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of LCX0

INDICATORS:

Zero - If C(Xn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of Xn is exceeded, then ON

NOTES:

1. This instruction changes the number to its negative (if ≠ 0) while moving it from bits 0-17 of Y to Xn. The operation is executed by forming the twos complement of the string of 18 bits. An overflow condition exists if $C(Y)_{0-17}$ = -2**17.
2. DL modification is flagged as illegal but executes with all zeros for data.
3. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

LDA	Load A-Register	235 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Y) --> C(A); 0...0 --> C(Y)

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

NOTES:

1. The LDAC instruction should only be used for gating purposes. It should not be used as a substitution for an LDA, STZ pair because of the performance penalty that is introduced.

2. ****DPS 88: LDAC, LDQC, SZNC, STAC, and STACQ are the only instructions that can be used for the indivisible test-and-set operations which are required for setting and releasing locks, or for closing and opening gates.

   Since execution of LDAC, LDQC, SZNC, STAC, and STACQ depends on the previous C(Y), the processor will obtain ownership of the 8-word block containing C(Y) prior to using C(Y) to execute the instruction. Obtaining ownership of the 8-word block means that the requesting processor, and the Memory Hierarchy Control of the CIU, will ensure that a valid copy of the block is obtained, and that the block is cleared from the cache of all other processors before the instruction is executed. After obtaining ownership of the block the processor completes execution of the instruction to set or release the lock without permitting the block to be siphoned to another processor. Thus the block is isolated in a time window where it can be accessed and modified only by the processor executing the instruction which sets or releases the lock.

To ensure that a lock does not get released before the actual completion of all stores performed while the lock was set, a synchronizing function is accomplished by coding a SYNC or STC2 instruction immediately before the instruction which releases the lock. If the value stored by STC2 is consistent with operating system conventions for a released lock, then the use of STC2 for synchronizing can also serve to release the lock.****

3. An Illegal Procedure fault occurs if illegal address modification is used.

LDAQ	Load AQ-Register	237 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(Y-pair) --> C(ASR); C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. 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 8 bytes (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).

2. If these conditions are met, the descriptor is loaded into the argument stack register (ASR). 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.

3. Any of the following conditions cause an IPR fault:

   a. Modifications DU, DL, CI, SC, and SCR.

   b. Illegal repeats RPT, RPD, and RPL.

   c. Type field T is not 1.

   d. If the base and bound limits of the operand descriptor are not modulo 8 bytes (subject to flag bit 27).

4. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault (DPS 88: IPR 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
```

****DPS 88 ONLY****

LDAT	Load Address Trap Register	336 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY:

If EA(17) = 0, then
C(Test Reg.)$_{27-35}$ --> C(Address Trap Reg. 0)$_{0-8}$
0 --> C(Address Trap Reg. 0)$_{9}$
C(Test Reg.)$_{44-69}$ --> C(Address Trap Reg. 0)$_{10-35}$

If EA(17) = 1, then
C(Test Reg.)$_{27-35}$ --> C(Address Trap Reg. 1)$_{0-8}$
0 --> C(Address Trap Reg. 1)$_{9}$
C(Test Reg.)$_{44-69}$ --> C(Address Trap Reg. 1)$_{10-35}$

If EA(15) = 1
then IPR fault on any attempted READ of the address

If EA(16) = 1
then IPR fault on any attempted WRITE of the address

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The processor provides two 36-bit Address Trap Registers that allow software to dynamically monitor READ/WRITE references to two locations defined by their virtual addresses.

   The LDAT instruction loads address trap register 0 or 1, as a function of EA(17), from bits 27-35 and 44-69 of the text register loaded by the EPAT instruction immediately preceding the LDAT instruction. These bits from the test register are the virtual address from the EPAT instruction, and represent the working space number, page number, and word in a working space that is mapped via a dense page table.

   The address trap registers should not be loaded from the virtual address of an EPAT instruction referencing a working space that is mapped via a fragmented page table.

2. To disable an Address Trap register, both the READ and WRITE flags for that register must be set to zero, i.e., EA(15-16) = 00.

3. Upon successfully "trapping" an address, the processor will execute an IPR fault entry into the Operating System.

4. The use of this instruction in other than Privileged Master Mode causes an IPR fault.

5. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

****

LDDn	Load Descriptor Register n	67n (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: 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 instructions have a direct load option and a vector option.

DIRECT LOAD OPTION
When the DRn is loaded from a descriptor segment, the contents of even and odd locations Y and Y+1 of the segment identified by DRm are loaded directly into DRn with no modification.

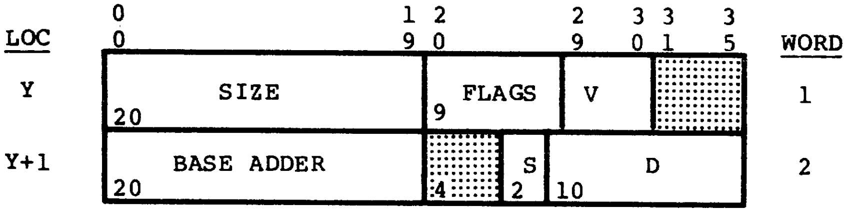
The ARn and SEGIDn registers are affected by this sequence of the instruction as follows:

a. ARn is set to all zeros.

b. SEGIDn is set to be self-identifying (S = 0, D = 177n) as described in next sequence.

VECTOR OPTION
When the effective address of the instruction operand is within a data segment the function performed by this instruction is specified by the vector at the even and odd locations Y and Y+1. The vector has the following format:

VECTOR



The shaded portion of the words is not interpreted and may contain any bit pattern.

The V field (bits 29 and 30) specifies the function to be performed.

V Field (Bits 29 and 30)	Function
00	Copy Copy selected descriptor into DRn; set SEGIDn to indicate where the descriptor came from; zero-fill ARn
01	Normal Shrink Shrink selected descriptor as indicated and load into DRn; set SEGIDn to indicate DRn as the source; zero-fill ARn
11	Data Stack Shrink Form a working data stack descriptor using DSDR and DSAR; load new descriptor into DRn; update DSAR; zero-fill ARn; set SEGIDn to indicate DRn as the source (DPS 8: conditionally clear address space framed by new descriptor).

COPY
Vector Type = 00 (See also LDPn instruction)

The S and D fields of the vector specify the descriptor to be loaded into DRn as follows:

When S = 0:

For D = 0000 through 1757 (octal) and D≤ PSR bound, the descriptor is loaded from the parameter stack 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.

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 Stack 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 the 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 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.

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 vector.

c. If an IPR or an STR (DPS 88: BND) fault occurs, DRn, ARn, and SEGIDn are not changed.

SHRINK
Vector Type = 01

The descriptor identified by the vector (as indicated for copy) is obtained, shrunk as indicated, and loaded into DRn. In Privileged Master mode for S = 0 and D = 1761 (octal), a type 1 or 3 descriptor is changed to 0 or 2 respectively, and shrunken.

The shrink operation is possible only when the S and D fields of the vector point to a standard descriptor or a super-descriptor.

a. Standard Descriptor

In a standard descriptor the base adder and size fields of the vector are relative to the base and bound fields of the selected descriptor.

New Base = Old Base + Base Adder

↑ Carry causes STR
(DPS 88: BND) fault

New Bound = Size

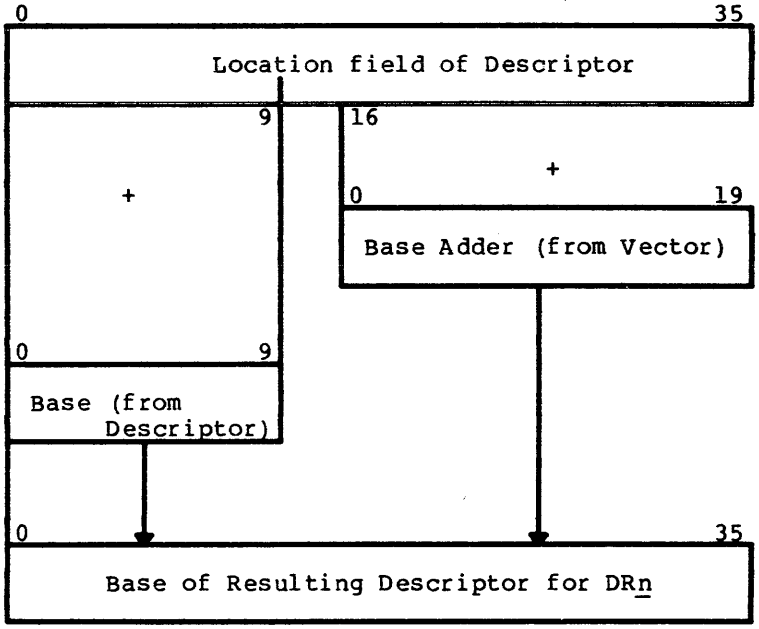
The flags field of the vector specifies permission to be granted or withheld and is combined with the flags field of the selected descriptor in such a way that no more permission is granted than is available (i.e., a bit-by-bit logical AND of the two flag fields). A fault will not occur if more permission is requested than is available. The result of the combination of the two flag fields is loaded into the DRn flag field. For a T = 2 or 3 descriptor, the flags field is only 3 bits; therefore, only these 3 bits are logically ANDed with the corresponding 3 bits from the vector.

The corresponding ARn is zero-filled and the SEGIDn is set to be self-identifying.

b. Super-Descriptor

The shrinking of a super-descriptor produces a standard descriptor. A super-descriptor of type T = 4 or 6 yields a standard descriptor of type T = 0 or 2, respectively.

The base value for the resulting standard descriptor is formed from the base and location fields of the selected descriptor and the base adder field of the vector as shown:


0
35
Location field of Descriptor
9
16
+
+
0
19
Base Adder (from Vector)
0
9
Base (from Descriptor)
0
35
Base of Resulting Descriptor for DRn

New Base = Base + (Location + Base Adder)

Carry causes STR (DPS 88: BND) fault

Carry causes STR (DPS 88: BND) fault

This new base and the 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 and ARn is zero-filled.

DATA STACK SHRINK
Vector Type = 11

Word 2 of the vector is ignored and the descriptor is generated from the data stack descriptor register (DSDR), the data stack address register (DSAR), and the size and flags fields of word 1 of the vector.

a. The first requirement is that:

   DSAR + Size $\leq$ Bound (DSDR)

   The three (DPS 88: five) least significant bits of the size field from the vector are forced to ones.

   ****DPS 8/70, 8/50, 8/52, 8/62: A size value that is one greater than it should be is used to check the DSDR bound****

   If the sum of DSAR plus size (rounded) exceeds the bound field or a carry is generated, an STR (DPS 88: BND) fault occurs and DRn, ARn, and SEGIDn remain unchanged.

b. The next requirement is a successful validation in which the new base is formed by adding DSAR to the base (DSDR). Generation of a carry causes an STR (DPS 88: BND) fault with no change in the contents of the registers.

c. ****DPS 8: Then, the data stack clear flag of the option register (OR) is checked. If this bit = 0, no clearing is required. If this bit = 1, the entire memory space to be framed by the generated descriptor is cleared.

   Any fault, including a Missing Page fault or STR fault generated during the clear memory operation, causes termination of this instruction with no change in the contents of the registers.****

d. Upon successful completion of the preceding operations, the new base (DSAR + Base (DSDR)) is loaded in the DRn base and the vector size field is loaded into the DRn bound field.

e. The new flags field for DRn is formed by combining the flags field of the vector and the flags field of the DSDR as described for the normal shrink of a standard descriptor.

f. The contents of the WSR and Type fields of DSDR are transferred to the WSR and T fields of DRn.

g. The corresponding ARn is set to zero and SEGIDn is set to be self-identifying as described for a normal shrink operation.

h. The DSAR is loaded with the value DSAR plus size

DPS 8: $C(DSAR)_{0-16} + SIZE_{0-16} + 1$ --> $C(DSAR)_{0-16}$*

***DPS 88: $C(DSAR)_{0-14} + SIZE_{0-14} + 1$ --> $C(DSAR)_{0-14}$

The DSAR is not allowed to "wraparound"; therefore, an STR (DPS 88: BND) fault is generated if the addition produces a carry.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, IR, RI, IT

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Illegal Procedure (IPR) Faults

Any of the following conditions causes an IPR fault:

a. Modifications RI, IR, IT, DU, and DL.

b. Illegal repeats RPT, RPD, and RPL.

c. Vector fields S = 0 and D = 1760 (octal).

d. If vector field V=01 and descriptor obtained is type T=5 or 7-15.

e. If instruction bit 29 = 1 and DRm contains a type T = 5 or 7-15 descriptor.

f. If vector field V=10.

2. Command Faults

   A Command fault is generated

   a. If vector fields S = 0 and D = 1761, 1763, or 1764 (octal) and the processor is not in Privileged Master mode

   b. If an access for a descriptor or vector (DPS 8: or for the memory clear)

      - Specifies working space zero and the processor is not in Privileged Master mode

      - Specifies working space register 0 and the processor is in the Slave mode

3. Memory Faults (STR; DPS 88: BND)

   Any of the following conditions cause an STR (DPS 88: BND) fault:

   a. Vector fields S = 0 and (D > bound field of parameter stack register and D < 1760).

   b. Vector fields S = 2 and D > bound field of argument stack register.

   c. Vector fields S = 1 or 3 and D > bound field of linkage segment register.

   d. Attempted shrink operation on standard descriptor with Base Adder + Size > bound field (DR_n_).

   e. Attempted shrink operation on super-descriptor with Location (DR_n_) + Base Adder + Size > bound field (DR_n_).

   f. An illegal carry or borrow while forming or size checking the base and bound fields or when generating the new DSAR value for a data stack shrink.

   g. An associative memory error.

   h. Descriptor flag bit 27=0 (not valid).

   i. Virtual address > 2^{24} words (DPS 88: 2^{26} words) and WS zero or dense paging is specified.

4. Missing Segment Faults

   A Missing Segment fault is generated if access is attempted to a segment for a vector, descriptor, or memory clear and flag bit 28 of the segment descriptor is 0.

5. Missing Page Faults

   A Missing Page fault is generated if access is attempted to a segment for a vector, descriptor, or memory clear and flag bit 30 of the page table word (PTW) for the accessed page is 0.

6. Missing Working Space Faults

   A Missing Working Space fault is generated if access is attempted to a segment for a vector, descriptor, or memory clear and flag bit 20 (DPS 88: flag bit 23) of the page table directory word (PTDW) is 0.

7. Security Fault, Class 1

   Any of the following conditions cause a Security Fault, Class 1:

   a. Attempted access to a segment for vectors when flag bit 32 of the PTW for the specified page is 1 (housekeeping) and the processor is in Slave mode.

   b. Attempted access to a segment for a descriptor when flag bit 32 of the PTW for the specified page is 0 (nonhousekeeping).

   c. ****DPS 8: An attempted data stack clear operation to a housekeeping page (flag bit 32 of the PTW is 1) and the processor is not in Privileged Master mode.****

8. Security Fault, Class 2

   Any of the following conditions cause a Security Fault, Class 2:

   a. Attempted access to a segment for a vector or descriptor when read flag bit 20 of the segment descriptor is 0.

   b. ****DPS 8: An attempted data stack clear operation when flag bit 21 of the data stack descriptor register (DSDR) is 0 or the accessed page does not have write permission (flag bit 31 of the PTW is 0).****

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
BUFVEC VEC     .ISR,BUFFER,320,READ
```

LDDSA	Load Data Stack Address Register	170 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: Bits 0-16 of C(Y) --> C(DSAR)
****DPS 88: Bits 0-14 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.

   ****DPS 88: The DSAR is a 15-bit register that holds a mod 8 word address.****

2. Modifications DU, DL, CI, SC, SCR and illegal repeats RPT, RPD, RPL cause an IPR fault.

3. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

   ****DPS 88: If the processor is not in the Privileged Master mode, the execution of this instruction causes an IPR fault.****

EXAMPLE: (from module ROL3)

```
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 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(Y-pair) --> C(DSDR)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. 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.

2. Any of the following conditions causes an IPR fault (the DSDR remains unchanged):

   a. Modification CI, SC, SCR, DU, or DL.

   b. Illegal repeat RPT, RPD, or RPL.

   c. If type field T is not equal to 0.

   d. ****DPS 8: If the base is not 0 modulo 8 bytes; or if bound is not 7 modulo 8 bytes; or if flag bit 22 is not 0.****

   e. ****DPS 88: If the base is not 0 modulo 32 bytes; or if bound is not 31 modulo 32 bytes; or if flag bit 22 is not 0.****

3. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

****DPS 88: If the processor is not in the Privileged Master mode, the execution of this instruction causes an IPR 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 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: 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 8 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. Any of the following conditions causes an IPR fault:

   a. Modification DU, DL, CI, SC, or SCR.

   b. If descriptor type field T of DRn is not 4 or 6.

   c. Illegal repeat RPT, RPD, or RPL.

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     =0777777,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 7-1)

PROCESSOR MODE: Any

SUMMARY: $C(Y)_{18-32}$ --> C(IR); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Master mode - Not affected

All others - If corresponding bit in C(Y) = 1, then ON; otherwise, OFF

NOTES:

1. 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	Undefined
32	Hexadecimal mode
33-35	Undefined

2. The Tally Runout indicator reflects bit 25 of C(Y) regardless of what address modification is performed on the LDI instruction for tally operations.

3. Master Mode cannot be changed by the LDI instruction.

4. 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.

5. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

LDO	Load Option Register	172 (1)

****DPS 8:

FORMAT: Single-word instruction format (see Figure 7-1).

PROCESSOR MODE: Any.

SUMMARY: Data Stack Clear Flag (DSCF) is loaded from $C(Y)_{18}$
0 = do not clear
1 = clear

Safe Store Bypass Flag (SSBF) is loaded from $C(Y)_{19}$
0 = bypass safestore during ICLIMB
1 = perform safestore during ICLIMB

Cache Read Control Flag (CRCF) is loaded from $C(Y)_{24}$
0 = bypass cache
1 = use cache

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, 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 three flag bits is mode dependent.

   In Privileged Master mode
   DSCF, SSBF, CRCF are loaded

   In Master mode
   DSCF and SSBF are not changed; CRCF is loaded

   In Slave mode
   DSCF, SSBF, CRCF are not changed

2. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, RPL cause an IPR fault.

3. If the SSBF is set to a 1, it is the responsibility of the operating system to preload the SSR.

4. ****DPS 8/20 and 8/44: This instruction is also valid for execution when the VS mode switch is in the OFF position; however, only the CRCF can be loaded.****

5. The DSCF controls the clearing of memory when a data stack shrink is performed with the LDDn instruction. DSCF = 1 means clear memory; DSCF = 0 means do not clear memory.

6. The SSBF controls bypassing the safestore part of an Inward CLIMB. SSBF = 0 means bypass safestore; SSBF = 1 means perform safestore.

EXAMPLE:

```
1      8         16              32
*      LOAD SAFE STORE REGISTER AND OPTION REGISTER; Privileged Master
                                                        mode only
       LDSS      CPOSS
       LDO       =O204000,DL     SSBF,CRCF ON
       TRA       MSFRM
SLVSS  LDSS      CPNOSS
       LDO       =O200000,DL     SSBF ON
       .
       .
```

****

LDO	Load Option Register	172 (1)

****DPS 88:

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY:

$C(Y)_0$ --> Hex Permission Flag
0 = inhibit hex; 1 = enable hex
$C(Y)_{1-2}$ --> Lockup Fault Time Limit
See note 1.
$C(Y)_3$ --> Safe-Store Bypass Flag (SSBF)
0 = perform; 1 = bypass
$C(Y)_4$ --> Data Stack Clear Flag (DSCF)
0 = don´t clear; 1 = clear
$C(Y)_{5-17}$ --> Option Register bits 5-17

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The Lockup fault time limit is:

Bits 1-2	Time Limit
00	2 ms
01	4 ms
10	8 ms
11	16 ms

The specified time limit is effective in Slave mode only. When in Privileged Master or Master mode the Lockup fault time limit is 32 milliseconds. Upon entry to, and while executing in Hyper mode, the Lockup fault timer is reset to zero. Thus the Lockup fault may not be detected until up to 64 milliseconds have elapsed.

2. Bits 5-17 should be filled with zero values to ensure compatibility with future systems. These fields are ignored and have no effect on operation.

3. The execution this instruction in other than Privileged Master mode causes an IPR fault.

4. Bits 18-35 of the Option Register can only be loaded by the following instructions, which are valid in Hyper mode only: LDHC (bits 18-32), LGCOS (bit 33), LVMS (bit 34), LMSD (bit 35).

5. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

****

LDPn	Load Pointer Register n	47n (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY:

If DU or DL modifications are not used
  $C(Y)_{0-23}$ --> C(ARn)
  $C(Y)_{24-35}$ --> C(SEGIDn)
  $C(Y)_{24-35}$ are interpreted as S,D field
If DU modification is used
  EA --> $C(ARn)_{0-17}$
  0 --> $C(ARn)_{18-23}$
  0 --> C(SEGIDn)
  0,0 is interpreted as S,D field
If DL modification is used
  0 --> $C(ARn)_{0-17}$
  EA_{0-5} --> $C(ARn)_{18-23}$
  EA_{6-17} --> C(SEGIDn)
  EA_{6-17} are interpreted as S,D field
In all cases
  S,D selects a descriptor as in the Copy Version of LDDn
  C(Selected Descriptor) --> C(DRn) or DRn type field is changed. SEGIDn is loaded as in the Copy Version of LDDn

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. 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.

   Bits 0-23 of the contents of memory location Y are loaded in ARn and bits 24-35 are interpreted as S and D fields. The interpretation of the S and D fields and the pertinent action are described in note 2 below.

2. The S and D fields of the pointer locate the descriptor to be loaded into DRn as follows:

When S = 0:

For D = 0000 through 1757 (octal) and D ≤ PSR bound, the descriptor is loaded from the parameter stack 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 Stack 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.

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 STR (DPS 88: BND) fault occurs, DRn, ARn, and SEGIDn are not changed.

3. An IPR fault occurs if bit 29=1 and the operand segment is not type T = 0, 2, 4, or 6.

4. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

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
```

LDPS	Load Parameter Stack Register	771 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(Y-pair) --> C(PSR); C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. 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.

2. 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.

3. This instruction is identical with LDAS, except that it loads the parameter stack register (PSR) instead of the argument stack register (ASR).

4. Illegal Procedure faults and Command faults are the same as for LDAS.

EXAMPLE: (BRT1 vicarious fault handler)

```
1      8         16                  32
_____________________________________________________________
       LDP       P.SSR,.SSR,DL       (Load descriptor of fault
                                      frame in safestore stack)
       LDP       P.SSR,.CTYP,DL      (Change to type 0)
       LDAS      .WASR,,P.SSR        (Restore ASR from safestore)
       LDPS      .WPSR,,P.SSR        (Restore PSR from safestore)
```

LDQ	Load Q-Register	236 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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

LDQC	Load Q-Register and Clear	032 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Y) --> C(Q); 0...0 --> C(Y)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero	- If C(Q) = 0, then ON; otherwise, OFF
Negative	- If bit 0 of C(Q) = 1, then ON; otherwise, OFF

NOTES:

1. The LDQC instruction should only be used for gating purposes. It should not be used as a substitution for an LDQ, STZ pair because of the performance penalty that is introduced.

2. ****DPS 88: LDAC, LDQC, SZNC, STAC, and STACQ are the only instructions that can be used for the indivisible test-and-set operations which are required for setting and releasing locks, or for closing and opening gates.

   Since execution of LDAC, LDQC, SZNC, STAC, and STACQ depends on the previous C(Y), the processor will obtain ownership of the 8-word block containing C(Y) prior to using C(Y) to execute the instruction. Obtaining ownership of the 8-word block means that the requesting processor, and the Memory Hierarchy Control of the CIU, will ensure that a valid copy of the block is obtained, and that the block is cleared from the cache of all other processors before the instruction is executed. After obtaining ownership of the block, the processor completes execution of the instruction to set or release the lock without permitting the block to be siphoned to another processor. Thus the block is isolated in a time window where it can be accessed and modified only by the processor executing the instruction which sets or releases the lock.

To ensure that a lock does not get released before the actual completion of all stores performed while the lock was set, a synchronizing function is necessary. This synchronizing function is accomplished by coding a SYNC or STC2 instruction immediately before the instruction which releases the lock. If the value stored by STC2 is consistent with operating system conventions for a released lock, then the use of STC2 for synchronizing can also serve to release the lock.****

3. An Illegal Procedure fault occurs if illegal address modification is used.

LDSS	Load Safe Store Register	773 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(Y-pair) --> C(SSR); C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. 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. flag bits 20, 21, 27, and 28 = 1 and flag bits 25 and 26 = 0 for T = 1.

   b. flag bits 20 and 21 = 1 for T = 3.

   c. ****DPS 8: Base must be 0 modulo 8 bytes****

   d. ****DPS 88: Base must be 0 modulo 32 bytes****

   If these conditions are met, the descriptor is loaded into the safe store register (SSR); otherwise, an IPR fault is generated and the SSR remains unchanged.

2. Each successful execution of LDSS causes the 2-bit stack control register (SCR) to be initialized to binary 11 indicating a previous frame size of 64 words. (The SCR is associated with the SSR and contains a code that denotes the size of the last frame on the stack.)

3. Any of the following conditions causes an IPR fault:

   a. Modification DU, DL, CI, SC, or SCR.

   b. Illegal repeat RPT, RPD, or RPL.

   c. If T is not equal to 1 nor 3.

   d. If either the flag bit or the base checks fail.

4. **** DPS 8: If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.****

   ****DPS 88: If the processor is not in the Privileged Master mode, the execution of this instruction causes an IPR 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    =O377001,DU
       STX0    .WREGS,,P0
       TRA     RETOUT
```

LDT	Load Timer Register	637 (0)

FORMAT:	Single-word instruction format (see Figure 7-1)
PROCESSOR MODE:	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. ****DPS 8: The use of this instruction in the Slave mode causes a Command fault.****
2. ****DPS 88: 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.

LDWS	Load Working Space Registers	772 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY:

If EA_{17} = 0, then
$C(Y)_{0-8,\ 9-17,\ 18-26,\ 27-35}$ --> C(WSR)0,1,2,3

If EA_{17} = 1, then
$C(Y)_{0-8,\ 9-17,\ 18-26,\ 27-35}$ --> C(WSR)4,5,6,7

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. 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 of the effective address.

   ****DPS 88: Execution of this instruction clears the associated "hidden" registers holding the WSPTD words for the most recently accessed working spaces.****

2. Modifications CI, SC, SCR, DU, DL and illegal repeats RPT, RPD, RPL cause an IPR fault.

3. ****DPS 8: If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.****

   ****DPS 88**** If the processor is not in the Privileged Master mode, the execution of this instruction causes an IPR fault.****

4. 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   VRD     9/001, 9/001, 9/013, 9/27
WS4    VFD     9/45, 9/45, 9/63, 9/510
       .
       .
       .
       DLWS    WS03                         Load WSR 0-3 from EVEN word
       LDWS    W547                         Load WSR 4-7 from Odd word
```

LDXn	Load Index Register n from Upper	22n (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0,1,...7 as determined by op code
$C(Y)_{0-17}$ --> C(Xn); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of LDX0 cause IPR fault.

INDICATORS:

Zero	- If C(Xn) = 0, then ON; otherwise, OFF
Negative	- If $C(Xn)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged as illegal but executes with all zeros for data.

2. An Illegal Procedure fault occurs if illegal address modification is used.

****DPS 88 ONLY****

LIMR	Load Interrupt Mask Register	553 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: $C(A)_{0-7}$ --> $C(\text{Interrupt Mask Register})_{0-7}$

If $C(A)_8$ = 1 the "ALL" register is set to OFF.

$C(A)_{9-35}$ must be zero.

When i = 0 to 7, a 0 in bit i of the Interrupt Mask Register will prevent acknowledgement of interrupt level queue i; a 1 in bit i of the interrupt Mask Register will permit acknowledgement of interrupt level queue i, and if there is a pending interrupt in queue i, the CIU will send an interrupt present signal.

C(A), C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The use of this instruction in other than Privileged Master mode causes an IPR fault.

2. Bits 9-35 are to be filled with zero values in order to ensure compatibility with future systems. Nonzero values in these fields are ignored and will have no effect on the operation.

3. The ALL register (one bit) can only be set ON by the CIU hardware, and causes all interrupt levels to be masked. If $C(A)_8$ = 0, then the ALL register remains unchanged. If $C(A)_8$ = 1, then the ALL register is set OFF. ICR level bits can be set and queue entries can be made, independent of the state of the ALL register.

4. This instruction loads the Interrupt Mask Register in the port assigned to this CPU in the Control CIU. CPU port selection (which CIU) is determined by bit 23 (Control CIU) of the Option Register. The Control CIU bit can be changed by the SSF, or by the LDHC instruction in Hyper mode, if reconfiguration requires the use of an alternate Port-CIU-Interrupt Mask Register.

5. In DPS 8/70, 8/50, 8/52, 8/62 processors the mnemonic SMCM (Set Memory Controller Mask Register) was assigned to operation code 553(0). The mnemonic has been changed to reflect the new functionality. $C(A)_{0-7}$ rather than $C(A)_{0-15}$ and $C(Q)_{0-15}$, are used to set the Interrupt Mask Register.

6. Interrupt queue entries are processed by the RIW instruction.

7. The effective address (Y) is not used by the LIMR instruction.

8. When LIMR is used to mask interrupts, program an RIMR instruction immediately following the LIMR instruction to ensure that the masking has been accomplished before executing other instructions which depend on the interrupts being masked.

9. 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 7-1)

PROCESSOR MODE: Any

SUMMARY: Rotate C(AQ) left by the number of positions indicated by bits 11-17 of Y (Y modulo 128); enter each bit leaving bit position 0 of AQ into 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 in the instruction must be a decimal number. To ´right-rotate´ <u>n</u> bits, use LLR 72-<u>n</u>.
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 7-1)

PROCESSOR MODE: Any

SUMMARY: Shift C(AQ) left by the number of positions indicated by bits 11-17 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
Carry	- If bit 0 of C(AQ) changes during the shift, then ON; otherwise OFF. When the Carry indicator is ON, the algebraic range of AQ has been exceeded

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 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY:

**** DPS 8: $C(Y)_{0-14}$ --> C(PDBR) ****

****DPS 88: $C(Y)_{0-16}$ --> C(PDBR) ****

C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The contents of bits 0-14 (DPS 88: 0-16) of Y replace the contents of the page directory base register (PDBR) and the Page Table Word Associative Memory (PTWAM) (DPS 88: Paging Buffer), if enabled, is cleared.

2. Modifications CI, SC, SCR, DU, DL and illegal repeats RPT, RPD, RPL cause an IPR fault.

3. ***DPS 8: If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.****

   ****DPS 88: If the processor is not in the Privileged Master mode, the execution of this instruction causes an IPR fault.****

EXAMPLE:

```
1      8      16              32
_____________________________________________________________________
* LOAD PAGE TABLE DIRECTORY BASE REGISTER
       LPDBR  PDBAS           base is 512
       RSW    2
       CANA   .FBT5,DU
       TNZ    *+2
       ANA    =O777777777773
       STA    YOKO
       ANA    7,DL
       TNZ    SLVSS
```

LPL	Load Pointers and Lengths	467 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1      8        16
_____________________________________
       LPL      LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: **** DPS 8: Control information required to recover from a mid-instruction interrupt of a multiword instruction is loaded from C(Y,Y+1,...,Y+7) C(pointer and length registers)

Bits 15-17 of Y = 000 for the first location. The actual contents of these bit positions are ignored and are assumed to be zero.****

****DPS 88: Control information required to recover from a mid instruction interrupt of a multiword instruction is loaded from C(Y,Y+1). The hardware assumes Y17 = 0 for the first location and increments addressing accordingly. No check is made.****

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. The LPL instruction provides the capability for loading the pointers for sending and receiving addresses, for sending and receiving field lengths, and for other required control information when an interruptible multiword instruction is interrupted during execution. See "Pointer And Length Registers" in Section IV.

3. ****DPS 8: The address register bit of the modification field for the operand descriptor is stored in bit 29. Bits 33-35 are the address registers designated by bits 0-2 of the Y field of the descriptor. Word 3 of the operand stores this data for operand descriptor 1, word 5 of the operand stores this data for operand descriptor 2, and word 7 of the operand stores this data for operand descriptor 3.****

4. The pointer and length registers enable the hardware to resume processing an interrupted instruction after a return from servicing the interrupt.

5. ****DPS 8: 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.****

   ****DPS 88: Location Y must be forced to a multiple of 2 by entering an 1 in column 7 of the statement that defines Y, or by using the EVEN pseudo-operation.****

6. This instruction is normally only used by routines that process interrupts.

LREG	Load Registers	073 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Y,...,Y+6) --> C(X0,...,X7,A,Q,E) where bits 15-17 of Y = 000; C(Y,...,Y+6) unchanged

Registers are loaded as follows:

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)

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 means of an 8 entered 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.

LRL	Long Right Logical Shift	773 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Shift C(AQ) right by the number of positions indicated by bits 11-17 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.

LRS	Long Right Shift	733 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Shift C(AQ) right by the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with bit 0 of C(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 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0,1....or 7 as determined by op code
$C(Y)_{18-35}$ --> C(Xn); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of LXL0 cause IPR fault.

INDICATORS: Zero - If C(Xn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. DU modification is flagged as illegal but executes with all zeros for data.

2. An Illegal Procedure fault occurs if illegal address modification is used.

MLR	Move Alphanumeric Left to Right	100 (1)

FORMAT:

<table>
<tr><td>00 – 08</td><td>09</td><td>10</td><td>11 – 17</td><td>18 – Op Code – 27</td><td>28</td><td>29 – 35</td></tr>
<tr><td>FILL</td><td>T</td><td>0</td><td>MF2</td><td>100(1)</td><td>I</td><td>MF1</td></tr>
</table>

<table>
<tr><td>00 – 02</td><td>03 – 17</td><td>18 – 20</td><td>21 – 22</td><td>23</td><td>24 – 32</td><td>33 – 35</td></tr>
<tr><td colspan="2">Y1</td><td rowspan="2">CN1</td><td rowspan="2">TA1</td><td rowspan="2">0</td><td colspan="2">N1</td></tr>
<tr><td>a1</td><td>Y1</td><td></td><td>R</td></tr>
</table>

<table>
<tr><td>00 – 02</td><td>03 – 17</td><td>18 – 20</td><td>21 – 22</td><td>23</td><td>24 – 32</td><td>33 – 35</td></tr>
<tr><td colspan="2">Y2</td><td rowspan="2">CN2</td><td rowspan="2">TA2</td><td rowspan="2">0</td><td colspan="2">N2</td></tr>
<tr><td>a2</td><td>Y2</td><td></td><td>R2</td></tr>
</table>

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
```

PROCESSOR MODE: Any

SUMMARY: C(string 1) --> C(string 2)

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 are dissimilar, each character will have 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. The contents of string 1 remain unchanged except in cases of string overlap.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 and MF2

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Truncation - If L1 is greater than L2, then ON; otherwise, OFF

NOTES:

1. 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.

2. L2 = 0 does not necessarily mean that the instruction functions as a no-op, as the Truncation indicator may be affected.

3. 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 1) store the K characters into character positions 1 through K of the string and 2) "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 K is less than eight double-words of characters when K 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.

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              xxxxABCDEFGHIJKLb̸b̸  (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,b̸b̸b̸12345678R
FLD2   BSS    2              xxxxxx123456789-  (Result)
       USE
```

NOTE:

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 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,)

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 defined as 0, 1, 2, 3, 4, 5, 6, 7, ALL, QU, A, or Q.

EXAMPLE:

```
1      8      16
              MLR    (1,0,0,ALL)(,,,QU)
              ADSC9  0,0,24,P.IOQ
              ADSC9  ,,24
```

This example would move 24 words from P.IOQ to QU.

See "Multiword Modification Field" and "Alphanumeric Operand Descriptors" in Section V, and "Alphanumeric Instructions" under "Multiword Operations" in Section VI for additional information.

MME	Master Mode Entry	001 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Generates a MME fault which causes the processor to switch to Privileged Master mode and to execute an Inward CLIMB instruction using the entry descriptor obtained from the word pair in
****DPS 8: real memory location 32 octal.****
****DPS 88: operating system memory location 32 octal.****

ILLEGAL ADDRESS MODIFICATIONS: None
****DPS 8/70, 8/50, 8/52, 8/62: 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 as the MME fault has higher priority.****

ILLEGAL REPEATS: RPT, RPD, RPL cause an Illegal Procedure fault.

INDICATORS: Master Mode - ON.

NOTES:

1. 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 MME fault is indicated in the safestore frame by a code of 00010 in bits 12-16 of word 5.

2. 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} = 0 Do not load X0.
   C_{19} has no effect. Turn Master Mode indicator ON.
   C_{22-23} = 00 Inward CLIMB.
   S,D has no effect.

3. 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 (IC).

4. 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.

****DPS 88 ONLY****

MMF	Move to Memory Format	364 (1)

FORMAT:

00	01	02–10	11–17	18–27	28	29–35
EC	B	0	MF2	364 (1)	I	MF1

00–17	18–20	21–29	30–35
Y1	CN1	0	N1

00–17	18–20	21–29	30–35
Y2	CN2	0	N2

PROCESSOR MODE: Any

SUMMARY: This instruction performs the inverse of MRF. Starting at location YC1+(L1-1), 1, 2, 3, or 4 characters are moved right-to-left to locations starting at YC2+(L2-1). Maximum allowable length for L1 and L2 is 4. Only the rightmost 6 bits (30-35) of descriptors 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 EC (bit 0) and B (bit 1) bits of word 1 have the following effect on the move:

o If B = 0, characters are moved unchanged from string 1 to string 2, 9 bits at a time.

o If B = 1, 8-bit characters (a byte) are picked up from string 1, and a zero is concatenated in the most significant bit position to form a 9-bit character. This 9-bit character is placed in string 2. String 1 contains a binary integer which is converted to memory character format.

- EC = 1 enables the following check, providing a hardware method to detect that overflow occurred during some previous operation on the binary data contained in string 1. After the last 9-bit character or 8-bit byte to be moved has been picked up from string 1, bit zero of this character or byte is compared with all of the bits remaining in string 1 which were not moved. A successful compare indicates that no data has been lost, but an unsuccessful compare means data has been lost and the Overflow indicator will be set.

  Note that this check is not conclusive. If numerous arithmetic operations are performed on 16/32 bit binary data, it is possible to overflow, generate a carry out of the working register, and produce a case where the extended sign bits again agree with the designated sign bit. Thus, the programmer should make intermediate checks for overflow and/or carry as appropriate to ensure that data is not lost.

- If EC = 0, the above check does not take place and the overflow indicator is not affected.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2

ILLEGAL REPEATS: RPT, RPD, RPL

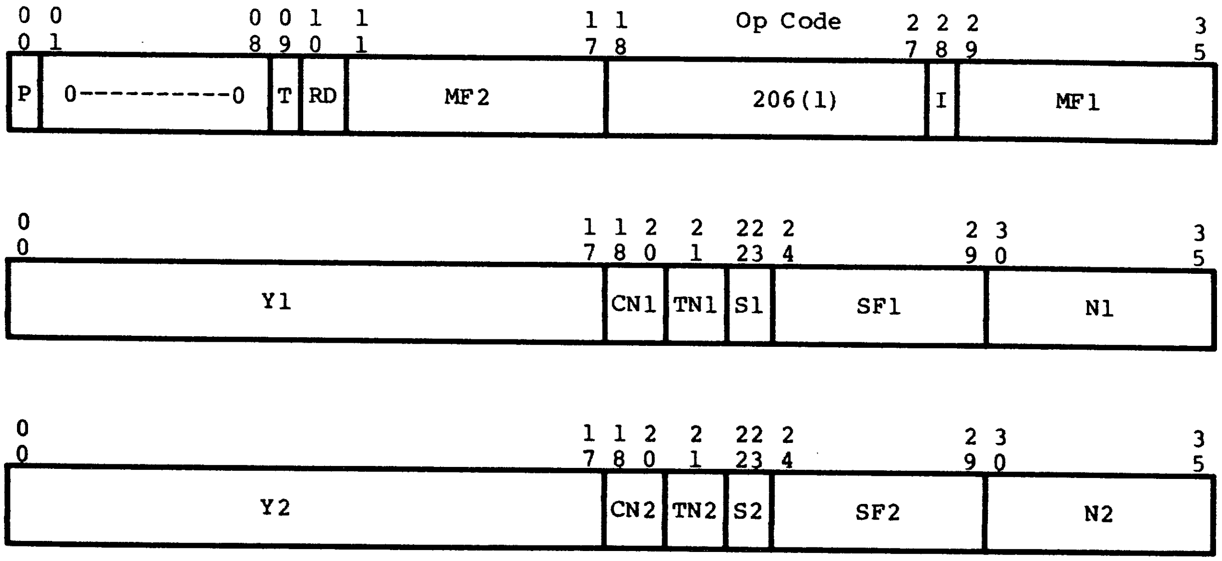
INDICATORS: Overflow indicator may be turned ON. See Summary.

NOTES:

1. If L2 > L1 or L1 > 4, an IPR fault will occur.
2. On an unsuccessful extended sign compare (EC = 1), the data is always moved. Then the Overflow indicator is turned ON. If the Overflow Mask indicator is OFF, the processor executes the overflow fault.
3. When bytes (binary data) are being moved, there will always be bits in string 1 which are not moved. However, when 9-bit characters are being moved, all of string 1 contents may be moved. In such a case, EC = 1 has no effect on the operation.
4. If L2 = 0 or L1 = L2 = 0, the Overflow Indicator is not affected and the instruction functions as a no-op.
5. If EC = 1 and the check is successful, the Overflow Indicator is not affected, i.e., it is not reset.
6. The primary purpose of this instruction is for the case where string 1 is effectively word or upper half word aligned, while string 2 is byte aligned. However, no hardware check is made to force string 1 to be word or upper half word aligned.
7. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are 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
```

PROCESSOR MODE: Any

SUMMARY: C(string 2) * C(string 1) --> C(string 2)

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 greater than 127, then ON; otherwise unchanged

Exponent Underflow - If exponent of floating-point result is less than -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 or MF2, 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.

   ****DPS 88: If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.****

   ****DPS 8: If an illegal digit or sign is detected, the receive field is not 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
```

****DPS 88****

MP2DX	Multiply Using Two Decimal Operands Extended	246 (1)

FORMAT:

00	01	02 08	09	10	11 17	18 Op Code 27	28	29 35
EA	NS	0	T	RD	MF2	246 (1)	I	MF1

00 17	18 20	21	22 23	24 29	
Y1	CN1	TN1	SX1	SF1	N1

00 17	18 20	21	22 23	24 29	
Y2	CN2	TN2	SX2	SF2	N2

PROCESSOR MODE: Any

SUMMARY: C(string 2) * C(string 1) --> C(string 2)

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 AD3D.

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:

00	01	02 08	09	10	11 17	18 Op Code 27	28	29 35
P	0	MF3	T	RD	MF2	226(1)	I	MF1

00 17	18 20	21	22 23	24 29	30 35
Y1	CN1	TN1	S1	SF1	N1

00 17	18 20	21	22 23	24 29	30 35
Y2	CN2	TN2	S2	SF2	N2

00 17	18 20	21	22 23	24 29	30 35
Y3	CN3	TN3	S3	SF3	N3

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
```

PROCESSOR MODE: Any

SUMMARY:

C(string 2) * C(string 1) --> C(string 3)

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 scaled 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. 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. 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 is specified), OFF

Exponent Overflow - If exponent of floating-point result is greater than 127, then ON; otherwise, unchanged

Exponent Underflow - If exponent of floating-point result is less than -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. ****DPS 88: If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.****

****DPS 8: If an illegal digit or sign is detected, the receive field is not 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
```

****DPS 88 ONLY****

MP3DX	Multiply Using Three Decimal Operands Extended	266 (1)

FORMAT:

00	01	02–08	09	10	11–17	18–27 (Op Code)	28	29–35
EA	NS	MF3	T	RD	MF2	266 (1)	I	MF1

00–17	18–20	21	22–23	24–29	30–35
Y1	CN1	TN1	SX1	SF1	N1

00–17	18–20	21	22–23	24–29	30–35
Y2	CN2	TN2	SX2	SF2	N2

00–17	18–20	21	22–23	24–29	30–35
Y3	CN3	TN3	SX3	SF3	N3

PROCESSOR MODE: Any

SUMMARY: C(string 2) * C(string 1) --> C(string 3)

The decimal number of data type TN2, sign and decimal type SX2, and starting location YC2, is multiplied by the decimal number of data type TN1, sign and decimal type SX1, 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 SX3. If SX3 indicates a scaled 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. The character set is defined by EA. Placement of overpunched sign in the output is controlled by NS. If RD is a 1, rounding takes place 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

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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(A) * C(Y) --> C(AQ), left adjusted; C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

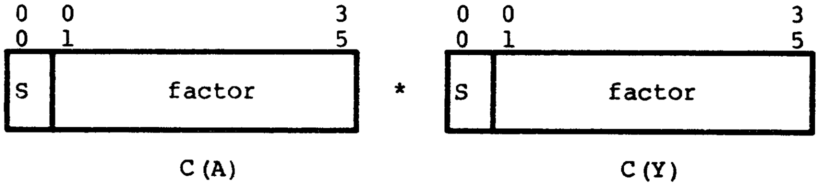
Negative - If bit 0 of C(AQ) = 1, then ON; otherwise, OFF

Overflow - If range of AQ is exceeded, then ON

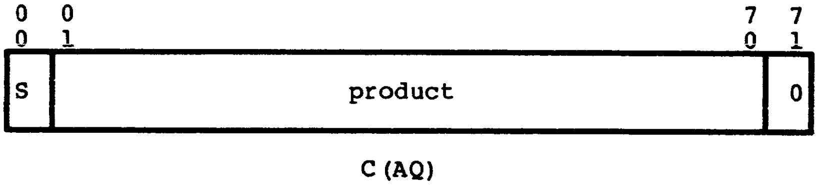
NOTES:

1. 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.

2. Overflow can occur only when A and Y both = -1 and the result exceeds the range of the AQ-register.



yielding:



3. An Illegal Procedure fault occurs if illegal address modification is used.

MPY	Multiply Integer	402 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Q) * C(Y) --> C(AQ), right adjusted; C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: None

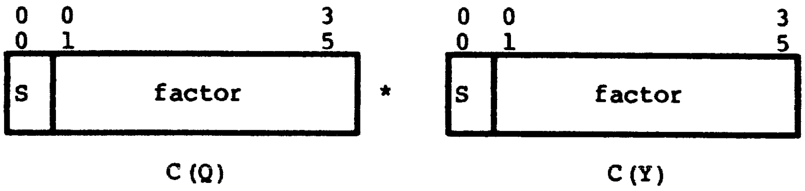
INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

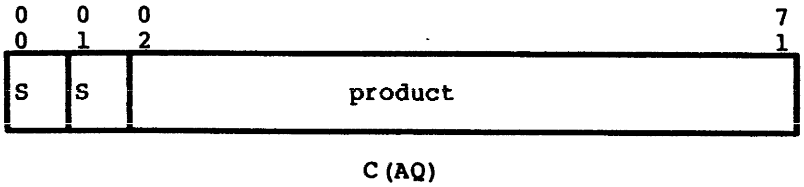
Negative - If bit 0 of C(AQ) = 1, then ON; otherwise, OFF

NOTES:

1. 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:



In the case of (-2**35) * (-2**35) = +2**70, bit 1 of AQ is used to represent the product rather than the sign. No overflow can occur.

2. An Illegal Procedure fault occurs if illegal address modification is used.

****DPS 88 ONLY****

MRF	Move to Register Format	360 (1)

FORMAT:

00	01	12 10	11 17	18 27	28	29 35
SE	B	0	MF2	360 (1)	I	MF1

00 17	18 20	21 29	30 35
Y1	CN1	0	N1

00 17	18 20	21 29	30 35
Y2	CN2	0	N2

PROCESSOR MODE: Any

SUMMARY: Starting at location YC1+(L1-1), 1, 2, 3, or 4, 9-bit characters are moved, right to left, to starting location YC2+(L2-1). Maximum allowable length for L1 and L2 is 4. Only the rightmost 6 bits (30-35) of descriptors are interpreted for length. Likewise, when a register is specified as containing the length, only the rightmost 6 bits of the register are interpreted. Bits 0 (SE) and 1 (B) of the first word enable the following actions to occur during the move.

<u>B =0. SE = 0.1</u>

When B = 0, the 9-bit characters from string 1 are moved unchanged to string 2. When L2 > L1, the remaining character positions of L2 are filled as follows:

If SE = 0, zeros fill the remaining character positions.

If SE = 1, bit 0 of the last character moved is treated as the sign and is extended to fill the remaining character positions.

B = 1. SE = 0.1

When B = 1, bit 0 is removed from each 9-bit character of string 1 and the resulting 8-bit bytes are placed right justified in string 2, i.e., the 9-bit characters from string 1 represented binary data but the extra bit (bit 0 of the 9-bit characters) separated each 8-bit byte of binary data. The SE bit will affect the results of the move as follows:

If SE = 0, zeros fill the remaining bit positions of string 2.

If SE = 1, bit 0 of the last 8-bit byte moved to string 2 (this is bit 1 of the original 9-bit character from string 1) is extended to fill the remaining bit positions of string 2.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2

ILLEGAL REPEATS: RPT, RPD, RPL

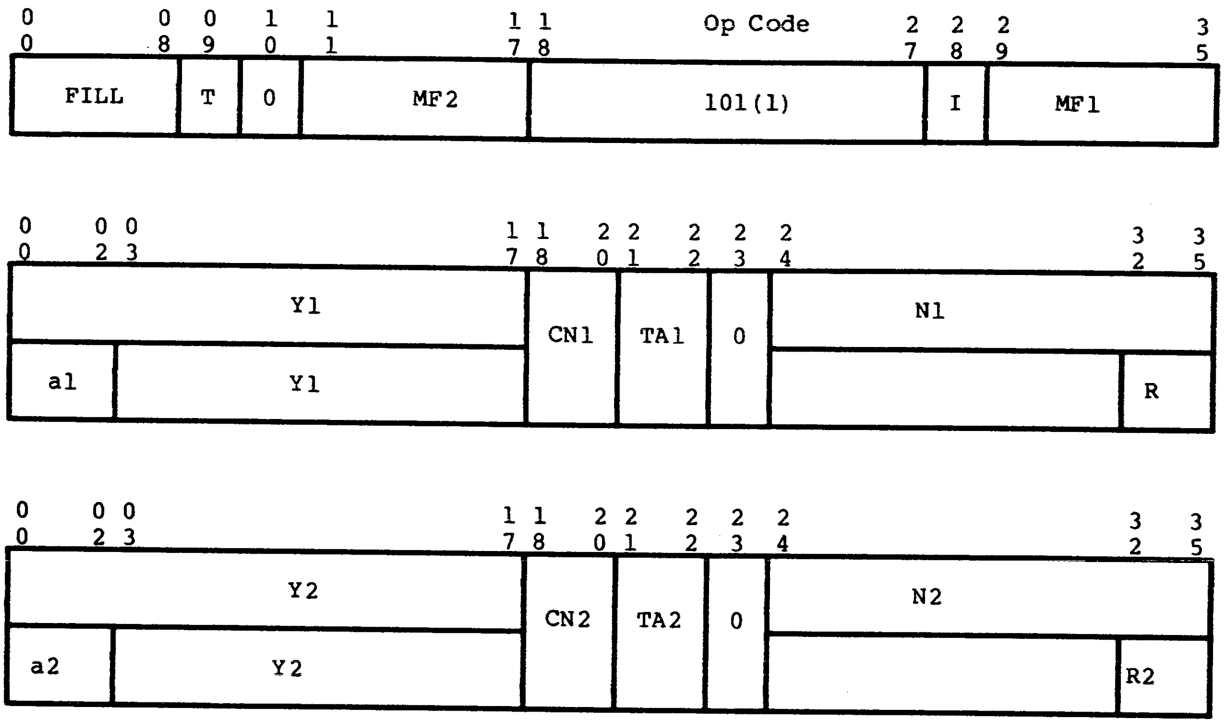
INDICATORS: None affected

NOTES:

1. If L1 > L2 or if L2 > 4, an IPR fault will occur.
2. The primary purpose of this instruction is to move a byte-aligned sending field to a word or upper half-word-aligned receive field, operating on the characters as described during the move. Note that no hardware check is made to force the word or upper half-word alignment of the receive field.
3. If L1 = 0 or L1 = L2 = 0, the instruction functions as a no-op.
4. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are 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
```

PROCESSOR MODE: Any

SUMMARY: C(string 1) --> C(string 2)

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 greater than L2, then ON; otherwise, OFF

NOTES:

1. 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.

2. Refer to Note 3 of the MLR instruction for information on string replication.

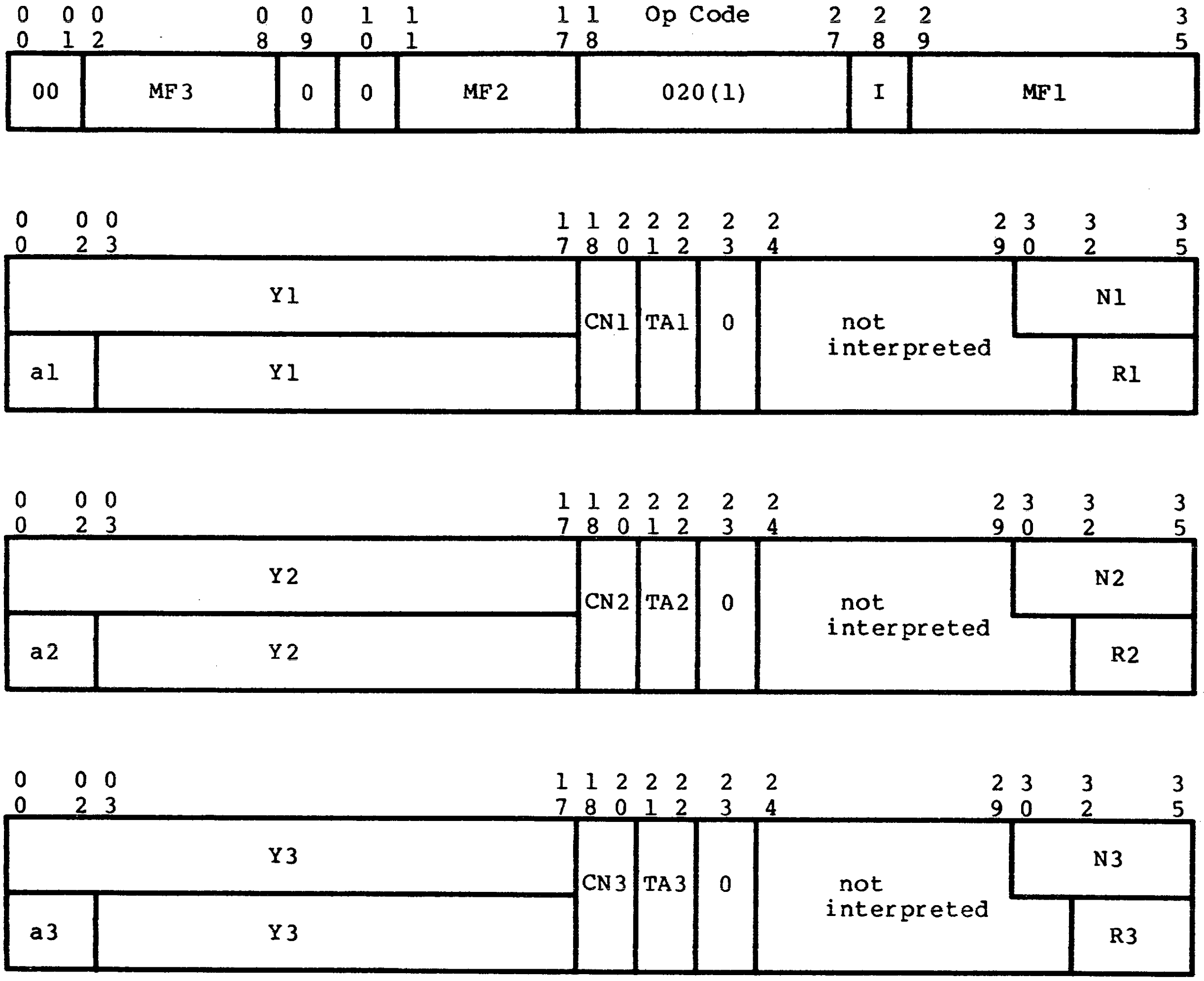
3. L2 = 0 does not necessarily mean that the instruction functions as a no-op because the truncation indicator may be affected.

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                     xxxxƀƀABCDEFGHIJKL   (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,ƀƀƀ12345678R
FLD2      BSS       2                     xxxx-00123456789   (Result)
          USE
```

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
```

PROCESSOR MODE: Any

SUMMARY:

$$C(\text{string 1}) \xrightarrow{\text{string 2 control}} C(\text{string 3})$$

Starting at location YC1, the string of alphanumeric characters of data type TA1 is moved under control of the micro-operation sequence of length L2 and type TA2 = 00 that starts at location YC2 to the string of alphanumeric characters of data type TA3 starting at location YC3. 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 hardware is not responsible for results, nor can it guarantee identical results on future machines, if any overlap is defined for the three strings.) The contents of the alphanumeric character string that starts at YC1 and the micro-operation sequence that starts at YC2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1, MF2, or MF3; if a move from an exhausted sending string or call to an exhausted micro operation sequence is attempted, if L2 = 0, or if illegal repeats are used.

   ****DPS 88: If an IPR fault occurs for one of the conditions described in the preceding sentence, part or all of the receive field may be changed before the IPR fault occurs.****

2. TA2 is assumed to be 00 and is not interpreted by the hardware.

3. Refer to "Micro-Operations" in this section for additional information.

4. 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.

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*,.-ƀƀƀƀ,(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 is an explanation of the above micro-operation sequence:
*      (CHT,0),8H*,.-ƀƀƀƀ - 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 (ƀ)
*      (MVC,10)  - Move 10 characters from FLD1 (SMITHROGER)
*      (INSB,2)  - Insert Edit Table Entry #2 (,)
*      (INSB,5)  - Insert Edit Table Entry #5 (ƀ)
*      (MVC,7)   - Move 7 characters from FLD1 (WILLIAM)
*      (INSB,5)  - Insert Edit Table Entry #5 (ƀ)
*      (MVC,1)   - Move 1 character from FLD1 (S)
*      (INSB,3)  - Insert Edit Table Entry #3 (.)
*      (INSB,5)  - Insert Edit Table Entry #5 (ƀ)
*      (INSB,4)  - Insert Edit Table Entry #4 (-)
*      (INSB,5)  - Insert Edit Table Entry #5 (ƀ)
*      (INSB,0),1H# - Insert specified character (#)
*      (MVC,2)   - Move 2 characters from FLD1 (25)
*                                memory contents in BCD characters
FLD   BSS    5                 *ƀSMITHROGER,ƀWILLIAMƀS.ƀ-ƀ#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            codeƀerror#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:

00	01 08	09	10	11 17	18 Op Code 27	28	29 35
P	0-----0	T	RD	MF2	300(1)	I	MF1

00 02	03 17	18 20	21 22	23	24 29	30 35
Y1		CN1	TN1	S1	SF1	N1
a1	Y1					

00 02	03 17	18 20	21 22	23	24 29	30 35
Y2		CN2	TN2	S2	SF2	N2
a2	Y2					

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
```

PROCESSOR MODE: Any

SUMMARY: C(string 1) --> C(string 2)

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 scaled 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 floating and scaled formats. If P = 0, positive signed 4-bit results are stored with octal 14 as the plus sign. 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 greater than 127, then ON; otherwise, unchanged

Exponent Underflow - If exponent of floating-point result is less than -128, then ON; otherwise, unchanged

Overflow - If fixed point integer overflow, then ON; otherwise, unchanged.

****DPS 8: In addition, if internal register 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:

   a. DU or DL modification is specified for MF1 or MF2, or if illegal repeat is 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. Refer to Note 3 of the MLR instruction for information on string replication.

4. ****DPS 88: If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.****

   ****DPS 8: If an illegal digit or sign is detected, the receive field is not changed before the IPR fault occurs.****

EXAMPLES:

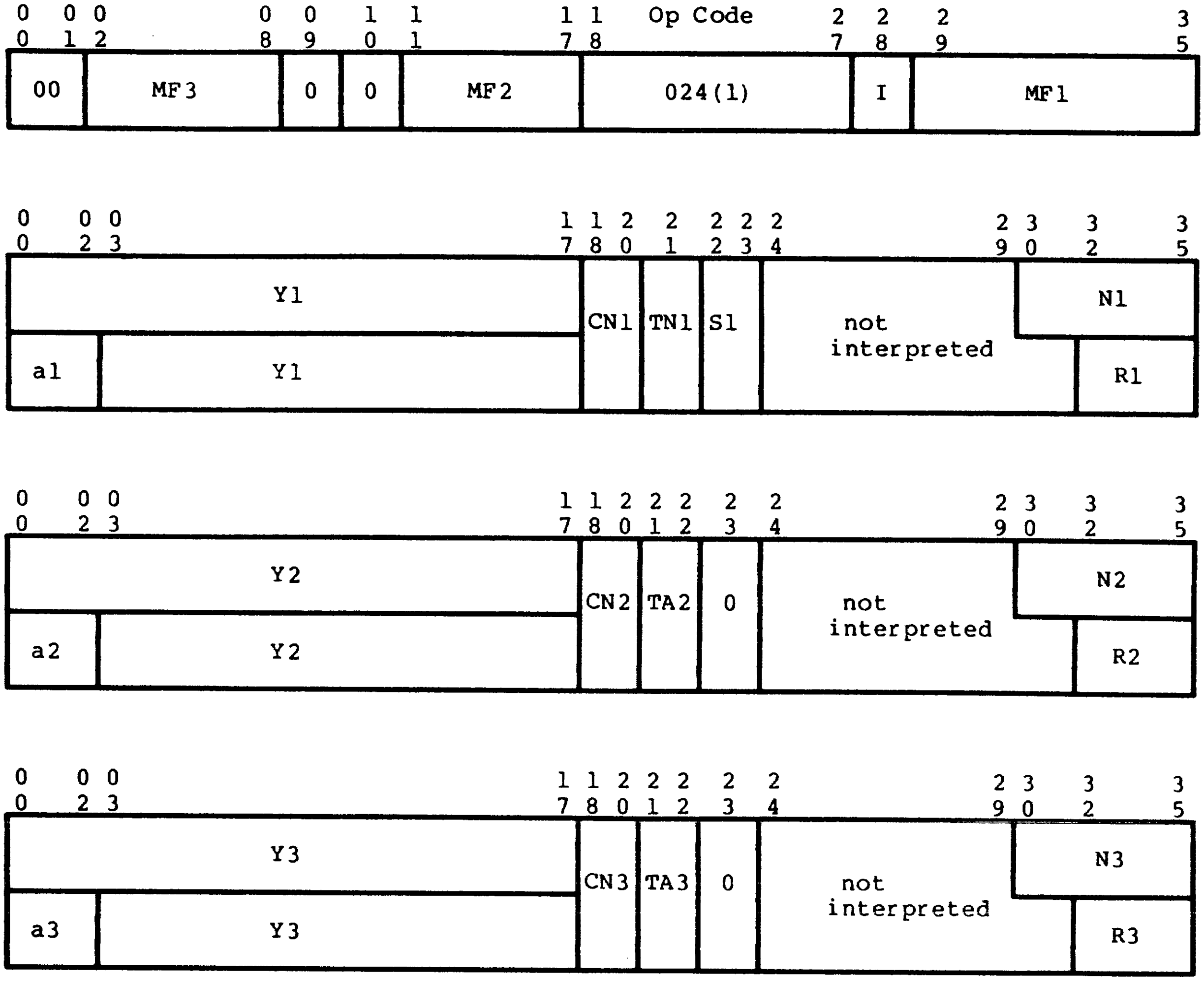
```
1      8       16              32
_____________________________________________________________
       MVN     ,,1             with rounding option
       NDSC4   FLD1,0,8,2,-3   sending field operand descriptor
       NDSC4   FLD2,1,7,1,-2   receiving field operand descriptor
       USE     CONST.          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   receiving 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
```

PROCESSOR MODE: Any

SUMMARY:

```
              string 2 control
C(string 1) ---------------> (string 3)
```

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. 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 hardware is not responsible for results, nor can it guarantee identical results on future machines, if any overlap is defined for the three strings. The operation stops when L3 is exhausted.

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.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Illegal Procedure fault same as for MVN. In addition, an Illegal Procedure fault occurs if L2 equals zero (DPS 8: IPR if L3 = 0), or if the micro-operation sequence terminates abnormally.

   ****DPS 88: Normal termination occurs when L3 is decremented to zero; thus, if L3 = 0 at the start of the instruction, and there are no other faults, no operation is performed and the instruction terminates normally. No attempt is made by hardware to access the L1 or L2 field when L3 = 0 at the start of the instruction.****

   ****DPS 88: If an IPR fault occurs because of an illegal numeric digit (not illegal sign), illegal micro operation, or insufficient length of L1 or L2, part or all of the receive field may be changed before the IPR fault occurs.****

2. Refer to "Micro-Operations" in this section for additional information.

EXAMPLES:

```
1      8       16              32
________________________________________________________________________
       MVNE                    with ($) float and (.) inserted
       NDSC9   FLD1,0,10,2     sending field operand descriptor
       ADSC9   FLD2,0,14       micro-op string operand descriptor
       ADSC6   FLD3,0,12       receiving field operand descriptor
       USE     CONST.          memory contents in ASCII characters
FLD1   EDEC    10A300405-      000300405-00
FLD2   MICROP  (CHT,0),8Hƀ*+-$,.0,(MFLC,7),(ENF,8),(INSB,7)
       MICROP  (MVC,2),(INSN,4) memory contents in BCD characters

FLD3   BSS     2               ƀ ƀ ƀ $ 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 - ƀ ƀ ƀ (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
```

****DPS 88 ONLY****

MVNEX	Move Numeric Edited Extended	004 (1)

FORMAT:

00	01	02 08	09 10	11 11	18 Op Code 27	28	29 35
EA	0	MF3	EIT	MF2	004 (1)	I	MF1

00 17	18 20	21	22 23	24 29	30 35
Y1	CN1	TN1	SX1	not interpreted	N1

00 17	18 20	21 22	23	24 29	30 35
Y2	CN2	TA2	0	not interpreted	N2

00 17	18 20	21 22	23	24 29	30 35
Y3	CN3	TA3	0	not interpreted	N3

PROCESSOR MODE: Any

SUMMARY:

$$C(\text{string 1}) \xrightarrow{\text{string 2 control}} C(\text{string 3})$$

The function of this instruction is similar to the MVNE instruction, but with the added capability of allowing the specification of the coded character set (ASCII or EBCDIC) used in the input data, and the coded character set (ASCII, EBCDIC, or BCD) used in initializing the Edit Insertion Table (EIT) for output. Bit 0 (EA) of the instruction specifies the coded character set for the input data (0=EBCDIC, 1=ASCII). Bits 9 and 10 (EIT) specify the coded character set for initializing the EIT as follows:

EIT	
00	EBCDIC
01	BCD
10	ASCII
11	BCD

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 9 and 10 of the instruction. SX1 determines the location of the sign of the input data (leading, trailing, overpunched, separate).

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. Overpunched signs - The MVNEX instruction does not derive appropriate information from ASCII input data with overpunched signs. Incorrect result data or an IPR fault occurs. Such data must be moved via the MVNX instruction (or via other processing) to produce data with separate signs or no signs.

3. Refer to "Micro-Operations" for additional information.

4. An Illegal Procedure fault occurs if DU or DL modifications are specified for MF1, MF2, or MF3, or if illegal repeats are used.

****

****DPS 88 ONLY****

MVNX	Move Numeric Extended	340 (1)

FORMAT:

00	01	02 08	09	10	11 17	18 Op Code 27	28	29 35
EA	NS	0	T	RD	MF2	340 (1)	I	MF1

00 17	18 20	21	22 23	24 29	
Y1	CN1	TN1	SX1	SF1	N1

00 17	18 20	21	22 23	24 29	
Y2	CN2	TN2	SX2	SF2	N2

PROCESSOR MODE: Any

SUMMARY: C(string 1) --> C(string 2)

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. 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.

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 if DU or DL modifications are specified for MF1 or MF2, or if 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.

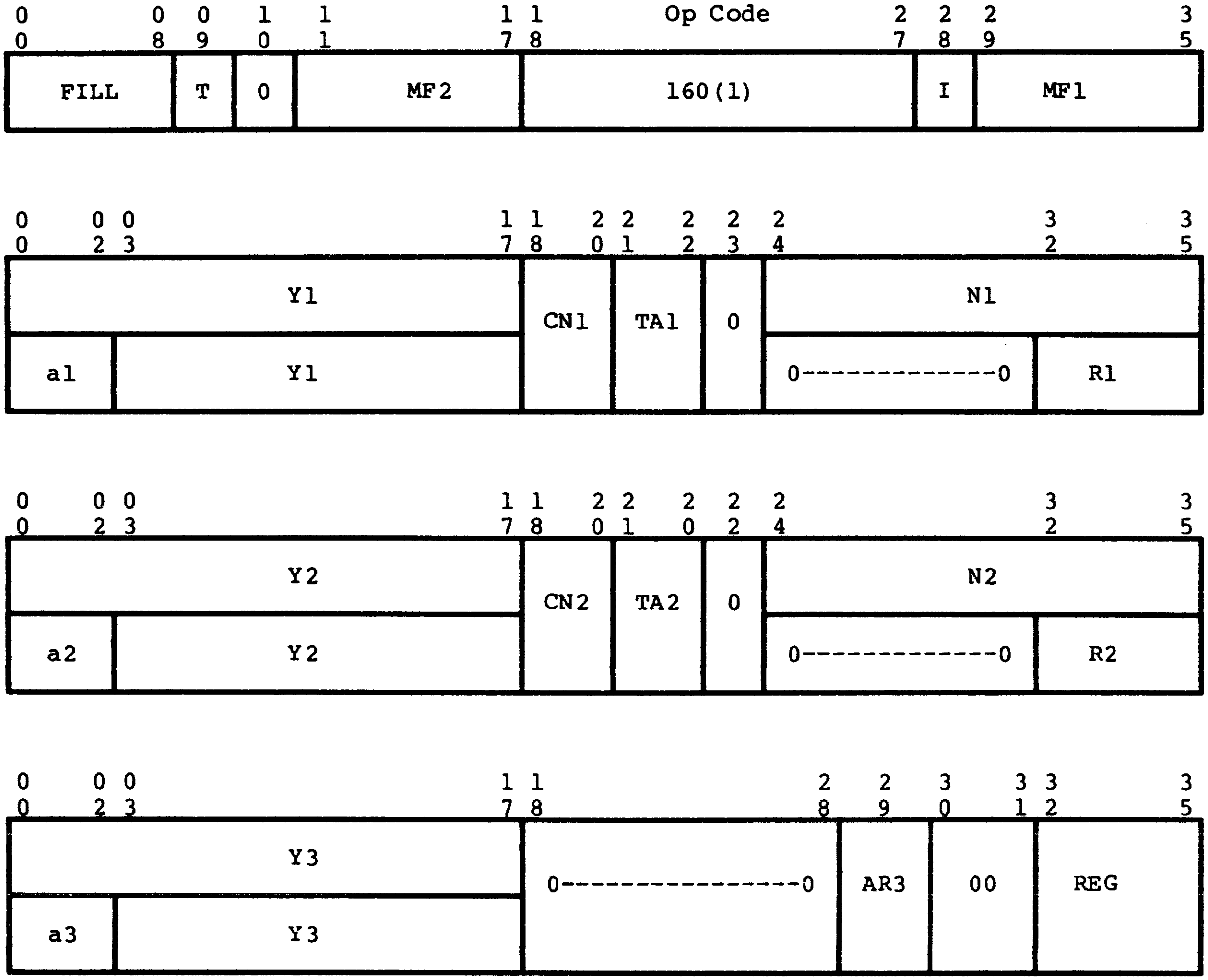
7. The hardware recognizes an implied plus sign on input data. For unpacked data (TN=0) with indicated overpunched sign (SX1 = 00 or 11), if the hardware does not find a plus or minus overpunched sign character in the overpunched sign character position, the hardware checks for a numeric digit (0-9). 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.

The following table shows the 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

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
```

PROCESSOR MODE: Any

SUMMARY:

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. The hardware is not responsible for results, nor can it guarantee identical results on future machines, if any overlap is defined for the three strings.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and REG field for Y3

ILLEGAL REPEATS:

RPT, RPD, RPL cause an Illegal Procedure fault.

INDICATORS:

Truncation - If L1 is greater than L2, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1, MF2, or REG fields for Y3. A Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is a 1.

2. Refer to Note 3 of the MLR instruction for information on string replication.

3. ****DPS 8/20 and 8/44: When pre-paging, the hardware assumes that the length of the translate table corresponds to the data type identified by TA1 as follows:

TA1	Table Length
4-bit	4 words
6-bit	16 words
9-bit	128 words ****

4. L2 = 0 does not necessarily mean that the instruction functions as a NOP; because, the Truncation indicator may be affected.

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,bbbbb123456       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
```

NOTE: The translation table length is determined by the highest possible index character octal value that may be found in the indexing data string.

```
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,ƀABCDEFG      2X
       UASCI   2,HI&.](<\      3X
       UASCI   2,^JKLMNOP      4X
       UASCI   2,QR-$*);´      5X
       UASCI   2,/STUVWX       6X
       UASCI   2,YZ_,%="!      7X
       EDITP   RESTORE
       USE
```

NOTES:

1. The translation table length is determined by the highest octal value for the characters of the indexing string (Field 1). 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.

2. 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 7-1)

CODING FORMAT:

```
1      8       16
______________________________
       NARn    LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: For n = 0,1,...,or 7 as determined by op code

$C(Y)_{0-17} \rightarrow C(ARn)_{0-17}$

$C(Y)_{18-20} \xrightarrow{\text{translated}} C(ARn)_{18-23}$; C(Y) unchanged

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

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****DPS 88, DPS 8/20 and 8/44: Illegal Procedure fault occurs if descriptor CN field contains xx1 for TN = 0.****

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 7-1)

PROCESSOR MODE: Any

SUMMARY: $- C(A) \dashrightarrow C(A)$ if $C(A) \neq 0$

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPL causes IPR fault

INDICATORS:

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: 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.

NEGL	Negate Long (AQ-Register)	533 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: $- C(AQ) \dashrightarrow C(AQ)$ if $C(AQ) \neq 0$

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPL causes IPR fault

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: This instruction changes the number in AQ to its negative (if ≠ 0). The operation is executed by forming the twos complement of the string of 72 bits.

NOP	No Operation	011 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: No operation takes place; the effective address is always prepared.

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: The use of Indirect then Tally modifiers ID, DI, IDC, DIC, SCR, or SC causes changes in the address and tally fields of the referenced indirect words; the Tally Runout indicator may be set ON.

NOTES:

1. 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.

2. An Illegal Procedure fault occurs when an illegal repeat is used.

ORA	OR to A-Register	275 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: 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

ORSA	OR to Storage from A-Register	255 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: Any

SUMMARY:
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

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS:
RPL
RPT or RPD of ORSX0

INDICATORS:
Zero - If $C(Y)_{0-17}$ = 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.

ORXn	OR to Index Register n	26n (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: 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

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of ORX0

INDICATORS:

Zero - If C(Xn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. DL modifications 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 7-1)

PROCESSOR MODE: Privileged Master Mode or Master Mode

SUMMARY: Modify bound field of the argument stack register (ASR).

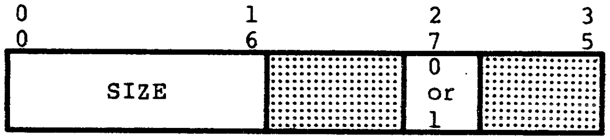
ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. This instruction provides a means of modifying the bound field of the ASR. The one-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 one; that is, 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 with 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.

2. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, RPL cause an IPR fault. An IPR fault is also generated if the execution of this instruction is attempted when the processor is not in the Privileged Master or Master mode.

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 7-1)

PROCESSOR MODE: Any

SUMMARY: No operation takes place

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The PULS1 instruction is identical to the NOP instruction except that it causes certain unique synchronizing signals to appear in the processor logic circuitry.
2. Attempted repetition with the RPT, RPD, or RPL instruction causes an IPR fault.

PULS2	Pulse Two	013 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: No operation takes place

ILLEGAL ADDRESS MODIFICATIONS: None

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The PULS2 instruction is identical to the NOP instruction except that it causes certain unique synchronizing signals to appear in the processor logic circuitry.

2. Attempted repetition with the RPT, RPD, or RPL instructions causes an IPR fault.

QLR	Q-Register Left Rotate	776 (0)

FORMAT:	Single-word instruction format (see Figure 7-1)
PROCESSOR MODE:	Any
SUMMARY:	Rotate C(Q) left by the number of positions indicated by bits 11-17 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 7-1)

PROCESSOR MODE: Any

SUMMARY: Shift C(Q) left by the number of positions indicated by bits 11-17 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(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

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.

QRL	Q-Register Right Logical Shift	772 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Shift C(Q) right by the number of positions indicated by bits 11-17 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(Q) = 0, then ON; otherwise, OFF
Negative	- If $C(Q)_0$ = 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.

QRS	Q-Register Right Shift	732 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Shift C(Q) right by the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with bit 0 of C(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 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.

****DPS 88 ONLY****

RCCL	Read Calendar Clock	633 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Calendar Clock) --> C(AQ)$_{0-71}$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The operand address development is allowed to proceed but does not affect the instruction.

2. Processor port selection (which CIU) is determined by bit 23 (Control CIU) of the Option Register. This control CIU bit can be changed by the SSF, or by the LDHC instruction in Hyper mode, if reconfiguration requires the use of an alternate Port-CIU-Clock. The Calendar Clock can be loaded via the privileged LCCL instruction.

3. The Calendar Clock counts in units of one microsecond.

4. The Calendar Clock is initially loaded by the SSF (SMAS) with the value that is the number of microseconds that have elapsed since 00:00 hours, Greenwich Mean Time (GMT), January 1, 1901.

****

RET	Return	630 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: $C(Y)_{0-17}$ --> C(IC); $C(Y)_{18-35}$ --> C(IR); C(Y) unchanged

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. The relation between the 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	0
32	Hexadecimal
33-35	000

2. The handling of the master mode indicator is described under Indicator, above.

3. The Tally Runout indicator will reflect bit 25 of C(Y) regardless of any address modification performed on the RET instruction (for tally operations).

4. The RET instruction does not load the instruction segment register (ISR) and the SEGID(IS). The return is always within the current instruction segment.

5. The RET instruction may be thought of as an LDI instruction followed by a transfer to the location specified by $C(Y)_{0-17}$.

6. 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.

7. ****DPS 88: The RET instruction does not function properly if it is placed in a fault vector or interrupt vector.****

8. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****DPS 88 ONLY****

RIMR	Read Interrupt Mask Register	233 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(Interrupt Mask Register)$_{0-7}$ --> C(A)$_{0-7}$
000....000 --> C(A)$_{8-35}$

CPU port selection (which CIU) is determined by bit 23 (Control CIU) of the Option Register

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR. Address modifications have no effect on the operation but are performed by the hardware.

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The use of this instruction in other than Privileged Master mode causes an IPR Fault.
2. A CPU cannot read the Interrupt Mask Register in a CIU port which is not assigned to that CPU.
3. In DPS 8 processors, the mnemonic RMCM (Read Memory Controller Mask Register) was assigned to operation code 233(0). The mnemonic has been changed to reflect the change in functionality.
4. The Interrupt Mask Register is only loaded into the A-register, rather than the A- and Q-registers.
5. The effective address is not used by the RIMR instruction.
6. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

****DPS 88 ONLY****

RIW	Read Interrupt Word Pair	412 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: If any unmasked interrupt queue has an entry, then C(Word Pair Entry from Interrupt Queue) --> C(AQ). The Control CIU´s Interrupt Queue Base Register and internal queue pointers are used for locating the oldest entry in the highest priority unmasked interrupt queue. No CPU address information is used. The entry from the interrupt queue contains the level number.

If no unmasked interrupt queue has an entry, then C(Y + Reserved Memory Base Register) --> C(AQ). The effective address Y is added to the Reserved Memory Base Register, and the resulting address is used to read the contents of a Reserved Memory location with no paging. This Reserved Memory location should be established by convention as the "null" word pair location.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, RI, IR, IT

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The use of this instruction in other than Privileged Master mode causes an IPR fault.

2. The interrupt level bit is reset provided no other entry remains in the queue for that level. Information returned is for the highest priority interrupt level present in the ICR. (ICR bit 0 has highest priority, bit 7 has lowest priority.)

3. An Illegal Procedure fault occurs if illegal address modification or illegal repeats are used.

4. The "null" word pair location and contents are defined by C(Reserved Memory Base Register) plus the effective address.

   Null Word Pair Contents (to be initialized by SSF):
   1st word = constant value that is different from any valid queue entry, 2nd word = "don´t care".

5. CPU port selection (which CIU) is determined by bit 23 (Control CIU) of the Option Register.

6. Queue Entry Format:

   Each queue entry is a word pair. The first word has the following format:

INTERRUPT REPORT WORD

00 08	09 17	18 23	24	25 26	27 28	29 31	32 35
COMMAND	CHANNEL NUMBER	RESERVED FOR FUTURE USE	CIU	IOX	RFU	LVL	SYS ID

2nd word - level 1 (fault status):

00	01 03	04 16	17	18 26	27 35
1	IN	ZERO		IOX OR CHANNEL FAULT STATUS	TRANSACTION

CIU STATUS

2nd word - level 7 (special status): This word will contain information from the device controller.

2nd word - other levels - undefined.

****

****DPS 8 ONLY****

RMCM	Read Memory Controller Mask Register	233 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY:

$$\left.\begin{array}{l}\text{C (Memory Controller Interrupt Mask Register)}\\ \text{C (Memory Controller Access Mask Register)}\\ \text{of Memory Unit specified by bits 0-2 of Y}\end{array}\right\} \dashrightarrow C(AQ)$$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: ****DPS 8/20 and 8/44 ONLY: RPT, RPD, RPL****

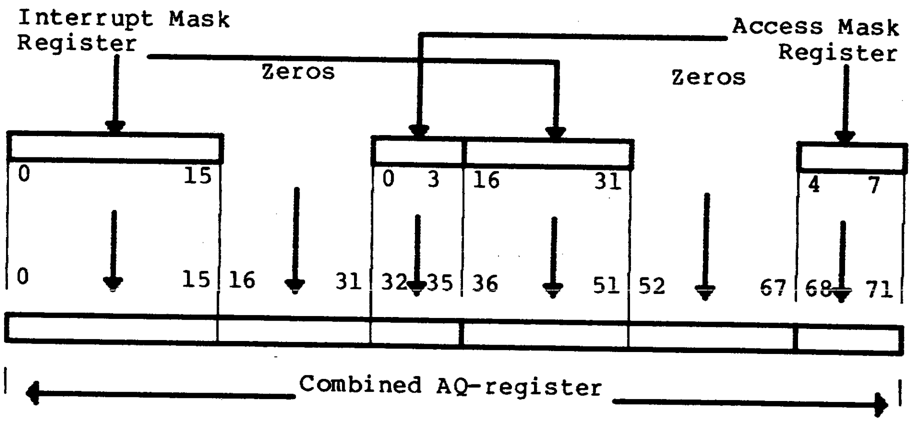
INDICATORS:

Zero - If C(AQ) = 0, then ON; otherwise, OFF

Negative - If $C(AQ)_0$ = 1, then ON; otherwise, OFF

NOTES:

1. The effective address Y is used in selecting a memory module as with a normal memory access request. However, the selected module does not transmit the contents of an addressed memory location, but the contents of its Memory Controller Interrupt Mask Register (IMR) and Memory Controller Access Mask Register (AMR).



2. If the use of this instruction is attempted by a processor in the Slave mode, a fault occurs.

3. If the processor has no mask register assigned to it, then zeros are returned to C(AQ).

4. 1´s in C(AQ) indicate interrupt cells or ports which are masked.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

RPD	Repeat Double	560 (0)

FORMAT:

00 07	08	09	10	11 17	18 Op Code 26	27	28	29	30 35
TALLY	A	B	C	TERM. COND.	560(0)	0	1	0	DELTA

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 k1,k2,...,k7 is 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.

PROCESSOR MODE: Any

SUMMARY: 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.

ILLEGAL ADDRESS MODIFICATIONS: No modifications are allowed

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. 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 = 1, then bits 0-17 of the RPD instruction --> C(X0).

   ****DPS 88: This occurs prior to any detection of an IPR fault that may occur on the instructions to be repeated.****

3. 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.

4. The repetition cycle consists of the following steps:

   a. Execute the pair of repeated instructions.

   b. Bits 0-7 of C(X0) - 1 --> bits 0-7 of C(X0).

   c. If a terminate condition is met (see 7b), set the Tally Runout indicator OFF and exit.

   d. If bits 0-7 of C(X0) = 0, set the Tally Runout indicator ON and exit.

   e. Go to (a).

5. 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 if a particular instruction can be repeated.

6. Address modification for the pair of repeated instructions:

   For each of the two repeated instructions, only the modifiers R and RI and only the designators specifying X1,...,X7 are permitted.

   All other modifier designations result in an IPR fault.

   ****DPS 88, DPS 8/20 and 8/44: AR modification is permitted.****

The effective address Y (for R) or the address YI of the indirect word being referred to (for RI) with bit 29 = 0 (no AR modification) is:

a. For the first execution of each of the two repeated instructions:

   Y + C(R) --> Y_1 or YI_1

   Y_1 or YI_1 --> C(R)

b. For any subsequent execution of the first of the two repeated instructions:

   If A=1, then DELTA + C(R) --> Y_n or YI_n

   Y_n or YI_n --> C(R)

   If A=0, then C(R) --> Y_n or YI_n, where n>1

c. For any subsequent execution of the second of the two repeated instructions:

   If B=1, then DELTA + C(R) --> Y_n or YI_n;
   Y_n or YI_n --> C(R)

   If B=0, then C(R) --> Y_n or YI_n, where n>1

The effective address Y (for R) or the address YI of the indirect word being referred to (for RI) with bit 29 = 1 (AR modification) is:

****DPS8/70,8/50,8/52,8/62: Bit 29=1 causes an IPR fault****

a. 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)

b. For any subsequent execution of the first of the two repeated instructions:

   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

c. For any subsequent execution of the first of the two repeated instructions:

   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

   where: se - sign extended

   A and B - the contents of bits 8 and 9 of index register 0 (X0)

ARm	- address register m selected by instruction bits 0,1, and 2

In the case of RI, only one indirect reference is made per repeated execution. The tag field of the indirect word is not interpreted as usual but is ignored. Instead, the modifier R and the designator R = N are applied.

7. 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 the 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. (See 7-c below.)
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 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.

****DPS 88: The IC reported with this fault points to the RPD instruction and not the instruction being repeated. This is required for restart purposes.

8. Upon exit from the repetition cycle:

Bits 0-7 of C(X0) contain the tally residue; that is, 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 specified by the designator of each of the two repeated instructions will contain either:

a. The contents of the designated Xn 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 after the last execution of the repeated pair if A or B, respectively, is zero.

If the exit was due to a fault, the Xn specified by the designator of each of the two repeated instructions may contain either:

a. The contents of the designated Xns when the fault occurred plus the DELTA associated with each instruction A and B = 1, or

b. The contents of the designated Xns when the fault occurred.

9. A Repeat Double (RPD) 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.

10. The repeated instruction must use index register modification; otherwise, an IPR fault occurs.

11. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

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:

00 07	08 09	10	11 17	18 Op Code 26	27	28	29	30 35
TALLY	00	C	TERM. COND.	500(0)	0	1	0	000000

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.

It is also possible to use an octal number 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.

PROCESSOR MODE: Any

SUMMARY: Execute the next instruction either a specified number of times, until a specified termination condition is met, or until the link address of zero is detected.

ILLEGAL ADDRESS MODIFICATIONS: No modifications are allowed

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. If C = 1, then bits 0-17 of the RPL instruction --> C(X0).

   ****DPS 88: This occurs prior to any detection of an IPR fault that may occur on the instruction to be repeated.****

2. 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.

3. The repetition cycle consists of the following steps:

   a. Execute the repeated instruction.

   b. Bits 0-7 of C(X0) - 1 --> bits 0-7 of C(X0).

   c. If a terminate condition is met (see 6c), set the Tally Runout indicator OFF and exit.

   d. If the tally bits 0-7 of C(X0) = 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. Go to (a).

4. Many instructions cannot be repeated. If an instruction cannot be repeated, an illegal repeat causes an IPR fault to occur. Refer to the individual instruction descriptions to determine if a particular instruction can be repeated.

5. Address modification for the repeated instruction:

   For the repeated instruction, only the modifiers R and RI (the designators specifying R = X1,...,X7) are permitted. The modifier is effective only for the first execution of the repeated instruction.

   ****DPS 88, DPS 8/20 and 8/44: AR modification is permitted and is effective on each execution.****

   The effective address Y with bit 29 = 0 is:

   a. For the first execution of the repeated instruction:

      $Y + C(R) \text{ --> } Y_1$ or YI_1

      Y_1 or YI_1 --> C(R)

   b. For any subsequent execution of the repeated instruction:

      $Yn = C(Yn-1)_{0-17}$

      If $C(Yn-1)_{0-17} \neq 0$

      then $C(Yn-1)_{0-17}$ --> C(R)

The effective address Y with bit 29 = 1 is:

****DPS 8/70: Bit 29=1 is an IPR fault****

a. For the first execution of the repeated instruction:

$(se)Y + C(R) + C(ARm) \text{-->} Y_1$ or YI_1;

Y_1 or YI_1 --> C(R)

b. For any subsequent execution of the repeated instruction:

$Y_n = C(Y_{n-1})_{0-17} + C(ARm)$;

if $C(Y_{n-1})_{0-17} \neq 0$,

then $C(Y_{n-1})_{0-17}$ --> C(R)

where: se - sign extended

ARm - address register m selected by instruction bits 0, 1, 2

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:

Bits 0-17	00...0
Bits 18-35	$C(Y)_{18-35}$
Bits 36-71	$C(Y)_{36-71}$ for double precision

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.

6. The 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 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.

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, then exit from the repetition cycle. If the Overflow Mask indicator is OFF when an overflow occurs, then an overflow fault occurs. See 6-d below.

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.).

****DPS 88: The IC points to the RPL instruction and not the instruction being repeated. This is required for restart purposes.****

7. Upon exit from the repetition cycle:

   Bits 0-7 of C(X0) contain the tally residue; that is, the number of repeats remaining until a tally runout would have occurred. The terminate conditions in bits 11-17 remain unchanged.

   The Xn 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.

8. The repeated instruction must use index register modification; otherwise, an IPR fault occurs.

9. 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.

10. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

```
1      8      16
______________________________
       EAX7   A
       LDA    =3HIDD,DL
       RPL    5,TZE
       CMPA   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:

00 07	08	09	10	11 17	18 Op Code 26	27	28	29	30 35
TALLY	0	0	C	TERM. COND.	520(0)	0	1	0	DELTA

CODING FORMAT: RPT N,I,kl,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 kl,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.

It is also possible to use an octal number rather than the transfer instructions to denote termination conditions. Thus, if the field for kl,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).

PROCESSOR MODE: Any

SUMMARY: Execute the next instruction either a specified number of times or until a specified termination condition is met.

ILLEGAL ADDRESS MODIFICATIONS: No modifications are allowed

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. If C = 1, then bits 0-17 of the RPT instruction --> C(X0).

   ****DPS 88: This occurs prior to any detection of an IPR fault that may occur on the instruction to be repeated.****

2. 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 loop.

3. The repetition cycle consists of the following steps:

   a. Execute the repeated instruction.

   b. $C(X0)_{0-7}$ - 1 --> $C(X0)_{0-7}$.

   c. If a terminate condition is met (see 6b), set the Tally Runout indicator OFF and exit.

   d. $C(x0)_{0-7}$ = 0, then set the Tally Runout indicator ON and exit.

   e. Go to (a).

4. Many instructions cannot be repeated. If an instruction cannot be repeated, an illegal repeat causes an IPR fault to occur. Refer to the individual instruction descriptions to determine if a particular instruction can be repeated.

5. Address modification for the repeated instruction:

   For the repeated instruction, only the modifiers R and RI and only the designators specifying X1,...,X7 are permitted.

   ****DPS 88, DPS 8/20 and 8/44: AR modification is permitted.****

   The effective address Y (for R) or the address YI of the indirect word being referred to (for RI) with bit 29 = 0 is:

   a. For the first execution of the repeated instruction:

      Y + C(R) --> Y_1 or YI_1;

      Y_1 or YI_1 --> C(R)

b. For any subsequent execution of the repeated instruction:

DELTA + C(R) --> Y_n or YI_n;

Y_n or YI_n --> C(R)

The effective address Y (for R) or the address YI of the indirect word being referred to (for RI) with bit 29 = 1 is:

****DPS 8/70: Bit 29=1 causes an IPR fault.****

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)

b. For any subsequent execution of the repeated instruction (A or B = 1):

DELTA + C(R) + C(ARm) --> Y_n or YI_n

DELTA + C(R) --> C(R)

where: se - sign extended

ARm - address register m selected by instruction bits 0, 1, 2

In the case of RI, only one indirect reference is made per repeated execution. The tag field of the indirect word is not interpreted as usual but is ignored. Instead, the modifier R and the designator R = N are applied.

6. The Exit Conditions:

An exit is made from the repeat cycle if one of the terminate conditions exists or if tally is zero 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. 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 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 one 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, then exit from the repetition cycle. If the Overflow Mask indicator is OFF when an overflow occurs, then an Overflow fault occurs. See 6-c below.

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 an overflow occurs (divide check, parity error on indirect word or operand fetch, etc.).

****DPS 88: The IC reported with this fault points to the RPT instruction and not the instruction being repeated. This is required for restart purposes.****

7. Upon exit from the repetition cycle:

   Bits 0-7 of X0 contain the tally residue; that is, the number of repeats remaining until a tally runout would have occurred. The terminate conditions in bit 11-17 remain unchanged.

   If the exit was due to tally = 0 or a terminate condition, the Xn specified by the designator of the repeated instruction contains the contents of the designated Xn after the last execution plus DELTA.

   If the exit was due to a fault, the Xn specified by the designator of the repeated instruction may contain either:

   a. The contents of the designated Xn at the time the fault occurred, or

   b. The contents of the designated Xn at the time the fault occurred, plus DELTA.

   If bits 0-7 of C(X0) are equal to zero, the Tally Runout indicator is set ON; otherwise, OFF.

8. The repeated instruction must use index register modification; otherwise an IPR fault occurs.

9. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

```
1         8        16
          LDA      KEY
          EAX4     TABLE
          RPT      64,1,TZE
          CMPA     0,4
          TZE      FOUND
          .
          .
          .
TABLE     BSS      64
KEY       BSS      1
```

****DPS 88 ONLY****

RRES	Read Reserved Memory	231 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(Y + Reserved Memory Base Register) --> C(A)

The effective address Y is added to the Reserved Memory Base Register. The resulting address is used to read the contents of a Reserved Memory location with no paging.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, RI, IR, IT

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. This instruction is intended primarily for use in Privileged Master mode. The use of this instruction with effective address Y > 7 in Master mode or Slave mode causes a Bound fault.
2. Bit 29 should be filled with zero to ensure compatibility with future systems. The value of bit 29 is ignored - none of the Address Registers, nor any Descriptor Registers is used in the address formation.
3. The Reserved Memory Base Register is initialized by SMAS software to point to location 0 of some memory bank. The contents of the Reserved Memory area are initialized by SMAS to provide configuration reference information for Operating System Startup software.

4. To ensure compatibility with Slave programs that used the RSW instruction, SMAS software initializes and maintains the contents of the Reserved Memory area as follows:

   Word 0 (Data Switches) is set to the value specified to SMAS in OS configuration information or through maintenance console verbs.

   Word 1 (Configuration Switches) is set to zero by SMAS.

   Word 2 (Model Characteristics) is initialized as follows by SMAS, based on configuration information supplied to SMAS:

Bits	Model Characteristics
0-3	Zero
4-5	Processor Type = 11 (DPS 88)
6-11	Fault Base Register (0 modulo 64)
12-17	Zero
18	1 = BCD installed
19	1 = DPS installed
20	1 = Cache installed
21-23	Zero
24	Zero (Program can obtain the decor information via the STO instruction.)
25	1 = NPL peripherals
26-33	Zero
34-35	Processor Number

   Words 3 through 7 are set to zero by SMAS.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

****DPS 8 ONLY****

RSCR	Read System Controller Register	413 (0)

FORMAT: Single-word instruction format (see Figure 7-1).

PROCESSOR MODE: Any

NOTE: In slave mode all addresses cause the Elapsed Time Clock to be selected.

SUMMARY: The final real memory address is used in selecting a system controller and the function to be performed as follows:

Effective Address	Function	
XXXXXX0X	SCU Mode Register (Level 66 only)	
XXXXXX1X	C(Configuration switches) -->	C(AQ)
XXXXX02X	C(Interrupt mask port 0) -->	C(AQ)
XXXXX12X	C(Interrupt mask port 1) -->	C(AQ)
XXXXX22X	C(Interrupt mask port 2) -->	C(AQ)
XXXXX32X	C(Interrupt mask port 3) -->	C(AQ)
XXXXX42X	C(Interrupt mask port 4) -->	C(AQ)
XXXXX52X	C(Interrupt mask port 5) -->	C(AQ)
XXXXX62X	C(Interrupt mask port 6) -->	C(AQ)
XXXXX72X	C(Interrupt mask port 7) -->	C(AQ)
XXXXXX3X	C(Interrupt cells) -->	C(AQ)
XXXXXX4X	C(Elapsed time clock) -->	C(AQ)
XXXXXX5X	C(Elapsed time clock) -->	C(AQ)
XXXXXX6X	C(Mode Register selected store unit) -->	C(AQ)
XXXXXX7X	C(Mode Register selected store unit) -->	C(AQ)

NOTE: X - Address bits not used in determining the selected register, but used in selecting the pertinent system controller.

GCOS 8 Operation (Master & Privileged Master Mode)

Absolute Mode (Working Space Number 0): The Real address (no virtual to real address mapping) is equivalent to the effective address of GCOS III, without Master BAR modification.

Non-Absolute Mode (Working Space Number ≠ 0): The virtual address is generated using all legal tag field modification, address register and descriptor modification. This virtual address is then mapped to a real address. This real address is then the effective address of the instruction.

Slave Mode Operation

The effective address is generated, using the address register modification and any legal tag field modification (the BAR (Base Register) modification is not included in the effective address modification). When the effective address cycle is complete, the CPU shall force an address of 00000040 and cause the Elapsed Time Clock to be read from the System Controller that contains this memory address.

Level 66 System Controller

CONFIGURATION SWITCHES

00 08	09 11	12 15	16 19	20	21	22 29	30	31	32 35
Mask A	Store Size	On-line	Port No.	NU	Mode	Nonexistent Address	Inlace	Lower	Port Mask

36 44	45 56	57 63	64 67	68 71
Mask B	Not Used	Cyclic Priority	Not Used	Port Mask

Field	Code	Meaning
Mask A Mask B	1	A 1 in one or more of bit positions 0 through 7 and 36 through 43 indicates that the corresponding ports will receive all interrupts that are unmasked by interrupt mask registers A and B. Masks A and B may be assigned to more than one port under program control, but both interrupt mask registers A and B must not be assigned to the same port. A 1 in bit position 8 indicates that the interrupt mask register is unassigned and any 1s in bits 0 through 7 are ignored by the system controller.
Store Size	000	32K
	001	64K
	010	128K
	011	256K
	100	512K
	101	1M
	110	2M
	111	4M

Online	1	Store unit is able to accept requests. Bits 12 through 15 correspond to units A, A1, B, and B1, respectively.
Port Number		This field tells the requesting CPU its own port number assignment in the SCU.
Configuration Mode	0	Configuration register is not altered and instruction terminates normally.
	1	All settable bits of configuration register may be altered; not settable under program control.
Nonexistent Address		Logic not active for controller instructions RSCR and SSCR. Active for all others.
Interlace	0	Store selection is based solely on higher address bit (not interlaced).
	1	Stores A and B are selected based on address bit 22 and a higher order address bit determined by store size.
Lower Store	0	Lower address is in store pair A/A1.
	1	Lower address is in store pair B/B1.
Port Masks		Bits 32, 33, 34, 35 and 68, 69, 70, 71 indicate the state of ports 0, 1, 2, 3 and 4, 5, 6, 7, respectively.
	1	Indicates port enabled.
	0	Indicates port OFF.
		These bits may be altered by the SMCM instruction and the SSCR instruction if the configuration mode switch is in the program position.
Cyclic Priority		Ports grouped with equal priorities.
	0	Zero between groups of 1 bit separates priority groups.
	1	Adjacent 1 bits increase the number of ports within a group.

The Read System Controller interrupt mask port _n_ instructions cause the contents of the program interrupt mask register assigned to the specific port of the system controller to be loaded into the AQ-register. In addition, the contents of the port enable register (PER, one per system controller) are presented in the same format as for the RMCM instruction. If no program interrupt mask register is assigned to the port specified, only the PER is returned with the remaining bits as zeros. The format is as follows:

00–15	16–31	32–35	36–51	52–67	68–71
Interrupt Mask Register Bits 0-15	Zeros	PER Bits 0-3	Interrupt Mask Register Bits 16-31	Zeros	PER Bits 4-7

|← C(AQ) →|

The Read System Controller - Interrupt Cells instruction causes the contents of the interrupt cells to be loaded into the AQ-register. This instruction reads (without resetting) the cells. After the execution of this instruction, the AQ-register has the following format:

00–15	16–35	36–51	52–71
Interrupt Cells 0-15	Zeros	Interrupt Cells 16-31	Zeros

|← C(AQ) →|

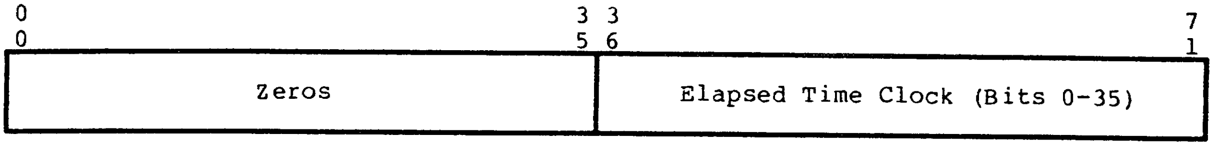
Define layout in groups of four:

0-3 level 0 for IOM 0,1,2,3
4-7 level 1 for IOM 0,1,2,3

Levels 1, 3, 5, and 7 are used for fault, terminate, marker, and special interrupts for channels 0 through 32.

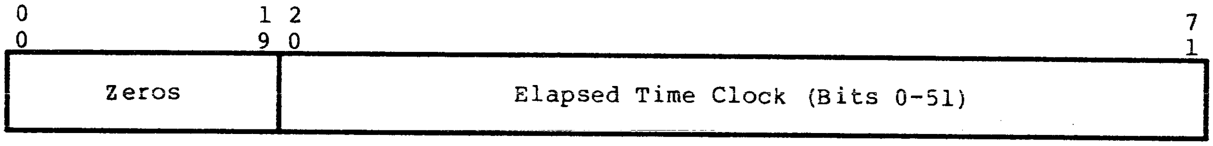
Levels 0, 2, 4 and 6 are used for fault, terminate, marker, and special interrupts for channels 32 through 63.

The Read System Controller - Elapsed Time Clock instruction causes the elapsed time clock register to be read into the AQ-register. Two versions of the elapsed time clock register exist. In early model controllers, the clock, in microsecond increments, is not settable and turns over approximately every 19 hours. This format in the AQ-register is as follows:



(6000 System Controller)

In later model controllers, the clock, in microseconds, is settable and turns over approximately every 142 years. Bits 20-55 are settable. This format in the AQ-register is as follows:



ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. When used with a 6000 system controller, the effective address must be within the lower store otherwise, a Command fault occurs.

2. The execution of the read elapsed time clock function of the RSCR instruction is allowed in Master, Privileged Master, and Slave modes of operation.

3. Port selection is based on the effective address = Y + Xn + AR; therefore, the base value of the descriptor is not added and the virtual to real address translation is not made. However, if bit 29 is 1, the specified address register is added when forming the effective address.

4. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

RSCR	Read System Controller Register	413 (0)

****DPS 88:

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Calendar Clock) --> $C(AQ)_{0-71}$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The operand address development is allowed to proceed but does not affect the instruction. Processor port selection (which CIU) is determined by bit 23 (control CIU) of the Operation Register. This control CIU bit can be changed by the SSF, or by the LDHC instruction in Hyper mode, if reconfiguration requires the use of an alternate Port-CIU-Clock. The calendar clock can be loaded via the privileged LCCL instruction.

2. The RSCR instruction performs the same function as the RCCL instruction, and is included in the repertoire to provide software compatibility.

3. The calendar clock counts in units of one microsecond.

4. The calendar clock is initially loaded by the SSF (SMAS) with the value that is the number of microseconds that have elapsed since 00:00 hours, Greenwich Mean Time (GMT), January 1, 1901.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

****DPS 8 ONLY****

RSW	Read Switches	231 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: The final computed address is used to select certain processor switches whose settings are read into the A-register.

The switches are selected in accordance with the type of processor installed. The configuration of the hardware control switches may vary for each type of processor.

ILLEGAL ADDRESS MODIFICATIONS: None
****DPS 8/20, 8/40: DU, DL, RI, IR, IT****

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: Illegal repeats RPT, RPD, and RPL cause an IPR fault.

****DPS 8/20, 8/40: Illegal address modification causes an IPR fault.****

****

S4BD(X)	Subtract 4-Bit Displacement from Address Register	522 (1)

FORMAT: Special arithmetic instruction format (see Figure 7-3)

CODING FORMAT:

```
1      8         16
_____________________________________________
          S4BD(X) word displacement,R,AR
```

When the mnemonic is coded with an X (S4BDX), bit 29 is forced to zero.

PROCESSOR MODE: Any

SUMMARY: Description is the same as for A4BD except that the formed values are subtracted from the AR.

ILLEGAL ADDRESS MODIFICATIONS: All except N, AU, QU, AL, QL, and index registers

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

```
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(X)	Subtract 6-Bit Displacement from Address Register	521 (1)

FORMAT: Special arithmetic instruction format (see Figure 7-3)

CODING FORMAT:

```
1      8       16
_____________________________________
               S6BD(X) word displacement,R,AR
```

When the mnemonic is coded with an X (S6BDX), bit 29 is forced to zero.

PROCESSOR MODE: Any

SUMMARY: Description is the same as for A6BD except that the formed values are subtracted from the AR.

ILLEGAL ADDRESS MODIFICATIONS: All except N, AU, QU, AL, QL, and index registers

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

```
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(X)	Subtract 9-Bit Displacement from Address Register	520 (1)

FORMAT: Special arithmetic instruction format (see Figure 7-3)

CODING FORMAT:

```
1      8      16
_______________________________________
       S9BD(X) word displacement,R,AR
```

When the mnemonic is coded with an X (S9BDX), bit 29 is forced to zero.

PROCESSOR MODE: Any

SUMMARY: Description is the same as for A9BD except that the formed values are subtracted from the AR.

ILLEGAL ADDRESS MODIFICATIONS: All except N, AU, QU, AL, QL, and index registers

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

```
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 7-1)

CODING FORMAT:

```
1      8      16
______________________________
       SARn   LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: For n=0,1,.., or 7 as determined by op code
C(ARn) --> $C(Y)_{0-23}$; $C(Y)_{24-35}$, 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:

```
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 7-1)

CODING FORMAT:

```
1      8         16
____________________________________
       SAREG     LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: $C(AR0,AR1,\ldots,AR7) \dashrightarrow C(Y,Y+1,\ldots,Y+7)_{0-23}$

Zeros $\dashrightarrow C(Y,Y+1,\ldots,Y+7)_{24-35}$

The hardware assumes bits 15-17 of Y = 000 for the first location. No check is made.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

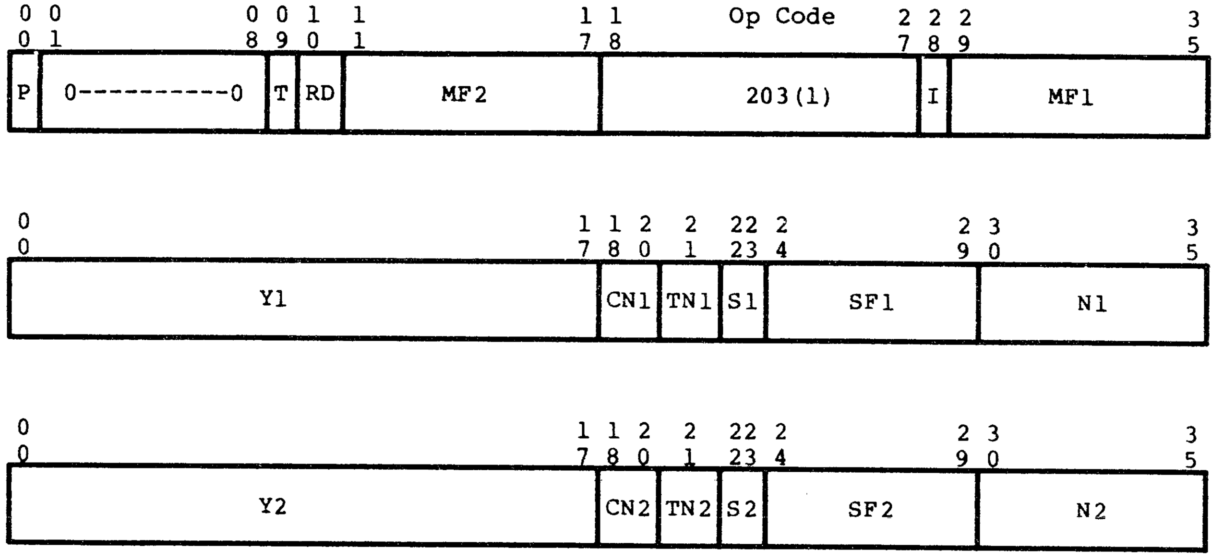
1. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.
2. Location Y must be forced to a multiple of 8 by entering an 8 in column 7 of the statement that defines Y, or by using the EIGHT pseudo-operation.

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
```

PROCESSOR MODE: Any

SUMMARY: C(string 2) - C(string 1) --> C(string 2)

Same as SB3D except that the difference 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 greater than 127, then ON; otherwise, unchanged

Exponent Underflow - If exponent of floating-point result is less than -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. 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 substraction.

      ****DPS 88: Note that DPS 88 accommodates all possible intermediate results without loss of significant digits.****

   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. ****DPS 88: 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
_____________________________________________________________
       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
```

****DPS 88 ONLY****

SB2DX	Subtract Using Two Decimal Operands Extended	243 (1)

FORMAT:

```
 0  0  0         0 0 1  1           1 1    Op Code    2 2 2         3
 0  1  2         8 9 0  1           7 8               7 8 9         5
|EA|NS|    0    |T|RD |     MF2     |     243 (1)    |I|     MF1     |

 0                                  1 1 2 2   2 2 2         2
 0                                  7 8 0 1   2 3 4         9
|               Y1                  |CN1|TN1|SX1|   SF1    |   N1   |

 0                                  1 1 2 2   2 2 2         2
 0                                  7 8 0 1   2 3 4         9
|               Y2                  |CN2|TN2|SX2|   SF2    |   N2   |
```

PROCESSOR: Any

SUMMARY: C(string2) - C(stringl) --> C(string2)

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 AD3D

NOTES:

1. All notes for AD3D apply to SB2DX.
2. See MVNX for information about coding of overpunched signs.

****

SB3D	Subtract Using Three Decimal Operands	223 (1)

FORMAT:

00	01	02 08	09	10	11 17	18 Op Code 27	28	29 35
P	0	MF3	T	RD	MF2	223(1)	I	MF1

00 17	18 20	21	22 23	24 29	30 35
Y1	CN1	TN1	S1	SF1	N1

00 17	18 20	21	22 23	24 29	30 35
Y2	CN2	TN2	S2	SF2	N2

00 17	18 20	21	22 23	24 29	30 35
Y3	CN3	TN3	S3	SF3	N3

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
```

PROCESSOR MODE: 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 scaled 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. 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. 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.

      ****DPS 88: Note that DPS 88 accommodates all possible intermediate results without loss of significant digits.****

   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. ****DPS 88: 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
       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

       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
```

****DPS 88 ONLY****

SB3DX	Subtract Using Three Decimal Operands Extended	263 (1)

FORMAT:

00	01	02 08	09	10	11 17	18 Op Code 27	28	29 35
EA	NS	MF3	T	RD	MF2	263 (1)	I	MF1

00 17	18	20	21	22 23	24 29	35
Y1	CN1	TN1	SX1	SF1	N1	

00 17	18	20	21	22 23	24 29	35
Y2	CN2	TN2	SX2	SF2	N2	

00 17	18	20	21	22 23	24 29	5
Y3	CN3	TN3	SX3	SF3	N3	

PROCESSOR MODE: Any

SUMMARY: C(string 2) - C(string 1) --> C(string 3)

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 scaled 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 EA. Placement of overpunched sign in the output is controlled by NS. If RD is a 1, rounding takes place 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 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 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: 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.

SBD(X)	Subtract Bit Displacement from Address Register	523 (1)

FORMAT: Special arithmetic instruction format (see Figure 7-3)

CODING FORMAT:

```
1      8       16
____________________________________
       SBD(X)  word displacement,R,AR
```

When the mnemonic is coded with an X (SBDX), bit 29 is forced to zero.

PROCESSOR MODE: Any

SUMMARY: Description is the same as for ABD except that the formed values are subtracted from the the AR.

ILLEGAL ADDRESS MODIFICATIONS: All except N, AU, QU, AL, QL, and index registers

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

```
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 7-1)

PROCESSOR MODE: 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

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.

NOTE: This instruction is identical to SBA 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.

SBLAQ	Subtract Logical from AQ-Register	137 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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
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.

NOTES:

1. This instruction is identical to SBAQ 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.

2. 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 7-1)

PROCESSOR MODE: 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
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

NOTE: This instruction is identical to SBQ 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.

SBLXn	Subtract Logical from Index Register n	12n (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0,1,..., or 7 as determined by op code
$C(Xn) - C(Y)_{0-17}$ --> C(Xn); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of SBLX0

INDICATORS:

Zero - If C(Xn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)_0 = 1$, then ON; otherwise, OFF

Carry - If a carry out of bit 0 of C(Xn) is generated, then ON; otherwise, OFF. When the Carry indicator is OFF, the range of Xn has been exceeded.

NOTES:

1. This instruction is identical to SBXn 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.

2. DL modification is flagged as illegal but executes with all zeros for data.

3. 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 7-1)

PROCESSOR MODE: 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

SBXn	Subtract from Index Register n	16n (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0, 1, ..., or 7 as determined by op code
$C(Xn) - C(Y)_{0-17}$ --> C(Xn); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL of SBX0

INDICATORS:

Zero - If C(Xn) = 0, then ON; otherwise, OFF

Negative - If $C(Xn)_0$ = 1, then ON; otherwise, OFF

Overflow - If range of Xn is exceeded, then ON

Carry - If a carry out of bit 0 of C(Xn) is generated, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged as illegal but executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SCD	Scan Characters Double	120 (1)

FORMAT:

00–10	11–17	18–27 Op Code	28	29–35
0--------------0	MF2	120(1)	I	MF1

<table>
<tr><th>00–02</th><th>03–17</th><th>18–20</th><th>21–22</th><th>23</th><th>24</th><th>35</th></tr>
<tr><td colspan="2">Y1</td><td rowspan="2">CN1</td><td rowspan="2">TA1</td><td rowspan="2">0</td><td colspan="2">N1</td></tr>
<tr><td>a1</td><td>Y1</td><td>0-------------0</td><td>R1</td></tr>
</table>

<table>
<tr><th>00–02</th><th>03–17</th><th>18–20</th><th>21–35</th></tr>
<tr><td colspan="2">Y2</td><td rowspan="2">CN2</td><td rowspan="2">not interpreted</td></tr>
<tr><td>a2</td><td>Y2</td></tr>
</table>

<table>
<tr><th>00–02</th><th>03–17</th><th>18–28</th><th>29</th><th>30–31</th><th>32–35</th></tr>
<tr><td colspan="2">Y3</td><td rowspan="2">0----------------0</td><td rowspan="2">AR3</td><td rowspan="2">00</td><td rowspan="2">REG3</td></tr>
<tr><td>a3</td><td>Y3</td></tr>
</table>

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
```

PROCESSOR MODE: Any

SUMMARY:

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 bits 0-17 of the address field of operand descriptor 2 when the REG field of MF2 specifies DU modification. The compare continues until an identical match is found or until the L1-1 tally runs out. 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. Bits 21-35 (or 18-35 if DU modification is specified) of descriptor 2 are not interpreted.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or REG3; 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. For i = 1 to L1-1, compare YC1-1+i with YC2, and compare YC1+i with YC2+1.
2. ****DPS 88: When N1 = 0 or 1, zero is stored in bits 12-35 of Y3 and the Tally indicator is still affected.****
3. The RL bit in 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
___________________________________________________________
       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
```

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
       ADSCn̄     LOCSYM,CN,,AM
       ARG       LOCSYM,RM,AM
```

PROCESSOR MODE: Any

SUMMARY: 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 REG3; 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. For i = 1 to L1-1, compare YC1 + L1 - i with YC2 + 1 and
YC1 + L1 - 1 - i with YC2.

2. **DPS 88: When N1 = 0 or 1, zero is stored in $Y3_{12-35}$ and the Tally indicator is still affected.****

3. The RL bit in 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
       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,b̸,b̸,b̸
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:

<table>
<tr><td>00</td><td>08</td><td>09</td><td>10</td><td>11</td><td>17</td><td>18</td><td>Op Code</td><td>27</td><td>28</td><td>29</td><td>35</td></tr>
<tr><td colspan="2">MASK</td><td>0</td><td>0</td><td colspan="2">MF2</td><td colspan="3">124(1)</td><td>I</td><td colspan="2">MF1</td></tr>
</table>

<table>
<tr><td>00</td><td>02</td><td>03</td><td>17</td><td>18</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>35</td></tr>
<tr><td colspan="4">Y1</td><td colspan="2" rowspan="2">CN1</td><td colspan="2" rowspan="2">TA1</td><td rowspan="2">0</td><td colspan="2">N1</td></tr>
<tr><td colspan="2">a1</td><td colspan="2">Y1</td><td>0-------------0</td><td>R1</td></tr>
</table>

<table>
<tr><td>00</td><td>02</td><td>03</td><td>17</td><td>18</td><td>20</td><td>21</td><td>35</td></tr>
<tr><td colspan="4">Y2</td><td colspan="2" rowspan="2">CN2</td><td colspan="2" rowspan="2">not interpreted</td></tr>
<tr><td colspan="2">a2</td><td colspan="2">Y2</td></tr>
</table>

<table>
<tr><td>00</td><td>02</td><td>03</td><td>17</td><td>18</td><td>28</td><td>29</td><td>30</td><td>31</td><td>32</td><td>35</td></tr>
<tr><td colspan="4">Y3</td><td colspan="2" rowspan="2">0----------------0</td><td rowspan="2">AR3</td><td colspan="2" rowspan="2">00</td><td colspan="2" rowspan="2">REG3</td></tr>
<tr><td colspan="2">a3</td><td colspan="2">Y3</td></tr>
</table>

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
```

PROCESSOR MODE: Any

SUMMARY:

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 until 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. Bits 21-35 (or 18-35 if DU modification is specified) of descriptor 2 are not interpreted.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or REG3; 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 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. 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
______________________________________________________________
       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
```

PROCESSOR MODE: Any

SUMMARY: Same as SCM except start at location YC1 + (L1-1) and progress toward location YC1.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL for MF1 or REG3; 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 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. 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
```

****DPS 8 ONLY****

SCPR	Store Central Processor Register	452 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: The contents of the register specified by the tag field replace the contents of the specified Y-pair. All other tag field values, except as specified below, cause an IPR fault.

Octal Tag	<<Product>>	Register Selection
00	DPS 8/70	Virtual Unit #1 History Register
01	all	Fault Reg.$_{0-35}$; 0 --> C(Y-pair)$_{36-71}$ Then 0 --> Fault Register
03	DPS 8/20 and 8/44	Extended Fault Register$_{0-35}$; 0 --> C(Y-pair)$_{36-71}$
06	DPS 8/70	Mode Register$_{0-35}$; 0 --> C(Y-pair)$_{36-71}$
06	DPS 8/20 and 8/44	Mode Register$_{0-35}$; 0 --> C(Y-pair)$_{36-53}$ Cache Mode Register --> C(Y-pair)$_{54-60}$; 0 --> C(Y-pair)$_{61-69}$; Lockup Fault Register --> C(Y-pair)$_{70-71}$
10	DPS 8/70	Virtual Unit #2 History Register
12	DPS 8/20 and 8/44	Address Trap Register --> C(Y-pair)$_{0-26}$; 0 --> C(Y-pair)$_{27-71}$
15	DPS 8/20 and 8/44	Cache Directory Entry selected by Y$_{7-15}$ --> C(A)
16	DPS 8/20 and 8/44	Associative Memory Directory Entry selected by Y$_{11-17}$ --> C(A)
17	DPS 8/20 and 8/44	Associative Memory Entry selected by Y$_{11-17}$ --> C(A)
20	all	Control Unit History Register
40	DPS 8/70	OU/DU History Register

ILLEGAL ADDRESS MODIFICATIONS: Tag field defines register

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. For tag field values 00, 10, 20, and 40, the history register stored is selected by the current value of a cyclic counter for each unit. The individual cyclic counters are advanced by one count for each execution of the instruction.
2. The use of tag field values other than those defined above causes an IPR fault.
3. Attempted repetition with the RPT, RPD, or RPL instructions causes an IPR fault.
4. Attempted execution by a processor that is in Slave mode causes a Command fault.

****

SDR<u>n</u>	Save Descriptor Register <u>n</u>	11<u>n</u> (1)

FORMAT: As described below

PROCESSOR MODE: Any

SUMMARY: This instruction stores the descriptor from DRn in the next available location of the argument stack.

The Y field of this instruction is not interpreted by hardware. No address bound checks are made. The argument stack is the operand segment.

Fault: If ASR flag bit 28 shows AS missing, a Missing Segment fault is generated;

****DPS 8: If ASR bound + 8 $\geq$ 8192 bytes, an STR fault is generated.****

****DPS 88: If ASR bound + 8 $\geq$ 8192 bytes, a BND fault is generated.****

If ASR flag bit 27 = 1 (bound valid), then
Effective Address = ASR Bound +1

If ASR flag bit 27 = 0 (bound not valid), then
Effective Address = 0

C(DRn) --> C(AS, EA-pair)

If the store into the argument segment does not cause a fault, then continue.

If ASR flag bit 27 = 1 (bound valid), then
ASR Bound + 8 --> ASR Bound

If ASR flag bit 27 = 0 (bound not valid), then
7 --> ASR Bound,
1 --> ASR flag bit 27 (bound valid)

2 --> SEGIDn$_{0-1}$

ASR Bound$_{7-16}$ --> SEGIDn$_{2-11}$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, RI, IR, IT
****DPS 88: None****

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. If a save is attempted to a nonhousekeeping page, an SCL1 fault is generated.

2. A missing Working Space, Missing Segment, or Missing Page fault may occur.

3. An STR fault occurs if there is an attempt to address more than 2^{24} words and either absolute or dense paging address is used; or if ASR Bound + 1 byte ≥ 8192 bytes (before ASR is updated).

   ****DPS 88: A BND fault occurs if there is an attempt to address more than 2^{26} words and either absolute or dense paging address is used; or if ASR Bound + 1 byte ≥ 8192 bytes (before ASR is updated).****

4. An SCL2 fault occurs if there is an attempted working space violation, or if the specified page does not have write permission. The descriptor itself does not require write nor save permissions.

5. An Illegal Procedure fault occurs if illegal repeats (****DPS 8: and illegal address modifications) are used.

EXAMPLE:

(To save and restore DR3)

```
1       3       16                  32
______________________________________________________
        SDR3                        (SAVE)
        STP3    SAVE3
        .
        .
        .
        LDP3    SAVE3               (RESTORE)
        .
        .
        .
SAVE3   BSS     1
```

****DPS 88 ONLY****

SFR	Store Fault Register	452 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(Fault Register) --> $C(Y)_{0-32}$; 0 --> $C(Y)_{33-35}$
C(FR) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The use of this instruction in other than Privileged Master mode causes an IPR fault.
2. The Fault Register is not cleared after storing. The Fault Register is loaded each time a fault trap occurs. Thus no "clear" functionality is required.
3. In DPS 8 processors, the mnemonic SCPR (Store Central Processor Registers) was assigned to operation code 452(0). The mnemonic has been changed to reflect the change in functionality.
4. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

****DPS 8 ONLY****

SMCM	Set Memory Controller Mask Register	553 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY:

C(AQ) --> { C (Memory Controller Interrupt Mask Register)
C (Memory Controller Access Mask Register) of
Memory Unit specified by bits 0-2 of Y
C(AQ), C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

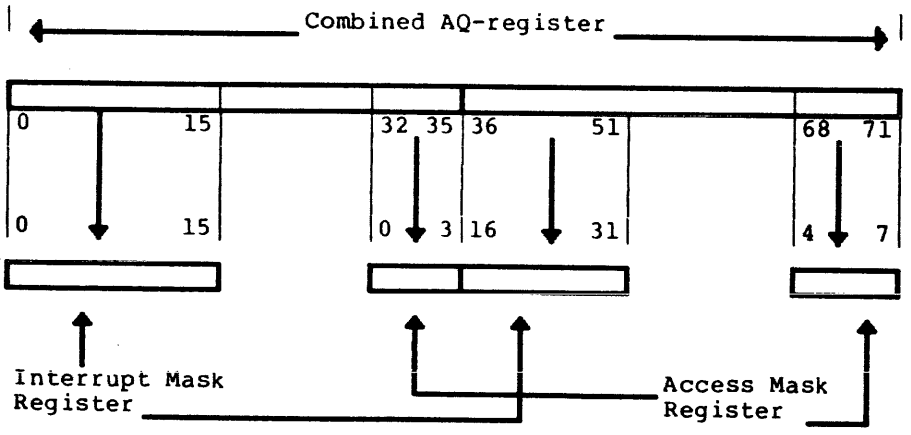
ILLEGAL REPEATS:

****DPS 8/70: RPL****
****DPS 8/20, 8/44: RPT, RPD, RPL****

INDICATORS: None affected

NOTES:

1. The effective address Y is used in selecting a memory module as with a normal memory access request. However, the selected module does not store the data received in a memory location but in its Memory Controller Interrupt Mask Register and Memory Controller Access Mask Register.



2. If the use of this instruction is attempted by a processor in the Slave mode, a Command fault occurs.

3. The address field used to select the SCU (port number) is the absolute page address.

4. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

****DPS 8 ONLY****

SMIC	Set Memory Controller Interrupt Cells	451 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(A) is used to set selected interrupt cells ON in the system controller of the memory unit selected by bits 0-2 of Y; C(A), C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: ****DPS 8/20 and 8/44: RPT, RPD, RPL****

INDICATORS: None affected

NOTES:

1. The effective address Y is used in selecting a memory module as with a normal memory access request. However, the selected module does not store the data received in a memory location, but uses it to set selected interrupt cells ON.

   For i = 0, 1,...,15 and bit 35 of C(A) = 0:

   if bit i of C(A) = 1, then set interrupt cell i ON

   For i = 0, 1,...,15 and bit 35 of C(A) = 1:

   if bit i of C(A) = 1, then set interrupt cell (16+i) ON.

2. If the use of this instruction is attempted by a processor in the Slave mode, a Command fault occurs.

3. The address field used to select the SCU (port number) is the absolute page address.

4. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

SPDBR	Store Page Table Directory Base Register	151 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: ****DPS 8: C(PDBR) --> $C(Y)_{0-14}$

0 --> $C(Y)_{15-35}$ ****

****DPS 88:

C(PDBR) --> $C(Y)_{0-16}$

0 --> $C(Y)_{17-35}$****

C (PDBR) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The contents of the page directory base register (PDBR) replace the contents of Y. The PDBR remains unchanged.

2. Modifications DU, DL, CI, SC, SCR and illegal repeats RPT, RPD, RPL cause an IPR fault.

3. ****DPS 8: If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault. ****

   ****DPS 88: If the processor is not in the Privileged Master mode, the execution of this instruction causes an IPR fault.****

SPL	Store Pointers and Lengths	447 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1      8      16
_____________________________
       SPL    LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: ****DPS 8: C(pointer and length registers) --> C(Y,Y+1,...,Y+7)

Bits 15-17 of Y = 000 for the first location. The actual contents of these bit positions are ignored and are assumed to be zero.****

****DPS 88: C (pointer and length registers) --> C(Y,Y+1). The hardware assumes Y_{17} = 0 ****

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: Multiword Instruction Interrupt indicator (bit 30) OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

2. The SPL instruction provides the capability for storing the pointers for sending and receiving addresses, for sending and receiving field lengths, and for other required control information when an interruptible multiword instruction is interrupted during execution. These registers enable the hardware to resume processing an interrupted instruction after a return from servicing the interrupt. See "Pointer And Length Registers" in Section IV.

3. ****DPS 8: The address register bit of the modification field for the operand descriptor is stored in bit 29. Bits 33-35 are the address registers designated by bits 0-2 of the Y field of the descriptor. Word 3 of the operand stores this data for operand descriptor 1, word 5 of the operand stores this data for operand descriptor 2, and word 7 of the operand stores this data for operand descriptor 3.****

4. ****DPS 8: 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.****

   ****DPS 88: Use E in column 7, or use the EVEN pseudo-operation.****

5. The SPL instruction is normally only used by routines that process interrupts.

6. After an interrupt occurs, the SPL must be executed before any multiword instruction to avoid destruction of the pointer and length information.

EXAMPLE:

```
1         8        16                  32
_____________________________________________________________________________
          SPL      REGWS               store interrupt registers
           .
           .
           .
REGWS     BSS                          WD 0   0 0 0 z/n Tally Counter          - IR1
                                         +1   0 0 0 z/n Tally Counter          - IR2
                                         +2   Desc. 1 Pointer Control Data     - IR3
                                         +3   Level 0 Descr. 1 Len. Res.       - IR4
                                         +4   Descr. 2 Pointer Control Data    - IR5
                                         +5   0 0 0 0 Descr. 2 Len. Res.       - IR6
                                         +6   Descr. 3 Pointer Control Data    - IR7
                                         +7   0 0 0 0 Descr. 3 Len. Res.       - IR8

                                       indicator affected? - interrupt set OFF
```

SREG	Store Registers	753 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(X0,...,X7,A,Q,E,TR) --> C(Y,...,Y+7)
where Y_{15-17} = 000

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 --> $C(Y+7)_{27-35}$

Registers unchanged

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 7-1)

PROCESSOR MODE: 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)

****DPS 8 ONLY****

FORMAT: Single-word instruction format (see Figure 7-1)

This instruction is used with the Level 66 (four-megaword) System Controller.

PROCESSOR MODE: Privileged Master Mode

SUMMARY: The SSCR instructions are the inverse of the RSCR instructions. These instructions are executed on the Level 66 System Controller regardless of the state of the TEST/NORMAL switch. The address codes are as follows:

Octal Address	Registers
X0000X	C(AQ) --> SCU Mode Register
X0001X	C(AQ) --> Configuration Switches
X0002X	C(AQ) --> Interrupt Enable Register, port 0
X0012X	C(AQ) --> Interrupt Enable Register, port 1
X0022X	C(AQ) --> Interrupt Enable Register, port 2
X0032X	C(AQ) --> Interrupt Enable Register, port 3
X0042X	C(AQ) --> Interrupt Enable Register, port 4
X0052X	C(AQ) --> Interrupt Enable Register, port 5
X0062X	C(AQ) --> Interrupt Enable Register, port 6
X0072X	C(AQ) --> Interrupt Enable Register, port 7
X0003X	C(AQ) --> Interrupt Cells
X0004X	C(AQ) --> Elapsed Time Clock
X0005X	C(AQ) --> Elapsed Time Clock
X0006X	C(AQ) --> Mode Register - Selected Store Unit
X0007X	C(AQ) --> Mode Register - Selected Store Unit

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: ****DPS 8/20 and 8/44: RPT, RPD, RPL****

INDICATORS: None affected

NOTES:

1. If the processor does not have a mask register assigned in the selected system controller, an STR fault (not control) occurs.

2. For computed addresses X0006X and X0007X, store unit selection is done by the normal address decoding function of the system controller.

3. The address field used to select the SCU (port number) and the register is the absolute page address.

4. A Command fault occurs if execution of this instruction is attempted by a processor in Slave mode.

5. An Illegal Procedure fault occurs if illegal address modifications (DPS 8/20, 844: or illegal repeats) are used.

6. Refer to the RSCR instruction for System Controller formats.

****

SSQ	Subtract Stored from Q-Register	156 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0,1,..., or 7 as determined by op code
$C(Xn) - C(Y)_{0-17}$ --> $C(Y)_{0-17}$; C(Xn) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL
RPT or RPD of SSX0

INDICATORS:

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

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 7-1)

PROCESSOR MODE: 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.

****DPS 88 ONLY****

STAC	Store A Conditional	354 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: If C(Y) = 0, then C(A) --> C(Y)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If initial C(Y) = 0, then ON; otherwise OFF

NOTES:

1. ****DPS 8: STAC causes an IPR fault in these processors.****

2. If the initial C(Y) is nonzero, then C(Y) is not changed by the STAC instruction.

3. LDAC, LDQC, SZNC, STAC, and STACQ are the only instructions that can be used for the indivisible test-and-set operations which are required for setting and releasing locks, or for closing and opening gates.

   Since execution of LDAC, LDQC, SZNC, STAC, and STACQ depends on the previous C(Y), the processor will obtain ownership of the 8-word block containing C(Y) prior to using C(Y) to execute the instruction. Obtaining ownership of the 8-word block means that the requesting processor, and the Memory Hierarchy Control of the CIU, will ensure that a valid copy of the block is obtained, and that the block is cleared from the cache of all other processors before the instruction is executed. After obtaining ownership of the block, the processor completes execution of the instruction to set or release the lock without permitting the block to be siphoned to another processor. Thus the block is isolated in a time window where it can be accessed and modified only by the processor executing the instruction which sets or releases the lock.

To ensure that a lock does not get released before the actual completion of all stores performed while the lock was set, a synchronizing function is necessary. This synchronizing function is accomplished by coding a SYNC or STC2 instruction immediately before the instruction which releases the lock. If the value stored by STC2 is consistent with operating system conventions for a released lock, then the use of STC2 for synchronizing can also serve to release the lock.

4. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

****DPS 88 ONLY****

STACQ	Store A Conditional on Q	654 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: If C(Y) = C(Q), then C(A) --> C(Y)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL

INDICATORS: Zero - If initial C(Y) = C(Q), then ON; otherwise OFF

NOTES:

1. If the initial C(Y) is ≠ C(Q), then C(Y) is not changed by the STACQ instruction.

2. LDAC, LDQC, SZNC, STAC, and STACQ are the only instructions that can be used for the indivisible test-and-set operations which are required for setting and releasing locks, or for closing and opening gates.

   Since execution of LDAC, LDQC, SZNC, STAC, and STACQ depends on the previous C(Y), the processor will obtain ownership of the 8-word block containing C(Y) prior to using C(Y) to execute the instruction. Obtaining ownership of the 8-word block means that the requesting processor, and the Memory Hierarchy Control of the CIU, will ensure that a valid copy of the block is obtained, and that the block is cleared from the cache of all other processors before the instruction is executed. After obtaining ownership of the block, the processor completes execution of the instruction to set or release the lock without permitting the block to be siphoned to another processor. Thus the block is isolated in a time window where it can be accessed and modified only by the processor executing the instruction which sets or releases the lock.

To ensure that a lock does not get released before the actual completion of all stores performed while the lock was set, a synchronizing function is necessary. This synchronizing function is accomplished by coding a SYNC or STC2 instruction immediately before the instruction which releases the lock. If the value stored by STC2 is consistent with operating system conventions for a released lock, then the use of STC2 for synchronizing can also serve to release the lock.

3. 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 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(ASR) --> C(Y-pair); C(ASR) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. 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.

2. Modifications DU, DL, CI, SC, SCR and illegal repeats RPT, RPD, RPL cause an IPR fault.

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 7-1)

PROCESSOR MODE: Any

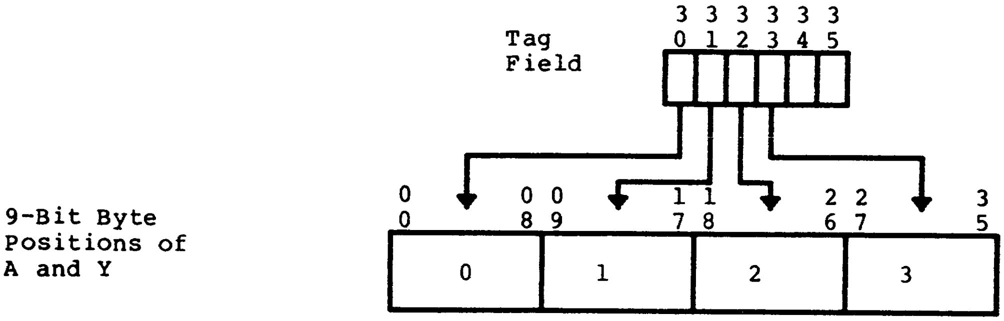
SUMMARY: 9-bit bytes of C(A) --> corresponding characters of C(Y); the byte positions affected are specified in the tag field; C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS: All modifications except for address register

ILLEGAL REPEATS: RPT, RPD, RPL cause an IPR to occur

INDICATORS: None affected

NOTE: Binary 1s 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 two-digit octal number. Bit positions 34 and 35 are ignored.



EXAMPLE: The instruction STBA LOC,04 moves byte 3 from 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 7-1)

PROCESSOR MODE: 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) unchanged

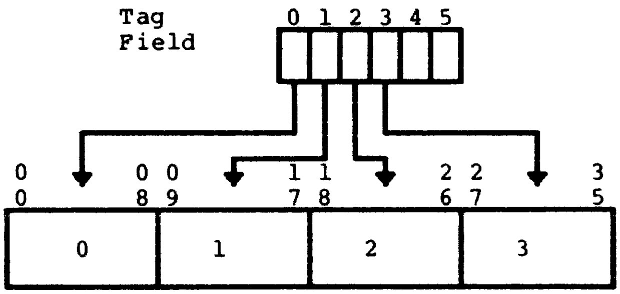
ILLEGAL ADDRESS MODIFICATIONS: All modifications except for address register

ILLEGAL REPEATS: RPT, RPD, RPL cause an IPR to occur

INDICATORS: None affected

NOTE: Binary 1s 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 two-digit octal number. Bit positions 34 and 35 are ignored.

9-Bit Byte Positions of Q and Y



EXAMPLE: The instruction STBQ LOC,04 moves byte 3 from C(Q) to the corresponding byte position of C(LOC) (04 octal = 000100 binary). All other byte positions of C(LOC) are unaffected.

****DPS 88 ONLY****

STBZ	Store Block of Zeros	257 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: 0...0 --> C(Y,...,Y+7)
where bits 15-17 of Y are forced to zero for the first location and then incremented through the block of eight words.

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. This instruction is performed using a block store, in which the processor declares ownership of the block and then stores the zeros into a block in the O-cache without first reading the addressed block from the memory hierarchy. The block is not forced to memory from the O-cache. The use of the CCAC1 instruction, or the natural displacement mechanism of the cache causes the block to be written to memory.

2. This instruction has the following purposes:

   o It provides a means for initializing some or all of main memory with correct EDAC.

   o It may provide a performance advantage when clearing cache/memory.

   o It allows an operating system to clear main memory blocks which have uncorrectable EDAC errors prior to giving the memory to the maintenance software.

3. The segment containing Y must start at a 0 mod 8 boundary.

4. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

STC1	Store Instruction Counter Plus 1	554 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(IC) + 1 --> C(Y); C(IR) --> $C(Y)_{18-32}$; 000 --> $C(Y)_{33-35}$; C(IC), C(IR) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. 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	0
32	Hexadecimal
33-35	000

2. The ON state corresponds to a 1 bit; the OFF state corresponds to a 0 bit.

3. 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).

4. 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 7-1)

PROCESSOR MODE: Any

SUMMARY: $C(IC) + 2 \rightarrow C(Y)_{0-17}$; $C(Y)_{18-35}$, C(IC) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. ****DPS 88, DPS 8/50, 8/52, 8/62, and 8/70:
Execution of STC2 is delayed until all other outstanding writes have been initiated by the System Control Unit. This delay provides a synchronizing function which is required at the end of a block of gated code, immediately preceding the operation that opens that gate. Otherwise the hardware will not guarantee that all stores preceding the gate opening have been completed. STC2 combines the synchronizing function with the gate-opening function.****

2. ****DPS 88:
LDAC, LDQC, SZNC, STAC, and STACQ are the only instructions that can be depended upon for the indivisible test-and-set operations which are required for setting and releasing locks, or for closing and opening gates.

   Since execution of LDAC, LDQC, SZNC, STAC, and STACQ depends on the previous C(Y), the processor will obtain ownership of the 8-word block containing C(Y) prior to using C(Y) to execute the instruction. Obtaining ownership of the 8-word block means that the requesting processor, and the Memory Hierarchy Control of the CIU, will ensure that a valid copy of the block is obtained, and that the block is cleared from the cache of all other processors before the instruction is executed. After obtaining ownership of the block, the processor completes execution of the instruction to set or release the lock without permitting the block to be siphoned to another processor. Thus the block is isolated in a time window where it can be accessed and modified only by the processor executing the instruction which sets or releases the lock.

To ensure that a lock does not get released before the actual completion of all stores performed while the lock was set, a synchronizing function is necessary. This synchronizing function is accomplished by coding a SYNC or STC2 instruction immediately before the instruction which releases the lock. If the value stored by STC2 is consistent with operating system conventions for a released lock, then the use of STC2 for synchronizing can also serve to release the lock.****

3. 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 7-1)

PROCESSOR MODE: Any

SUMMARY: Characters of C(A) --> corresponding characters of C(Y); the character positions affected are specified in the tag field; C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS: All modifications except for address register

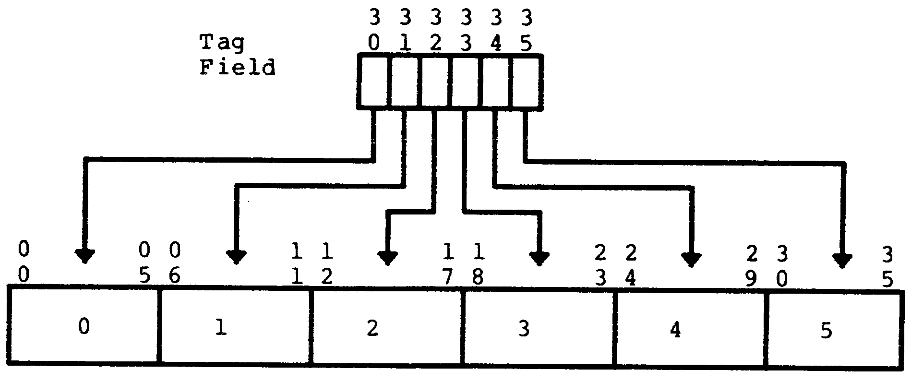
ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Binary 1s in the tag field specify the character positions of A and Y affected as indicated in the diagram below. The tag field is entered as one two-digit octal number.

6-Bit Character Positions of A and Y



For 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.

2. ****DPS 88:
The processor does not zone store to memory. Thus, in executing this instruction, the processor reads the word from memory, updates the specified character position, and writes the word back to memory. This is accomplished as two separate memory operations and no memory lock is invoked on the read.****

3. An Illegal Procedure fault occurs if illegal repeats are used.

STCQ	Store 6-bit Characters of Q-Register	752 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Characters of C(Q) --> corresponding characters of C(Y); the character positions affected are specified in the tag field.

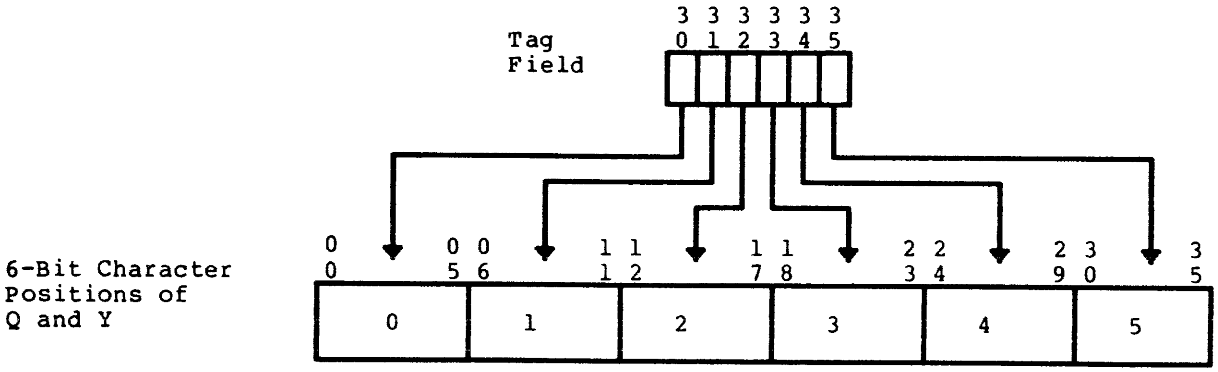
ILLEGAL ADDRESS MODIFICATIONS: All modifications except for address register

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Binary 1s in the tag field specify the character positions of Q and Y affected as indicated in the diagram below. The tag field is entered as one two-digit octal number.



For 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.

2. ****DPS 88:
The processor does not zone store to memory. Thus, in executing this instruction, the processor reads the word from memory, updates the specified character position, and writes the word back to memory. This is accomplished as two separate memory operations and no memory lock is invoked on the read.****

3. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STDn	Store Descriptor Register n	05n (1)

FORMAT: As described below

PROCESSOR MODE: Any

SUMMARY: C(DRn) --> C(Y),C(Y-pair); C(DRn) unchanged

If instruction bit 29 = 0
then C(DRn) --> C(Y-pair) in 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.
Faults: If DRn does not have store permission. (bit 18 for T = 8,9,11; bit 22 for all other types) an SCL2 fault occurs.
If DRm page is not housekeeping, an SCL1 fault occurs.
If DRm segment or page does not have write permission, an SCL2 fault occurs.

If instruction bit 29 = 1
and DRm descriptor type T = 0,2,4,6
then C(DRn) --> C(Y-pair) of an operand segment.
Note: DRn store permission is not required.
Faults: If processor in Master or Slave mode and DRm page is housekeeping, an SCL1 fault occurs.
If DRm segment or page does not have write permission, an SCL2 fault occurs.

If instruction bit 29 = 1
and DRm descriptor type T = 5 or 7-15
then an IPR fault occurs.

ILLEGAL ADDRESS MODIFICATIONS: ****DPS 8: DU, DL, IR, RI, IT cause IPR fault.****

****DPS 88: If descriptor of operand segment has type T=1 or 3, then DU, DL, IR, RI, IT cause IPR fault. If descriptor of operand segment has type T=0, 2, 4, 6, then DU, DL, CI, SC, SCR cause IPR fault.****

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. This set of eight instructions is used to store the contents of a descriptor register (DRn) in the even and odd memory locations Y and Y+1, in either a descriptor or operand segment.

2. If the descriptor register (DRn) is being stored in a descriptor segment the store flag (of DRn) must be on.

3. When storing a descriptor register into an operand segment the store flag is not examined by hardware.

4. Illegal address modifications and illegal repeats RPT, RPD, RPL cause an IPR fault. An IPR fault is also generated if DRm contains a type T = 5 or 7-15 descriptor.

STDSA	Store Data Stack Address Register	150 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: ****DPS 8: C(DSAR) --> $C(Y)_{0-16}$

0 - - 0 --> $C(Y)_{17-35}$ ****

****DPS 88:

C(DSAR) --> $C(Y)_{0-14}$

0 - - 0 --> $C(Y)_{15-35}$ ****

C(DSAR) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, RPL cause an IPR fault.
2. ****DPS 8: If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.****

   ****DPS 88: If the processor is not in the Privileged Master mode, the execution of this instruction causes an IPR fault.****

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 7-1)

PROCESSOR MODE: 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. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, RPL cause an IPR fault.

2. ****DPS 8: If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.****

   ****DPS 88: If the processor is not in the Privileged Master mode, the execution of this instruction causes an IPR fault.****

EXAMPLES:

```
1      8       16               32
------------------------------------------------------------------
       INHIB   OFF
SNDPR  NULL
       LDP     P4,SD.RMS,DL
       RSW     2                read processor number
       EAX7    0,AL
       ANX7    3,DU             extract processor number
       STA     RSWA2,7          save RSW2
       RSW     1
       STA     RSWA1,7          save RSW1
       LDP     P0,SD.HDP,DL
       XEC     CACHS,7          save cache control bits
       LDX3    POINT,7
       STSS    SREGS,3          store SSR
       STDSD   SREGS+2,3        store DSDR
       STWS    SREGS+4,3
       STWS    SREGS+5,3
       STO     SREGS+6,3        store option register
       SPDBR   SREGS+7,3        store page table directory base register
       LDP     P.CR,SD.CR,DL
       LDP     KLS,SD.KL,DL
       LCPR    .CRLUF,02,P.CR   reset control bits to lock cache
```

STE	Store Exponent Register	456 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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: None

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modification is used.

STI	Store Indicator Register	754 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(IR) --> $C(Y)_{18-35}$; $C(Y)_{0-17}$, C(IR) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The relation between bit positions of C(Y) and 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	0
32	Hexadecimal
33-35	000

2. The ON state corresponds to a 1 bit; the OFF state to a 0 bit.

3. Bit 25 of C(Y) will contain the state of the Tally Runout indicator prior to address modification of the STI instruction (for tally operations).

4. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****DPS 8 ONLY****

STO	Store Option Register	152 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY:

C(DSCF) --> bit 18 of C(Y)

C(SSBF) --> bit 19 of C(Y)

C(CRCF) --> bit 24 of C(Y)

0 - - 0 --> remaining 33 bits of C(Y)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. This instruction allows the Data Stack Clear Flag (DSCF), the Safe Store Bypass Flag (SSBF), and the Cache Read Control Flag (CRCF) to be stored. (See the LDO instruction.)

DSCF	0 = do not clear
	1 = clear
SSBF	0 = bypass safe-store during ICLIMB
	1 = perform safe-store during ICLIMB
CRCF	0 = bypass cache
	1 = use cache

2. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, RPL cause an IPR fault.

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    safestore frame saved?
```

****

****DPS 88 ONLY****

STO	Store Option Register	152 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY:

Hex Permission Flag --> $C(Y)_0$
0 = inhibit hex; 1 = enable hex
Lockup Fault Register --> $C(Y)_{1-2}$
See note 2.
Safestore Bypass Flag --> $C(Y)_3$
0 = perform safestore; 1 = bypass
Data Stack Clear Flag --> $C(Y)_4$
0 = don´t clear; 1 = clear
Option Register bits 5-22 --> $C(Y)_{5-22}$
Control CIU --> $C(Y)_{23}$
Hyperpaging Bypass --> $C(Y)_{24}$
Processor Number --> $C(Y)_{25-26}$
CIU 0 ICR Select --> $C(Y)_{27-29}$
CIU 1 ICR Select --> $C(Y)_{30-32}$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. Bits 0-17 of the Option Register can be loaded via the LDO instruction. Bits 18-35 of the Option Register can be loaded by the following instructions, which are valid in Hyper mode only: LDHC (bits 18-32), LGCOS (bit 33), LVMS (bit 34), LMSD (bit 35).

2. The Lockup fault time intervals are:

Bits 1-2	Time Interval
00	2 ms
01	4 ms
10	8 ms
11	16 ms

The specified time interval is effective in Slave mode only. When in Privileged Master or Master mode the Lockup fault time interval is 32 milliseconds. Upon entry to, and while executing in Hyper mode, the Lockup fault timer is reset to zero. Thus the Lockup fault may not be detected until up to 64 milliseconds have elapsed.

3. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD and RPL cause an IPR fault.

****

STPn	Store Pointer n	45n (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(ARn) --> $C(Y)_{0-23}$

C(SEGIDn) --> $C(Y)_{24-35}$

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. This set of eight instructions provides the means to store the address register (ARn) and the associated segment identity register (SEGIDn) in a single memory location. The contents of the registers remain unchanged.

2. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, and RPL cause an IPR fault.

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    =O20160,DL       test null descriptor
       TZE     FANY
```

****DPS 8 ONLY****

STPDW	Store PTWAM Directory Word	155 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: Word n of C(PTWAM Directory) --> $C(Y)_{0-29}$

0 --- 0 --> $C(Y)_{30-35}$

where:

n = bits 12-17 of Y } bits 12-15 specify row of associative memory; bits 16-17 specify column of associative memory

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The contents of Page Table Word Associative Memory (PTWAM) directory word n are stored in bits 0-29 of memory location Y and zeros are stored in bits 30-35.

   Bits 0-26 represent the combination of working space number and virtual address that is stored in the directory word for subsequent association.

   If bit 27 is 1, the row in which the directory word is stored is full.

   Bits 28-29 specify the "round robin" counter for the row in which this directory word is stored in the associative memory.

2. The PTWAM directory has 4 columns and 16 rows. The four least significant bits of the virtual address (bits 27-30) are used to select a row. Thus, the four entries in each row have the same four least significant bits.

   ****DPS 8/20 and 8/44: 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 6 least significant bits.****

3. Modifications CI, SC, SCR, DU, DL and illegal repeats RPT, RPD, RPL cause an IPR fault.

4. The STPDW instruction functions regardless of whether the PTWAM is ON or OFF.

5. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

EXAMPLE:

```
1      8      16
_________________________________
       ORG    64
AMDW   BSS    64
AMPTW  BSS    64
       .
       .
       .
       INHIB  ON
       CAMP   1
       LDX4   0,DL
AMLOOP NULL
       STPDW  AMDW,4
       STPTW  AMPTW,4
       ADLX4  1,DU
       CMPX4  64,DU
       TNC    AMLOOP
```

****

STPS	Store Parameter Stack Register	751 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(PSR) --> C(Y-pair)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The execution of this instruction causes the current contents of the parameter stack register (PSR) to be stored in even and odd memory locations Y and Y+1. The contents of the PSR remain unchanged.

2. Modifications DU, DL, CI, SC, SCR and illegal repeats RPT, RPD, RPL cause an IPR fault.

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
```

****DPS 8 ONLY****

STPTW	Store PTWAM Register	157 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: Word n of C(PTWAM Register) --> $C(Y)_{0-35}$

where:

n = bits 12-17 of Y { bits 12-15 specify row of associative memory; bits 16-17 specify column of associative memory }

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The contents of Page Table Word Associative Memory (PTWAM) directory word n are stored in memory location Y. The 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 18-27 are the software control bits and bits 30-35 are the hardware control field bits in the Page Table Word (bit 30 and 35 are stored as 1s).

2. The PTWAM directory has 4 columns and 16 rows. The four least significant bits of the virtual address (bits 27-30) are used to select a row. Thus, the four entries in each row have the same four least significant bits.

   ****DPS 8/20 and 8/44: 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 6 least significant bits.****

3. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, and RPL cause an IPR fault.

4. The STPTW instruction functions regardless of whether the PTWAM is ON or OFF.

5. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

****

STQ	Store Q-Register	756 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Q) --> C(Y); C(Q) unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL

ILLEGAL REPEATS: RPL

INDICATORS: None affected

NOTE: Address modifications DU, DL, and illegal repeat RPL cause an IPR fault.

STSS	Store Safe Store Register	753 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(SSR) --> C(Y-pair)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The contents of the safe store register (SSR) are stored in even and odd memory locations Y and Y+1. The contents of the SSR remain unchanged.
2. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, and RPL cause an IPR fault.
3. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

   ****DPS 88: If the processor is not in the Privileged Master mode, the execution of this instruction causes an IPR fault.****

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     =07777,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     =0177777,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 7-1)
PROCESSOR MODE:	Any
SUMMARY:	C(TR) --> $C(Y)_{0-26}$; 0...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 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: ****DPS 8: C(test registers 0, 1) --> C(Y-pair)****
****DPS 88: C(test register)$_{0-71}$ --> C(Y-pair)****

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The test registers are loaded by the EPAT instruction. The STTA instruction then stores the results of the EPAT into memory.

   ****DPS 88: The contents of the Test Register are undefined except when the STTA immediately follows the EPAT instruction.****

2. The contents of test registers 0 and 1 are stored in memory locations Y and Y+1. The contents of the test registers remain unchanged.

3. Modifications DU, DL, CI, SC, SCR and illegal repeats RPT, RPD, RPL cause an IPR fault.

4. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault (DPS 88: IPR fault).

****DPS 8 ONLY****

STTD	Store Test Descriptor Registers	550 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(test registers 2, 3) --> C(Y-pair)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The test registers are loaded by the EPAT instruction. The STTD instruction then stores the results of the EPAT into memory.
2. The contents of test registers 2 and 3 are stored in even and odd memory locations Y and Y+1. The contents of the test registers remain unchanged.
3. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, and RPL cause an IPR fault.
4. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

****

STWS	Store Working Space Registers	752 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(WSR) 0, 1, 2, 3 --> C(Y)$_{0-8, 9-17}$, 18-26, 27-35
if bit 17 of effective address = 0

C(WSR) 4, 5, 6, 7 --> C(Y)$_{0-8, 9-17}$, 18-26, 27-35
if bit 17 of effective address = 1

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected

NOTES:

1. The contents of working space registers (WSRs) 0, 1, 2, and 3 or the contents of WSRs 4, 5, 6, and 7 are stored in memory location Y based on the value of bit 17 of the effective address.

2. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, and RPL cause an IPR fault.

3. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

   ****DPS 88: If the processor is not in the Privileged Master mode, the execution of this instruction causes an IPR fault.****

EXAMPLE:

```
1        8         16                  32
---------------------------------------------------------------------
TODES    NULL
         STWS      WSR                 store WSR 0-3
         STWS      WSR+1               store WSR 4-7, store contents
```

STXn	Store Index Register n in Upper	74n (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: For n = 0,1,...,or 7 as determined by op code
$C(Xn) \dashrightarrow C(Y)_{0-17}$; $C(Y)_{18-35}$ unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL
RPT, RPD 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 7-1)

PROCESSOR MODE: 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 7-1)

PROCESSOR MODE: Any

SUMMARY: If Carry indicator is ON, then C(A) - C(Y) --> C(A); C(Y) unchanged

If Carry indicator is OFF, then C(A) - C(Y) - 0...1 --> 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

NOTES:

1. This instruction is identical to SBA with the exception that when the Carry indicator is OFF at the beginning of the instruction, a positive 1 is subtracted from the least significant position.

2. 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) + 1's complement of C(Y) + 0...1 --> C(A)

   If Carry indicator is OFF, then C(A) + 1's complement of C(Y) --> C(A)

   The positive 1 added when ON represents the carry from the next less significant part of the multiword subtraction.

SWCQ	Subtract with Carry from Q-Register	172 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: If Carry indicator is ON, then C(Q) - C(Y) --> C(Q); C(Y) unchanged

If Carry indicator is OFF, then C(Q) - C(Y) - 0...1 --> 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

NOTES:

1. This instruction is identical to SBQ with the exception that when the Carry indicator is OFF at the beginning of the instruction, a positive 1 is subtracted from the least significant position.

2. 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) + 1's complement of C(Y) + 0...1 --> C(Q)

   If Carry indicator is OFF, then C(Q) + 1's complement of C(Y) --> C(Q)

   The positive 1 added when ON represents the carry from the next less significant part of the multiword subtraction.

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     =O733777,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(X)	Subtract Word Displacement from Address Register	527 (1)

FORMAT: Special arithmetic instruction format (see Figure 7-3)

CODING FORMAT:

```
1      8      16
_____________________________________
       SWD(X) word displacement,R,AR
```

PROCESSOR MODE: Any

SUMMARY: If bit 29 = 1: $C(AR\underline{n})_{0-17}$ - (y+C(DR))

-- > $C(AR\underline{n})_{0-17}$

If SWDX, bit 29 = 0: -(y+C(DR)) --> $C(AR\underline{n})_{0-17}$

In either case, zeros --> $C(AR\underline{n})_{18-23}$

Description is the same as for AWD except that the sum of the y field and the contents of the register specified by the DR field are subtracted from the AR. When the mnemonic is coded with an X (SWDX), bit 29 is forced to zero.

ILLEGAL ADDRESS MODIFICATIONS: All except N, AU, QU, AL, QL, and index registers

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
____________________________________________________________
       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 7-1)

PROCESSOR MODE: Any

SUMMARY: For N=0,1,..., or 7 as determined by op code
C(Xn) --> $C(Y)_{18-35}$; $C(Y)_{0-17}$ unchanged

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPL
RPT, RPD of SXL0

INDICATORS: None affected

NOTE: An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****DPS 88 ONLY****

SYNC	Gate Synchronize	014 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: This instruction causes the processor to wait until all of its outstanding memory hierarchy commands have been completed before executing the next instruction.

ILLEGAL ADDRESS MODIFICATIONS: None. Address modifications have no effect on the operation but are performed by the hardware.

ILLEGAL REPEATS: RPT, RPD, RPL cause an IPR fault.

INDICATORS: None affected.

NOTES:

1. LDAC, LDQC, SZNC, STAC, and STACQ are the only instructions that can be used for the indivisible test-and-set operations which are required for setting and releasing locks, or for closing and opening gates.

2. Since execution of LDAC, LDQC, SZNC, STAC, and STACQ depends on the previous C(Y), the CPU will obtain ownership of the 8-word block containing C(Y) prior to using C(Y) to execute the instruction. Obtaining ownership of the 8-word block means that the requesting CPU, and the Memory Hierarchy Control of the CIU, will ensure that a valid copy of the block is obtained, and that the block is cleared from the cache of all other CPUs before the instruction is executed. After obtaining ownership of the block, the CPU completes execution of the instruction to set or release the lock without permitting the block to be siphoned to another CPU. Thus the block is isolated in a time window where it can be accessed and modified only by the CPU executing the instruction which sets or releases the lock.

3. To ensure that a lock does not get released before the actual completion of all stores performed while the lock was set, a synchronizing function is necessary. This synchronizing function is accomplished by coding a SYNC or STC2 instruction immediately before the instruction which releases the lock. If the value stored by STC2 is consistent with operating system conventions for a released lock, then the use of STC2 for synchronizing can also serve to release the lock.

SZN	Set Zero and Negative Indicators from Storage	234 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Y) --> C(Z); 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

NOTE:

Zero	Negative	Relation
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 7-1)

PROCESSOR MODE: Any

SUMMARY: C(Y) --> C(Z); 0...0 --> C(Y)

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

NOTES:

1.

Zero	Negative	Relation
0	0	Number C(Y) > 0
1	0	Number C(Y) = 0
0	1	Number C(Y) < 0

2. ****DPS 8: Loss of efficiency may occur when using this instruction, since cache memory is cleared when the instruction is executed. If cache memory clearance is not desired, the SZN instruction should be used followed by a STZ instruction, instead of SZNC. (This condition does not exist when the 8K cache memory option is installed.)****

3. ****DPS 88: LDAC, LDQC, SZNC, STAC, and STACQ are the only instructions that can be used for the indivisible test-and-set operations which are required for setting and releasing locks, or for closing and opening gates.

   Since execution of LDAC, LDQC, SZNC, STAC, and STACQ depends on the previous C(Y), the processor will obtain ownership of the 8-word block containing C(Y) prior to using C(Y) to execute the instruction. Obtaining ownership of the 8-word block means that the requesting processor, and the Memory Hierarchy Control of the CIU, will ensure that a valid copy of the block is obtained, and that the block is cleared from the cache of all other processors before the instruction is executed. After obtaining ownership of the block, the processor completes execution of the instruction to set or release the lock without permitting the block to be siphoned to another processor. Thus the block is isolated in a time window where it can be accessed and modified only by the processor executing the instruction which sets or releases the lock.

To ensure that a lock does not get released before the actual completion of all stores performed while the lock was set, a synchronizing function is necessary. This synchronizing function is accomplished by coding a SYNC or STC2 instruction immediately before the instruction which releases the lock. If the value stored by STC2 is consistent with operating system conventions for a released lock, then the use of STC2 for synchronizing can also serve to release the lock.****

4. ****DPS 8/50, 8/52, 8/62 and 8/70: This instruction bypasses the cache memory and clears the cache block (4 words) containing address Y.****

5. The SZNC instruction should only be used for gating purposes. It should not be used as a substitute for a sequence of SZN, TZE, STZ because of the performance penalty that is introduced.

6. An Illegal Procedure fault occurs if illegal address modification is used.

SZTL	Set Zero and Truncation Indicators with Bit Strings Left	064 (1)

FORMAT:

						Op Code		
00	01 04	05 08	09	10	11 17	18 27	28	29 35
F	0000	BOLR	T	0	MF2	064(1)	I	MF1

00 02	03 17	18 19	20 23	24	32 35
Y1		C1	B1	N1	
a1	Y2			0-------------------0	R1

00 02	03 17	18 19	20 23	24	32 35
Y2		C2	B2	N2	
a2	Y2			0-------------------0	R2

CODING FORMAT:

```
1      8         16
______________________________________
       SZTL      (MF1),(MF2),BOLR,F,T
       BDSC      LOCSYM,N,C,B,AM
       BDSC      LOCSYM,N,C,B,AM
```

PROCESSOR MODE: Any

SUMMARY: C(string 1) : (BOLR) : C(string 2)

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 greater than L2, then ON; otherwise, OFF

****DPS 88: The Zero and Truncation indicators are affected even if L1 and/or L2 = 0.****

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modifications are used for MF1 or MF2 or if illegal repeats are used.

2. L2=0 does not necessarily mean that the instruction functions as a no-op, as the Truncation indicator may be 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:

<table>
<tr><td>00</td><td>01 04</td><td>05 08</td><td>09</td><td>10</td><td>11 17</td><td>18 Op Code 27</td><td>28</td><td>29 35</td></tr>
<tr><td>F</td><td>0000</td><td>BOLR</td><td>T</td><td>0</td><td>MF2</td><td>065(1)</td><td>I</td><td>MF1</td></tr>
</table>

<table>
<tr><td>00 02</td><td>03 17</td><td>18 19</td><td>20 23</td><td>24 32</td><td>35</td></tr>
<tr><td colspan="2">Y1</td><td rowspan="2">C1</td><td rowspan="2">B1</td><td colspan="2">N1</td></tr>
<tr><td>a1</td><td>Y2</td><td>0-------------------0</td><td>R1</td></tr>
</table>

<table>
<tr><td>00 02</td><td>03 17</td><td>18 19</td><td>20 23</td><td>24 32</td><td>35</td></tr>
<tr><td colspan="2">Y2</td><td rowspan="2">C2</td><td rowspan="2">B2</td><td colspan="2">N2</td></tr>
<tr><td>a2</td><td>Y2</td><td>0-------------------0</td><td>R2</td></tr>
</table>

CODING FORMAT:

```
1      8        16
       SZTR     (MF1),(MF2),BOLR,F,T
       BDSC     LOCSYM,N,C,B,AM
       BDSC     LOCSYM,N,C,B,AM
```

PROCESSOR MODE: Any

SUMMARY: C(string 1) : (BOLR) : C(string 2)

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

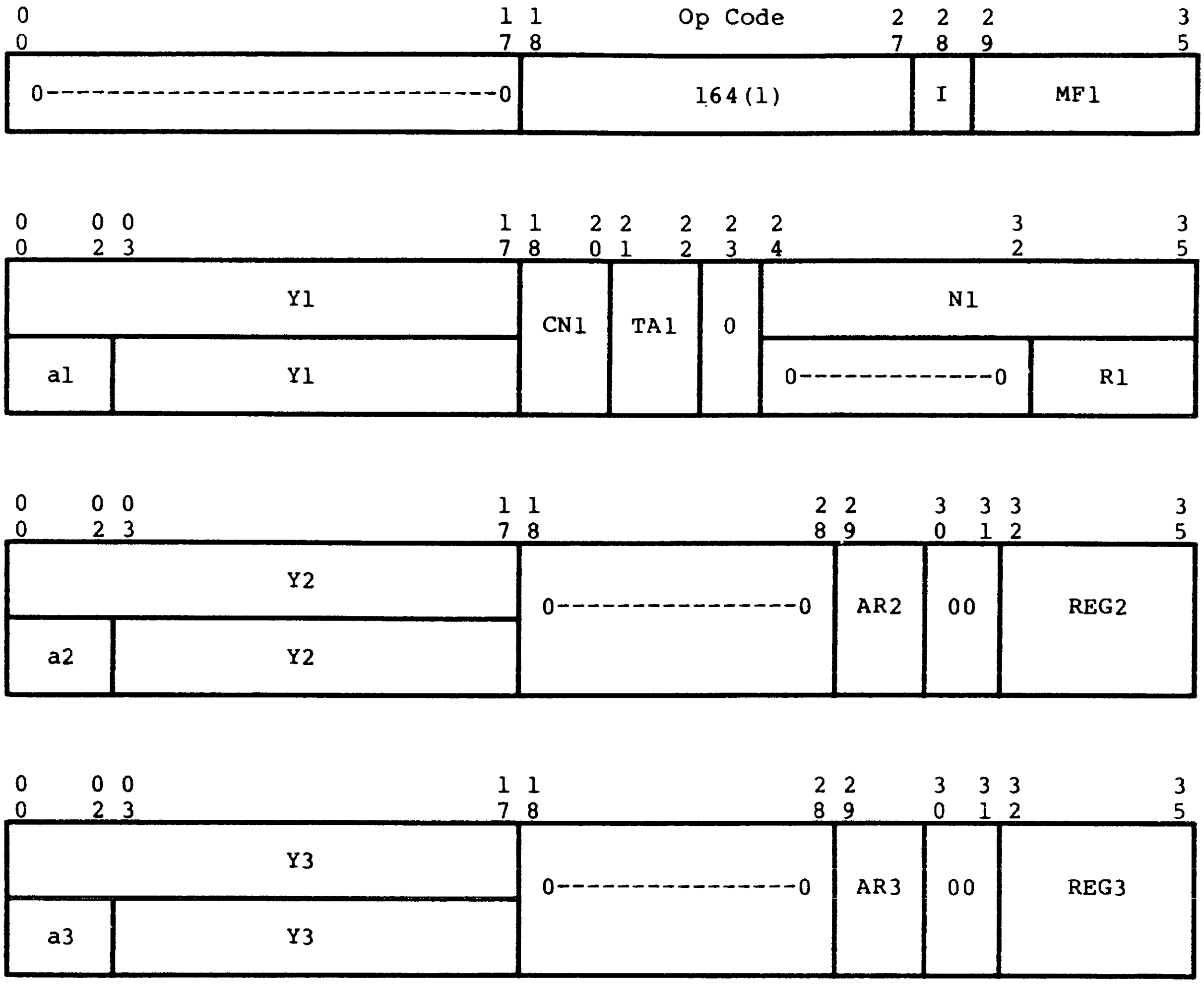
NOTE: Notes for SZTR are the same as for SZTL.

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
```

PROCESSOR MODE: Any

SUMMARY:

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.

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. ****DPS 8/20 and 8/44: When pre-paging, the hardware assumes that the length of the translate table corresponds to the data type identified by TA1 as follows:

TA1	Table Length	
4-bit	4 words	
6-bit	16 words	
9-bit	128 words	****

4. 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,b̸1234567890#        2001020304050607l0110013 (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 Field 1 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,b̸b̸1234;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 table characters
          OCT        000000000000,000000000000 (060-067)
          OCT        000000000040               (070-073)
          USE                              Result - nonzero found
```

NOTE: The highest possible value in Field 1 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
```

PROCESSOR MODE: Any

SUMMARY: 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 FLD1 IS OCTAL 17
       USE                     Result - no illegal character found
```

TEO	Transfer on Exponent Overflow	614 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1         8         16
          TEO       LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

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))

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. 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.

6. 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 7-1)

CODING FORMAT:

```
1      8        16
________________________
       TEU      LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: 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))

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. 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.
6. 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 7-1)

CODING FORMAT:

```
1      8      16
______________________
       TMI    LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

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))

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. 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.
6. 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 7-1)

CODING FORMAT:

```
1      8      16
________________________________
       TMOZ   LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: 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))

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. 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.

6. 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 7-1)

CODING FORMAT:

```
1      8      16
____________________
       TNC    LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

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))

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. 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.

6. 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 7-1)

CODING FORMAT:

```
1      8        16
______________________________
       TNZ      LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

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))

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. 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.

6. 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 7-1)

CODING FORMAT:

```
1      8       16
______________________________
       TOV     LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

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))

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. 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.

6. 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 7-1)

CODING FORMAT:

```
1      8      16
______________________
       TPL    LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

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))

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. 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.
6. 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 7-1)

CODING FORMAT:

```
1      8      16
______________________________
       TPNZ   LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: 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))

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. 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.
6. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

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 7-1)

CODING FORMAT:

```
1      8       16
       TRA     LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

```
Y --> C(IC)
If instruction bit 29=1 then
   $n = Y_{0-2}$
   C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))
```

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. 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.

6. 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 7-1)

CODING FORMAT:

```
1      8      16
       TRC    LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

```
If Carry indicator ON, then Y --> C(IC)
If Carry indicator ON and instruction bit 29=1 then
   n = Y0-2
   C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))
```

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. 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.

6. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TRTF	Transfer on Truncation Indicator OFF	601 (1)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1      8         16
________________________________
       TRTF      LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY: 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))

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. 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.
6. 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                    truncation indicator ON

*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 7-1)

CODING FORMAT:

```
1      8      16
______________________________
       TRTN   LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

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))

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. 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.

6. 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 7-1)

CODING FORMAT:

```
1      8      16
______________________
       TSS    LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

```
Y --> C(IC)
If instruction bit 29=1 then
   n = Y0-2
   C(DRn) --> C(ISR); C(SEGIDn) --> C(SEGID(IS))
```

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. 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.

6. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

7. 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 safestore frame is:

   - If fault is IPR or Fault Tag fault, caused by the tag field in the instruction or indirect word, then IR28 = 1.

   - If fault is STR or BND, caused by attempt to access an indirect word, then IR28 = 1.

   - If fault is STR or BND, caused by attempt to access the target location then

     ****DPS 8/20 and 8/44 IR28 = 0. ****

     ****DPS 88, DPS 8/50, 8/52, 8/62, and 8/70: IR28 = 1. ****

TSXn	Transfer and Set Index Register n	70n (0)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1      8         16
       TSXn      LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

```
For n = 0,1,..., or 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))
```

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. 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.

6. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****DPS 88 ONLY****

TTES	Transfer Table Entry Store	531 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1      8      16
______________________
       TTES   LOCSYM,R,AR
```

PROCESSOR MODE: Privileged Master Mode

SUMMARY: C(Transfer Table)$_{0-71}$ --> C(Y=pair)

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The use of this instruction in other than Privileged Master mode causes an IPR fault.

2. The Transfer Table is a LIFO (Last-in, First-out) queue containing information identifying the last 16 transfers that were taken. Each entry in the Table has the following format:

   Bits 0-8 Effective working space number (EWSN) of target segment.

   Bits 9-35 Word level virtual address (bits 14-40 of virtual address). If the virtual address refers to a fragmented page table, the five most significant bits of the 16-bit page key are lost.

   Bits 36-53 Old instruction counter.

   Bits 54-71 New instruction counter.

   When a successful transfer is executed, the appropriate entry is "pushed" onto the top of the table.

3. TTES causes the most recent entry to be "popped" off the stack and placed in memory at the location specified using the normal address development.

4. The TTTL instruction can be used to lock the table. When the table is locked, no further entries are made until the table is unlocked using the TTTU instruction.

5. The TTEZ instruction causes the hardware to push a zero entry into the table to allow software to mark the boundary between different sections of code. Subsequently, when the table is stored into memory, the zero entry can be used to distinguish between different sections of code.

6. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

****DPS 88 ONLY****

TTEZ	Transfer Table Entry Zero	524 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1      8      16
       TTEZ   LOCSYM,R,AR
```

PROCESSOR MODE: Privileged Master Mode

SUMMARY: Load Zero Entry in Transfer Table

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The use of this instruction in other than Privileged Master mode causes an IPR fault.

2. The Transfer Table is a LIFO (Last-in, First-out) queue containing information identifying the last 16 transfers that were taken. Each entry in the Table is a double-word quantity. See description of TTES for format.

   When a successful transfer is executed, the appropriate entry is "pushed" onto the top of the table.

3. TTES causes the most recent entry to be "popped" off the stack and placed in memory at the location specified using the normal address development.

4. The TTTL instruction can be used to lock the table. When the table is locked, no further entries are made until the table is unlocked using the TTTU instruction.

5. The TTEZ instruction causes the hardware to push a zero entry into the table to allow software to mark the boundary between different sections of code. Subsequently, when the table is stored into memory, the zero entry can be used to distinguish between different sections of code.

6. 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 7-1)

CODING FORMAT:

```
1      8        16
______________________
       TTF      LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

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))

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. 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.

6. 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 7-1)

CODING FORMAT: The TTN instruction is coded as follows:

```
1      8       16
______________________________
       TTN     LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

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))

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. 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.

6. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

```
1      8        16               32
_______________________________________________________________________
       TCT                       test character and translate
       ADSC9    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,ƀ1234567890#
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
```

****DPS 88 ONLY****

TTTL	Transfer Trace Table Lock	522 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1      8      16
______________________
       TTTL   LOCSYM,R,AR
```

PROCESSOR MODE: Privileged Master Mode

SUMMARY: Lock the Transfer Table

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The use of this instruction in other than Privileged Master mode causes an IPR fault.

2. The Transfer Table is a LIFO (Last-in, First-out) queue containing information identifying the last 16 transfers that were taken. Each entry in the Table is a double-word quantity. See description of TTES for format.

   When a successful transfer is executed, the appropriate entry is "pushed" onto the top of the table.

3. TTES causes the most recent entry to be "popped" off the stack and placed in memory at the location specified using the normal address development.

4. The TTTL instruction can be used to lock the table. When the table is locked, no further entries are made until the table is unlocked using the TTTU instruction.

5. The TTEZ instruction causes the hardware to push a zero entry into the table to allow software to mark the boundary between different sections of code. Subsequently, when the table is stored into memory, the zero entry can be used to distinguish between different sections of code.

6. Address modifications have no effect on the operation, but are performed by the hardware.

7. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

****

****DPS 88 ONLY****

TTTU	Transfer Trace Table Unlock	523 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1      8      16
       TTTU   LOCSYM,R,AR
```

PROCESSOR MODE: Privileged Master Mode

SUMMARY: Unlock the Transfer Table

ILLEGAL ADDRESS MODIFICATIONS: DU, DL, CI, SC, SCR

ILLEGAL REPEATS: RPT, RPD, RPL

INDICATORS: None affected.

NOTES:

1. The use of this instruction in other than Privileged Master mode causes an IPR fault.

2. The Transfer Table is a LIFO (Last-in, First-out) queue containing information identifying the last 16 transfers that were taken. Each entry in the Table is a double-word quantity. See description of TTES for format.

   When a successful transfer is executed, the appropriate entry is "pushed" onto the top of the table.

3. TTES causes the most recent entry to be "popped" off the stack and placed in memory at the location specified using the normal address development.

4. The TTTL instruction can be used to lock the table. When the table is locked, no further entries are made until the table is unlocked using the TTTU instruction.

5. The TTEZ instruction causes the hardware to push a zero entry into the table to allow software to mark the boundary between different sections of code. Subsequently, when the table is stored into memory, the zero entry can be used to distinguish between different sections of code.

6. Address modifications have no effect on the operation, but are performed by the hardware.

7. An Illegal Procedure fault occurs if illegal address modification or illegal repeats are used.

****

TZE	Transfer on Zero	600 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

CODING FORMAT:

```
1      8      16
------------------------
       TZE    LOCSYM,R,AR
```

PROCESSOR MODE: Any

SUMMARY:

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))

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. 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.

6. 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 7-1)

PROCESSOR MODE: Any

SUMMARY: [C(EAQ) + C(Y)] not normalized --> C(EAQ)

ILLEGAL ADDRESS MODIFICATIONS: CI, SC, SCR cause an IPR to occur.

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 greater than +127, then ON

Exponent Underflow - If exponent is less than -128, then ON

Carry - If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise OFF

NOTE: 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:

****DPS 8: Mode register, bit 33****

****DPS 88: Option register, bit 0****

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 7-1)
PROCESSOR MODE:	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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -128, then ON

NOTES:

1. This multiplication is executed like the FMP instruction with the exception that the final normalization is performed only in the case of both factor mantissas being = - 1.00...0.

2. The definition of normalization is located under the description of the FNO instruction.

3. When indicator bit 32=1 and the Hex Permission Flag = 1 the floating point alignment and normalization is hexadecimal. Otherwise, the floating point alignment and normalization binary. The Hex Permission Flag is:

   ****DPS 8: Mode register, bit 33****

   ****DPS 88: Option register, bit 0****

4. An Illegal Procedure fault occurs if illegal address modification is used.

UFS	Unnormalized Floating Subtract	535 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: 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 greater than +127, then ON
Exponent Underflow	- If exponent is less than -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:

   ****DPS 8: Mode register, bit 33****

   ****DPS 88: Option register, bit 0****

2. An Illegal Procedure fault occurs if illegal address modification is used.

XEC	Execute	716 (0)

FORMAT: Single-word instruction format (see Figure 7-1)

PROCESSOR MODE: Any

SUMMARY: Obtain and execute the instruction stored at memory location Y

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. If the instruction obtained from location Y is not a Repeat Double (RPD) instruction, and is not a multiword instruction, the next instruction to be executed is obtained from C(IC) + 1. This is the instruction contained in the memory location immediately following the location containing the XEC instruction, unless the contents of the instruction counter have been changed by the execution of the instruction obtained from memory location Y.

2. To Execute (XEC) a Repeat Double (RPD) instruction, the XEC instruction must be in an odd location. The instructions repeated are those that immediately follow the XEC instruction. The next instruction to be executed is obtained from C(IC) + 3.

3. 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.

4. 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.

5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

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 7-1)

PROCESSOR MODE: Any

SUMMARY: Obtain and execute the two instructions stored at the memory Y-pair locations (must be even and next odd location).

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. 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.

2. 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.

3. If the instruction obtained from the odd location of Y-pair is not a Repeat Double (RPO), the next instruction to be executed is obtained from C(IC) + 1. This is the instruction contained in the memory location immediately following the location containing the XED instruction unless the contents of the instruction counter have been changed by the execution of the two instructions obtained from the memory location Y-pair.

4. To Execute Double (XED) a pair that has Repeat Double (RPD) as the odd instruction of the pair, XED must be located at an odd address. The instructions repeated are those that immediately follow the XED instruction. The next instruction to be executed is obtained from (CIC) + 3.

5. If RPD is specified within a sequence of XEDs, the original and all subsequent XEDs in the sequence must be in odd locations.

6. An Illegal Procedure fault occurs if an attempt is made to XED any multiword instruction.

7. 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.

8. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

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
```

MICRO OPERATIONS

A description of the 17 micro-operations (MOPs) follows. The mnemonic, name, octal value, and the function performed is given for each MOP in a format similar to that for processor instructions.

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.

CHT	Change Table	21

SUMMARY: 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	02

SUMMARY:

Bit 0 of IF (IF0) specifies the nature of the floating suppression.

Bit 1 of IF (IF1) specifies if blank when zero option is used.

For IF0 = 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 IF0 = 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 IF1 = 1 (blank when zero): the BZ flag is set ON.

For IF1 = 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	14

SUMMARY: 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	11

SUMMARY: 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) > 8, an Illegal Procedure fault occurs.

INSB	Insert Blank on Suppression	10

SUMMARY:

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) > 8, an Illegal Procedure fault occurs.

INSM	Insert Table Entry One Multiple	01

SUMMARY: 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	12

SUMMARY:

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 is not equal to 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) > 8, an Illegal Procedure fault occurs.

INSP	Insert On Positive	13

SUMMARY: Same as INSN except that the responses for the SN values are reversed.

FLAGS: None affected

NOTE: If C(IF) > 8, an Illegal Procedure fault occurs.

LTE	Load Table Entry	20

SUMMARY: 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) > 8, an Illegal Procedure fault occurs.

MFLC	Move with Floating Currency Symbol Insertion	07

SUMMARY:

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 Illegal Procedure fault may be avoided by ensuring that the Z and BZ flags are ON.

MFLS	Move with Floating Sign Insertion	06

SUMMARY:

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	17

SUMMARY:

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	16

SUMMARY:

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 (sign already set), IF characters are moved into the receiving string (MSES equivalent to MVC).

****DPS 88: For MVNEX (sign already set), if characters are moved into the receiving string (MSES equivalent to MVC).

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	15

SUMMARY:

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	05

SUMMARY: 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	04

SUMMARY:

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

SES	Set End Suppression	03

SUMMARY:

Bit 0 of IF (IF0) specifies the setting of the ES switch.

Bit 1 of IF (IF1) specifies if blank when zero option is used.

If IF0 = 0, the ES flag is set OFF.

If IF0 = 1, the ES flag is set ON.

If IF1 = 1, the BZ flag is set ON.

If IF1 = 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 7-1. A dash (----) indicates an unassigned code. All unassigned codes cause an Illegal Procedure fault.

Table 7-2. Micro Operation Code Assignment Map

	0	1	2	3	4	5	6	7
00	----	INSM	ENF	SES	MVZB	MVZA	MFLS	MFLC
10	INSB	INSA	INSN	INSP	IGN	MVC	MSES	MORS
20	LTE	CHT	----	----	----	----	----	----
30	----	----	----	----	----	----	----	----

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 Illegal Procedure fault) if an attempt is made to move from an exhausted sending string or to use an exhausted MOP string.

Micro Operation Example

```
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),1Hb̸,(MVZB,7),(SES,8)
         MICROP    (INSB),1H.,(MVC,3),(INSN)
         MICROP    1H-,(LTE,1),1Hb̸,(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
```

SECTION VIII

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. Since this version 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 (DPS 8 Only) vector locations (in real memory) containing the entry descriptors are as follows:

Location (octal)	Vector
30-31	Interrupt
32-33	Fault
34-35	Special systems entry
40-41	Backup fault (DPS 8 only)

FAULT PROCEDURE

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, provided the Stack Control Register (SCR) defines the frame size as 64 words.

The second word of the "wired-in" ICLIMB instruction is assumed as described for interrupts. (See "Interrupt Procedure" later in this section.)

**** DPS 8: 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 DIS 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. ****

**** DPS 88: If an entry descriptor is not found in the fixed fault vector location or if another fault should occur while the processor is attempting to CLIMB to the fault handler, SSF is notified and the processor halts. ****

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 (DPS 88: seven) groups to establish priority for the recognition of a specific fault when two or more faults occur at the same time in different groups. See Tables 4-2 (DPS 88: Table 4-4) and 8-1 (DPS 88: 8-2).

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 8-1 lists the faults in order of the fault code, and shows the priority assigned by the processor, and the priority group number.

Faults in Groups I and II cause the operations in the processor to terminate unconditionally.

**** DPS 8: Faults in Groups III and IV cause the operations in the processor to terminate when the operation currently being executed is completed.

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 8-1. Processor Faults By Fault Code (DPS 8)

Fault Code	Octal Code	Fault Name	Priority	Group Priority
00000	00	Shutdown (SDF)	23	V
00001	02	Store memory (STR)	9	IV
00010	04	Master mode entry (MME)	10	IV
00011	06	Fault tag (FTAG)	13	IV
00100	10	Timer runout (TROF)	22	V
00101	12	Command (FCMD)	8	IV
00110	14	Derail (DRL)	11	IV
00111	16	Lockup (LUF)	4	II
01000	20	Connect (CON)	21	V
01001	22	Parity (FPAR)	7	IV
01010	24	Illegal procedure (IPR)	12	IV
01011	26	Operation not completed (FONC)	3	II
01100	30	Startup (SUF)	1	I
01101	32	Overflow (FOVF)	6	III
01110	34	Divide check (FDIV)	5	III
01111	36	Execute (EXF)	2	I
10000	40	Security fault, class 1 (SCL1)	14	IV
10001	42	Dynamic linking (DYNLF)	15	IV
10010	44	Missing segment (MSE)	16	IV
10011	46	Missing working space (MWS)	17	IV
10100	50	Missing page (MPG)	18	IV
10101	52	Security fault, class 2 (SCL2)	19	IV
(See NOTE)	--	Safe store stack fault (SSSF)	20	IV

NOTE: The safe store stack overflow fault has no fault code since it may occur with any other fault. The fault code is contained in bits 12-16 of safe store stack frame word 5. If a safe store stack fault occurs, bit 10 of word 5 is set in the safe store frame. Refer to Figure 8-4 for a description of the safe store stack.

Table 8-2. Processor Faults By Fault Code (DPS 88)

Fault Code	Fault Mnemonic	Fault Name	Priority	Group
00000	SDF	Shutdown	29	VII
00001	BND	Bound	10	IV
00010	MME	Master Mode Entry	11	V
00011	FTAG	Fault Tag	14	V
00100	TRO	Timer Runout	28	VII
00101	CMD	Command	9	IV
00110	DRL	Derail	12	V
00111	LUF	Lockup	5	II
01000	CON	Connect Received	27	VII
01001	MEMSYS	Memory System**	6	II
01010	IPR	Illegal Procedure	13	V
01011	ONC	Operation Not Complete	4	II
01100	SUF	Startup	1	I
01101	OFL	Overflow	8	III
01110	DIV	Divide Check	7	III
01111	EXF	Execute	2	I
10000	SCL1	Security Fault, Class 1	17	V
10001	DYNLF	Dynamic Linking	18	V
10010	MSE	Missing Segment	19	VI
10011	MWS	Missing Working Space	20	V
10100	MPG	Missing Page	21	VI
10101	SCL2	Security Fault, Class 2	22	VI
10110	--	Undefined	24	VI
10111	SSSF	Safestore Stack Fault	23	VI
11000	--	Undefined	15	V
11001	--	Undefined	16	V
11010	--	Undefined	30	VII
11011	--	Undefined	31	VII
11100	DIS*	DIS Hypermode Entry	25	VII
11101	CIOC*	CIOC Hypermode Entry	26	VII
11110	HTRO*	Hypertimer Runout	32	VII
11111	--	Undefined	3	II

* Hyperclimb only. If there is no hyperswitcher, these are classified as "undefined".

** Parity fault has been renamed as Memory System fault.

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:

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 will terminate upon recognition of F, and the operation will not be 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 (DPS 88: Central Interface Unit) indicating that some processor in the system executed a CIOC instruction directed to this processor.

5. Illegal Procedure (IPR)

   An illegal operation code, an illegal address (for instructions using the address field to specify a register), an illegal modifier (or modifier sequence), or an illegal instruction sequence was used.

   The attempted execution of an illegal instruction sequence or modification will generate an IPR fault. The attempted execution of a legal Master mode instruction in the Slave mode will cause a Command (FCMD) fault (DPS 88: IPR 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 if an attempt is made to repeat any multiword instruction with the use of RPT, RPD, or RPL instructions or to XEC or XED 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 Illegal Procedure fault is generated for each of the following conditions when the virtual memory is installed and enabled (DPS 8: each condition sets bit 0 of the fault register).

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 LDDn 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 (option register bit 19; bit 3 in DPS 88) 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.

An IPR fault occurs during the CLIMB instruction if the descriptor referenced by the S and D fields is not T = 0, 1, 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 an LDSS instruction if the descriptor to be loaded into the safe store 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).

**** DPS 88: Has a base that is not modulo-8 words (bits 31-35 are not equal to 00000). ****

An IPR fault occurs during the 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).

**** DPS 88: Has a base that is not modulo-8 words (bits 31-35 are not equal to 00000.) ****

c. Has a bound that is not 7 modulo-8 bytes (bits 17-19 are not equal to 111).

**** DPS 88: Has a bound that is not 31 modulo-32 bytes (bits 27-31 not equal to 11111). ****

d. Has flag bit 22 (store) = 1.

An IPR fault occurs during the LDEA<u>n</u> instruction if the descriptor to be loaded does not have T = 4 or 6 (super descriptor).

An IPR fault occurs during the LDAS and 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, TSX<u>n</u>), 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 registers 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 three major categories of program generated faults, each of which has several subcategories:

1. Arithmetic Faults

   a. Overflow (FOVF). 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 will 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 processor executes the Overflow fault. Note that the overflow mask bit (indicator register) will not affect automatic fault on truncation.

   b. Divide Check (FDIV). 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 instructions - If the mantissa of the divisor equals zero.

      4) DV2D, DV3D instructions - If the divisor is equal to 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 (TROF). This fault is generated when the timer count reaches zero and cycles to minus 1.

   **** DPS 8: 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; ****

   **** DPS 88: If the processor is in the Privileged Master mode or Master mode the recognition of this fault will be delayed until the processor returns to Slave mode; ****

   This delay does not inhibit the counting in the timer register. (See DIS instruction for an 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 *, the continuous use of the inhibit bit, or repeat mode loops exceeding the lockup time.

Master mode lockup time is set at 32 milliseconds and Slave mode lockup time is specified by the lockup fault register, which can be loaded in Privileged Master mode using the LCPR (DPS 88: LDO) instruction with the register specified in the tag field (TAG = 02).

c. Operation Not Completed (FONC). 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 FONC fault can be generated by disabling the SCU ports via program control while the program is being executed.

   NOTE: A FONC fault can also be generated by hardware malfunction.

3. Command Faults

   a. **** DPS 8: 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 attempted access to working space zero when the processor is not in the Privileged Master mode.

   c. **** DPS 8: Answer an XIP (interrupt present) or to execute RSCR, SSCR, RMCM, or SMCM instructions with respect to the interrupt mask register from a system controller port with no interrupt mask register assigned. ****

   d. Use a vector in Master mode or Slave mode with a LDD$\underline{n}$ or LDP$\underline{n}$ instruction that specifies S = 00 and D = 1761, 1763, or 1764 (octal) (type change, DSDR or SSR).

   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. Store Memory (STR) (DPS 88: 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 select a processor port not enabled.

   d. An attempt is made to access a "not ready" memory.

e. An attempt is made to use absolute addressing or dense paging with a relative virtual address ≥ 2**24 words (DPS 88: 2**26 words).

f. 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.

g. An attempt is made to access the contents of a type T = 0, 1, 2, or 3 segment and:

a. The upper or lower bound is exceeded.

b. The addition of the base and the effective address fields produces a carry.

h. An attempt is made to access the contents of a type T = 4 or 6 segment and:

a. The bound field is exceeded.

b. The addition of either the location and effective address fields or the location, effective address, and base fields produces a carry.

i. The E field is 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)

j. Boundary violations occur in the shrink operation as indicated in the descriptions for the LDDn instruction, or when preparing descriptors during a CLIMB instruction.

k. 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.

l. **** DPS 88: When a T = 6 descriptor is loaded into a DRn and the base or bound calculation for forming the standard descriptor produces a carry or borrow repectively. ****

Virtual Memory-Generated Faults

Virtual memory-generated faults are:

1. Security Fault, Class 1 (SCL1)

A Security Fault, Class 1, occurs:

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 8-3.

Table 8-3. Processor Modes

Bit Status	Privileged Master Mode	Master Mode		Slave Mode	Illegal Combination[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 an 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 - Safestore and push parameters on the argument stack.

2) SDRn instruction - Push to the argument stack.

3) STDn 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, 4, or 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 systems entry (PMME) is invoked, the fault detection is not overridden.

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 = 01 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.
SDRn	- Write to argument segment.

2. Dynamic Linking Fault (DYNLF)

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 (MSE)

A Missing Segment fault is generated when an attempt is made to access memory using a segment descriptor that has flag bit 28 equal to 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 (DPS 88:bit 23), page 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 or, in the case of a fragmented page table, no key match is found (DPS 8/70, 8/50, 8/52, 8/62: or multiple key matches are found).

**** DPS 88: Word 1, bit 35 of the safe store frame is defined as the Demand Paging Recovery Flag (DPRF). DPRF has a defined value only when a Missing Page fault occurs. The value of DPRF is undefined for all other faults.

When a Missing Page fault occurs the processor stores an appropriate value in DPRF 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 ****

**** DPS 8/20, 8/44: 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.

**** DPS 88: The only reasons for which the processor sets DPRF = 0 (not recoverable) in the safe store frame are:

a. Occurrence of a Missing Page fault while executing an RPT, RPD, or RPL instruction.

b. Occurrence of a Missing Page fault while executing one of the pair of instructions pointed to by an XED instruction. Note that if a Missing Page fault occurs while fetching the pair of instructions pointed to by an XED instruction, the hardware sets DPRF = 1 in the safe store frame.

c. Occurrence of a Missing Page fault during indirect and tally operations in which the number of indirect tally words updated (ITC) is > 7. See word 0, bits 14-17. ****

**** DPS 8/20, 8/44: The only reasons for which the processor sets FRTRY = 1 (not recoverable) in the safe store frame are:

a. Occurrence of a Missing Page fault while executing an RPT, RPD, or RPL instruction.

b. Occurrence of a Missing Page fault while executing an instruction pointed to by an XEC or XED instruction.

c. Occurrence of a Missing Page fault during an indirect and tally operation. ****

**** DPS 88, DPS 8/20, 8/44: Before the EIS numeric, MVE, DTB, or BTD instructions will execute, all pages containing parts of the operands and pages in which the results will 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. ****

**** DPS 88: On an indirect and tally chain the same indirect word must not be referenced more than once. On a recoverable tally chain (ITC ≤ 7 when Missing Page fault occurs), the hardware will "rewalk" the chain, requiring that all pages in the chain remain in memory before the operand can be reached. ****

**** DPS 8/70, 8/50, 8/52, 8/62: There is no hardware retry bit for the processor. Software can analyze the faulting instruction to determine whether recovery is possible. If it finds any of the three conditions listed above for the DPS 8/20 or 8/44 processor, or if it finds an EIS instruction with overlapped operands, then it must not resume the operation. EIS instructions interrupted by Missing Page faults must not be resumed from the point of interruption, but must be restarted. ****

6. Security Fault, Class 2 (SCL2)

A Security Fault, Class 2, is generated for the following flag 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 SDR_n_ instructions.

   3) An attempt is made to store data into type T = 1 or 3 segments using the STD_n_ 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) is not equal to 1. This fault is also detected in the CLIMB instruction when the new ISR is obtained and 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 as follows during the execution of the OCLIMB version of the CLIMB instruction if the data being loaded from the safe store frame is incorrect:

a. The descriptor to be loaded into the ISR does not have the following format:

   1) Type field T = 0.

   2) Flags 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, 22, 23, 27, and 28 = 1, and bits 21, 24, 25, and 26 = 0. **** DPS 88: Bits 23 and 24 are not checked. ****

3) Base field = 0 modulo-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 in conjunction with the CLIMB instruction (programmed, or as the result of a fault or interrupt), to report to the operating system that the safe store stack has only one or two 64-word frames remaining. This fault occurs and is reported as follows:

a. Programmed CLIMB

Programmed CLIMB. After completing the safe store on a programmed Inward CLIMB (SSR base and bound have been updated), if SSR bound < 191 words + 3 bytes, then the hardware does not access the instruction pointed to by the new ISR and IC, but executes the Safe Store Stack fault, which causes another safe store stack frame to be stored. This frame contains the "transferred to" domain registers from the programmed CLIMB. Word 5, bit 10 (SSSF) is set to one, and the fault code in bits 12-16 of word 5 are set to:

**** DPS 8 00000. ****
**** DPS 88 10111, to indicate the Safe Store Stack fault. ****

b. Fault or Interrupt CLIMB

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 along with the original fault or interrupt. If SSR bound < 191 words + 3 bytes, then the hardware sets word 5, bit 10 (SSSF) to one, leaving the original fault code (DPS 8: or interrupt cell #) in word 5, bits 12-16. The Safe Store Stack fault will NOT be executed; a separate safe store stack frame will NOT be stored.

NOTE: GCOS 8 monitors the SSSF bit in each fault or interrupt frame in the safe store stack and initiates appropriate action whenever this bit is 1.

c. Refer to Figures 8-3 (DPS 88) and 8-4 (DPS 8) for a description of the safe store stack.

8. **** DPS 8: Backup Fault

A Backup fault occurs if a fault or interrupt occurs during the initiation of a "wired-in" ICLIMB instruction, or if any fault occurs during the execution of this ICLIMB. ****

Hardware-Generated Faults

The hardware generated faults generally occur due to a failure in the hardware. The hardware generated faults are:

1. Operation Not Completed (FONC). This fault is generated due to one of the following conditions:

   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 (DPS 88: Central Interface Unit) terminated a double-precision cycle.

   c. When returning to an interrupted multiword instruction, incorrect data is loaded into the Pointer and Length Registers.

2. **** DPS 8: Parity (FPAR). 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 eliminated.

   c. Operand fetch. When a single-precision operand, C(Y), is requested, the contents of the memory pair at Y,Y+1 where Y is even, or Y-1,Y where Y is odd, are read from memory. The system controller will not report a parity error if it occurs in C(Y+1) or C(Y-1), but will restore 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 routine.

   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 will 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. **** DPS 88: Memory System (MEMSYS). This fault occurs on the following conditions:

   a. The data transfer from the CIU to the CPU is invalid. The selected CIU has returned a "fatal error" signal.

   b. An uncorrectable EDAC error has been detected by the CIU resident logic; note that detection is on an 8-word block basis.

   c. The processor has detected a parity error in the CIU interface port while verifying incoming parity. This condition will cause the CPU to Halt, in contrast to the prior system taking a fault; however, the resultant action may be a fault (SSF support is required).

4. Power Signal Faults. The power signal faults are as follows:

   a. Startup (SUF) - **** DPS 8: A Startup fault is generated when power restoration is detected. The operating system ignores the Startup fault. ****

      **** DPS 88: The Startup fault shall originate from the SSF maintenance computer; when the corresponding ASR control bit is set the Startup Fault shall be generated. ****

   b. Shutdown (SDF) - **** DPS 8: A Shutdown fault is generated when an impending power failure is detected. This fault is normally initiated by the frequency sensor that indicates decreasing rotational speed of the motor generator providing prime power to the system. The operating system ignores Shutdown faults. ****

      **** DPS 88: The power monitoring functions of the DPS 88 systems have detected a System shutdown condition. The Processor is notified via a signal sent to the collector which will cause a SDF fault. The source of the SDF fault can be the PASA or the SSF via an ASR control bit. ****

   c. Execute (EXF) - **** DPS 8: An Execute fault is generated when the EXECUTE switch on the processor maintenance panel is depressed. ****

      **** DPS 88: The Execute Fault shall originate from the SSF maintenance computer; when the corresponding Processor ASR control bit is set, the Execute Fault shall be generated. The Maintenance Panel Function (MPF) via the SSF provides the equivalent to the Execute pushbutton used on the prior systems. ****

5. **** DPS 8: Store Memory (STR) Fault. An STR fault is generated when an associative memory error occurs. However, an associative memory error is not detected during the execution of the STPDW or STPTW instruction. ****

MODE FAULTS

Privileged Master Mode Faults

When the processor is in Privileged Master (nonabsolute addressing) mode, all instructions must be fetched from a 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. 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, will be 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 shall result 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.

Any Mode Faults

Instructions that may refer to type T = 1 or 3 segments (LDP_n_, LDD_n_, SDR_n_, STD_n_, 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.)

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 given under "Memory Characteristics" discussed earlier in this document.

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 if a read from memory is allowed; if not 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 if 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 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 the segment to 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 it. 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).

**** DPS 88, DPS 8/20, 8/44: 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 instructions, 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 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. **** DPS 8: Bypass Cache Flag (bit 23) - Cache memory control operates as follows:

   a. The execution of one of the three read-and-clear memory instructions LDAC, LDQC, and SZNC does not cause the cache to be cleared. Also, operand fetches for these three instructions are always made from main memory, bypassing the cache. However, if a directory match occurs for the operand, the directory location is cleared.

   b. The execution of the Clear Cache (CCAC) instruction clears the 2K memory, but does not clear the 8K cache memory.

   c. The Cache Read Control Flag (CRCF) in the option register affects only operand reads from cache; instruction reads from cache continue to operate normally. When CRCF = 0, the cache is bypassed on all operand reads; if a directory match occurred for the operand, the directory location is cleared. When CRCF = 1, operand reads from cache operate normally.

      The CRCF has no effect on operand store operations; an operand store operation goes to backing store and, if a directory match occurs, the operand is also stored in cache memory.

      This control is subordinate to the mode register control.

   d. The CRCF also determines the type of memory command the processor sends to the Systems Control Unit (SCU) when performing operand read/store operations for all read-alter-rewrite (RAR) instructions. If CRCF = 0 on RAR instructions, the processor generates a read-lock/write-unlock command sequence to the SCU; if a cache directory match occurs, the directory location is cleared.

      If CRCF = 1 on all RAR instructions and a directory match occurs, the operand is read from cache, modified, and stored to both cache and backing store, and the processor generates a normal store command to the SCU. When a directory match does not occur, the processor generates a normal RAR command sequence to the SCU (the lock function is not invoked). ****

e. **** DPS 8/20, 8/44: When flag bit 23 of a segment descriptor with type T = 0, 1, or 4 is 0, cache memory is bypassed on all memory references using the descriptor. However, if a cache directory match occurs in a store operation (this should not normally happen), the operand will be stored both in cache and backing memory, but on a read operation cache is not interrogated. To avoid any inconsistency, operating system software ensures that this situation does not occur.

The above applies to both instruction and operand fetches. For example, if the descriptor in the ISR has flag bit 23 = 0, the cache is bypassed on all instruction fetches and all operand fetches not specifying a DRn; but if the instruction specifies a DRn for the operand address and the DRn has flag bit 23 = 1, normal cache usage would apply to the operand fetch. ****

f. **** DPS 8: Cache is not cleared upon the occurrence of an external interrupt.

g. If the virtual memory option is enabled but cache memory is not, the execution of the CCAC instruction results in no operation. Flag bit 23 has no effect on the operation, and the CRCF bit in the option register controls the command sequence generated by the processor to the SCU for RAR instructions as follows:

If CRCF = 0, a read-lock/write-unlock sequence is generated.

If CRCF = 1, a normal read/write sequence is generated.

At processor initialization, cache is cleared and disabled (turned OFF); it remains disabled until enabled by operating system software. When cache memory is disabled, flag bit 23 has no effect on the operation. ****

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).

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, is generated.

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 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 STR or 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 SDR_n_ instructions).

7. Available Segment Flag (bit 28) - The available segment flag indicates if 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 available) 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 V).

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.

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 IOX (but not IOM). 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 housekeeping bit of the PTW control field allows operating system software to assign certain mode-dependent use attributes on a page basis. The hardware monitors the PTW housekeeping bit on all instruction fetches, all operand fetches and stores, and all descriptor fetches and stores. The instructions and operands must be contained in a segment described as a type T = 0, 2, 4, or 6 descriptor and the pages may be assigned as housekeeping or nonhousekeeping pages. Descriptors to be used by a process must be contained in a type T = 1 or 3 segment and the pages must be assigned as housekeeping pages or the operation terminates with a Security Fault, Class 1.

4. IOM or IOX 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 associated with bit 35.

Mode Register Fault Traps (DPS 8 Only)

With the virtual memory option installed in the processor and enabled, the mode register functions as described below:

1. Bits 0-14 of the mode register are used as the modulo-8 real memory word address of the vector location from which the fault hardware obtains an entry descriptor. This address formed by the hardware is shown below:

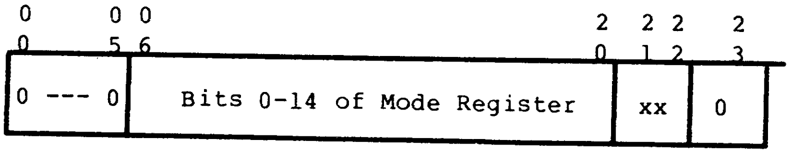


Figure 8-1. Fault Trap Address

xx - bits 21 and 22 are supplied by the hardware according to trap conditions as follows:

xx = 00, not used
xx = 01, op code trap
xx = 10, counter overflow
xx = 11, address match trap

2. The "trap on address match" is a comparison of the address switches on the maintenance panel with the 18-bit effective word address, which consists of y + Xn + ARn.

3. When one of the above three fault traps occurs, the resulting safe store frame uniquely identifies the fault as follows:

   a. Bit 9 of word 5 is set to 1.

   b. The 5-bit fault code is:

      00001 - op code trap
      00010 - counter overflow
      00011 - address match trap

Input-Output Multiplexer (IOM)-Detected Faults (DPS 8 Only)

The input-output multiplexer provides for the detection and indication of abnormal operating conditions, or faults. The two classes of faults recognized by the IOM are:

1. User faults

2. System faults

USER FAULTS

A user fault is an abnormal condition that may be caused by a user program operating in Slave mode in the processor. A user fault may be detected by the IOM Central or by a channel. If it is detected by the IOM Central, the fault is indicated to the channel and the channel is responsible for reporting the fault as a status in its regular status queue. A user fault condition does not cause the channel to be masked by the hardware, although the software may mask the channel. Because of their timing relationships, certain hardware malfunctions must be reported as user faults.

User faults are reported to the software in the channel status word as described below.

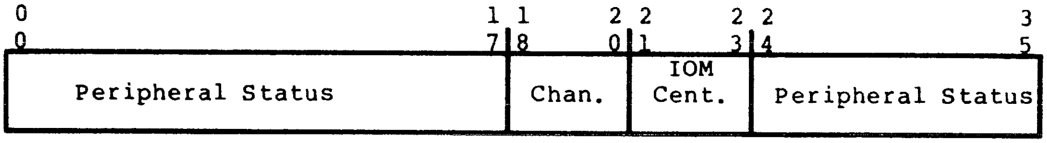


Figure 8-2. Channel Status Word

Chan. - Bits 18-20 indicate the channel status as determined by the channel.

IOM Cent. - Bits 21-23 indicate the central status as it was received from the IOM Central.

IOM Central-Detected User Faults

IOM Central faults are encoded on the user fault flag lines at the completion of service to the channel. The channel returns this code in the status word exactly as it is received from the IOM Central. The IOM Central status codes are given in Table 8-4.

Table 8-4. IOM Central Status Codes (DPS 8)

Fault Code (Octal)	Meaning
1	The list pointer word (LPW) tally field (bits 24-35) was zero and bits 21-22 contained 01, requesting tally checking.
2	The IOM Central was given two consecutive transfer data control words (TDCWs) during list service.
3	A boundary error occurred when performing the boundary check on a data control word (DCW) fetched during list service, with the data or DCW list referred to the page table.
4	A TDCW attempted to set the address extension control bit in the LPW (bit 20) when the LPW indicated restricted mode (LPW bit 18=1).
5	An instruction DCW (IDCW) was encountered during list service and the LPW indicated restricted mode (LPW bit 18=1).
6	A DCW fetched during list service indicated an illegal character position.
7	A parity error was detected on data from a channel during a data store service.

Since only three bits (bits 21-23) are available for central status indication, it is not possible to report simultaneous faults. The cause of a simultaneous fault indicated to the channel is the lowest numbered fault code described in Table 8-4.

Channel-Detected Faults

The channel status is defined as those fault conditions detected by the channel that are recorded in the channel status word, independent of a possible simultaneous indication from the IOM Central. The IOM Channel status codes are given in Table 8-5.

Table 8-5. IOM Channel Status Codes (DPS 8)

Fault Code (Octal)	Meaning
1	An unexpected peripheral control word (PCW) was encountered (i.e., a connect was received while the channel was busy).
2	An illegal instruction to the channel was encountered in the PCW.
3	The channel encountered an incorrect DCW.
4	The channel received an incomplete instruction sequence.
5	Not used.
6	A parity error occurred at the peripheral interface.
7	A parity error occurred on the I/O bus, data-to-channel-from-Central.

As in the case of IOM Central-detected faults, the channel-detected faults are ordered so that the lowest numbered fault is reported if simultaneous faults occur.

SYSTEM FAULTS

A system fault is an abnormal condition that cannot be caused by a user program operating in Slave mode, and therefore is assumed to have been caused by a software error or a hardware malfunction. System faults are detected by the IOM Central or the system controller and indicated by the system fault channel. The data channel being serviced when the system fault was detected is automatically masked by the IOM Central in an attempt to protect the system from another occurrence of the fault.

System Controller-Detected Faults

The system controller fault codes are placed in the system fault word by the IOM exactly as they are received on the illegal action lines from the system controller. The system controller fault codes are given in Table 8-6.

Table 8-6. System Controller Fault Codes (DPS 8)

Fault Code (Octal)	Meaning
1	Not used.
2	Nonexistent address.
3	Fault on condition (not used on a 4 megaword SCU).
4	Not used.
5	Data parity on transfer from memory to system controller.
6	Data parity in memory.
7	Data parity on transfer from memory to system controller and in memory.
10	Not control port (not used on a 4 megaword SCU).
11	Port disabled (masked).
12	Illegal instruction.
13	Memory not ready.
14	ZAC parity, active module to system controller.
5	Data parity, active module to system controller
16	ZAC parity, system controller to memory unit.
17	Data parity, system controller to memory unit.

IOM Central-Detected System Faults

The system faults detected by the IOM Central are given in Table 8-7.

Table 8-7. IOM Central System Faults (DPS 8)

Fault Code (Octal)	Meaning
1	The channel requested does not have a scratchpad.
2	A channel requested a service with an illegal code or a channel number of zero.
3	The page table pointer of the page table word scratchpad failed to properly store incoming data, or a parity error on the read data for the page table pointer, the page table word, or control words was detected.
4	The control word address is incremented to all zeros and the tally is not decremented to zero.
5	The tally was zero for an update LPW when the LPW was fetched for the connect channel.
6	The DCW fetched for the connect channel service did not have bits 18-20 equal to 7.
7	The DCW fetched for a data service was a TDCW or had bits 18-20 equal to 7.
10	The DCW fetched for a 9-bit channel specified an illegal character position.
11	No response occurred to an interrupt from a system controller within 16.5 microseconds.
12	A parity error occurred on the read data when accessing a system controller.
13	Illegal tally control for an LPW (bits 21-22 = 0) when the LPW was fetched for the connect channel. (May also indicate improperly installed NSAIG and NSAIE boards.)
14	The internally stored page table pointer flag for the requesting channel was zero. (May also indicate improperly installed cable from NSAIC to NSBIM.)
15	Caused by one of three conditions: a. Page missing. b. Channel data segmented (LPW 23=1), and indirect store service is required, and write control is reset (PTW 31=0) or housekeeping page is set (PTW 32=1). c. Channel requests a direct store and write control is reset (PTW 31=0) or housekeeping page is set (PTW 32=1).

Table 8-7 (cont). IOM Central System Faults (DPS 8)

Fault Code (Octal)	Meaning
16	The LPW fetched indicates use of address extension (LPW 20=1) while operating in the standard operating system mode.
17	No port was selected during an attempt to access memory.

INTERRUPT PROCEDURE

System Controller Interrupts (DPS 8)

Each system controller contains 32 interrupt cells that are used for communication among the active system modules (processors, I/O multiplexers, etc.). The interrupt cells are organized in a numbered priority chain. Any active system module connected to a system controller port may request the setting of an interrupt cell with a system controller command.

When one or more interrupt cells in a system controller are set, the system controller activates the interrupt present line to all system controller ports having an assigned interrupt mask in which one or more of the interrupt cells that are set are unmasked. Interrupt masks should be assigned only to processors.

During the initial part of the external interrupt procedure, the processor receives the 5-bit interrupt cell number from the System Controller Unit (SCU). After this number is received from the SCU, the processor generates an appropriate fault code and executes the "wired-in" ICLIMB version of the CLIMB instruction through the entry descriptor in locations 30-31 (octal). During the safe store portion of the ICLIMB, the hardware stores the 5-bit interrupt cell number and sets bit 11 of word 5 of the safe store frame ON to indicate that the safe store frame resulted in response to an interrupt.

Central Interface Unit Interrupts (DPS 88)

The Central Interface Unit (CIU) provides two sets of eight interrupt cells. Each set of eight can be assigned to either of the CPU ports, but each CPU port can have only one set of eight interrupt cells assigned to it. The eight interrupt cells correspond to the eight interrupt levels that can be selected to each I/O channel in the channel link word of the channel mailbox. Each CPU port has a mask register that permits the operating system to mask interrupts from any or all interrupt cells.

Each interrupt cell in the CIU has a queue of up to 256 entries (512 words) in Reserved Memory associated with it. The operating system obtains the next queue entry from Reserved Memory via the RIW (Read Interrupt Word) instruction. Each queue entry identifies the channel causing the interrupt, so that the operating system can locate the channel mailbox which contains the status information.

Inward Climb

The second word of the "wired-in" ICLIMB instruction has the following parameters:

E bit - 0 (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

bits 22-23 - 0 (ICLIMB version)

S,D fields - Ignored. **** DPS 8: If an entry descriptor is not found at a fixed memory location, the processor generates a Backup fault. ****

**** DPS 8: If an entry descriptor is not found at the fixed interrupt 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 DIS 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. ****

**** DPS 88: If an entry descriptor is not found in the interrupt vector location, or if a fault occurs while the processor is attempting to CLIMB to the interrupt handler, the SFF is notified and the processor halts. ****

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) is 1. If flag bit 26 of the new ISR is 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.

Pointer And Length Registers

Eight (DPS 88: two) 36-bit registers are utilized to store and load pointers for sending and receiving addresses and field lengths, and for other control information when a multiword instruction is interrupted.

The formats for these pointer and length registers are described earlier in this manual under the topic "Address Registers".

IC VALUES STORED ON FAULTS AND INTERRUPTS

If the safe store bypass flag in the option register is 0, a safe store is executed for any fault or interrupt. A description of the safe store stack is given in Figures 8-3 (DPS 88) and 8-4 (DPS 8).

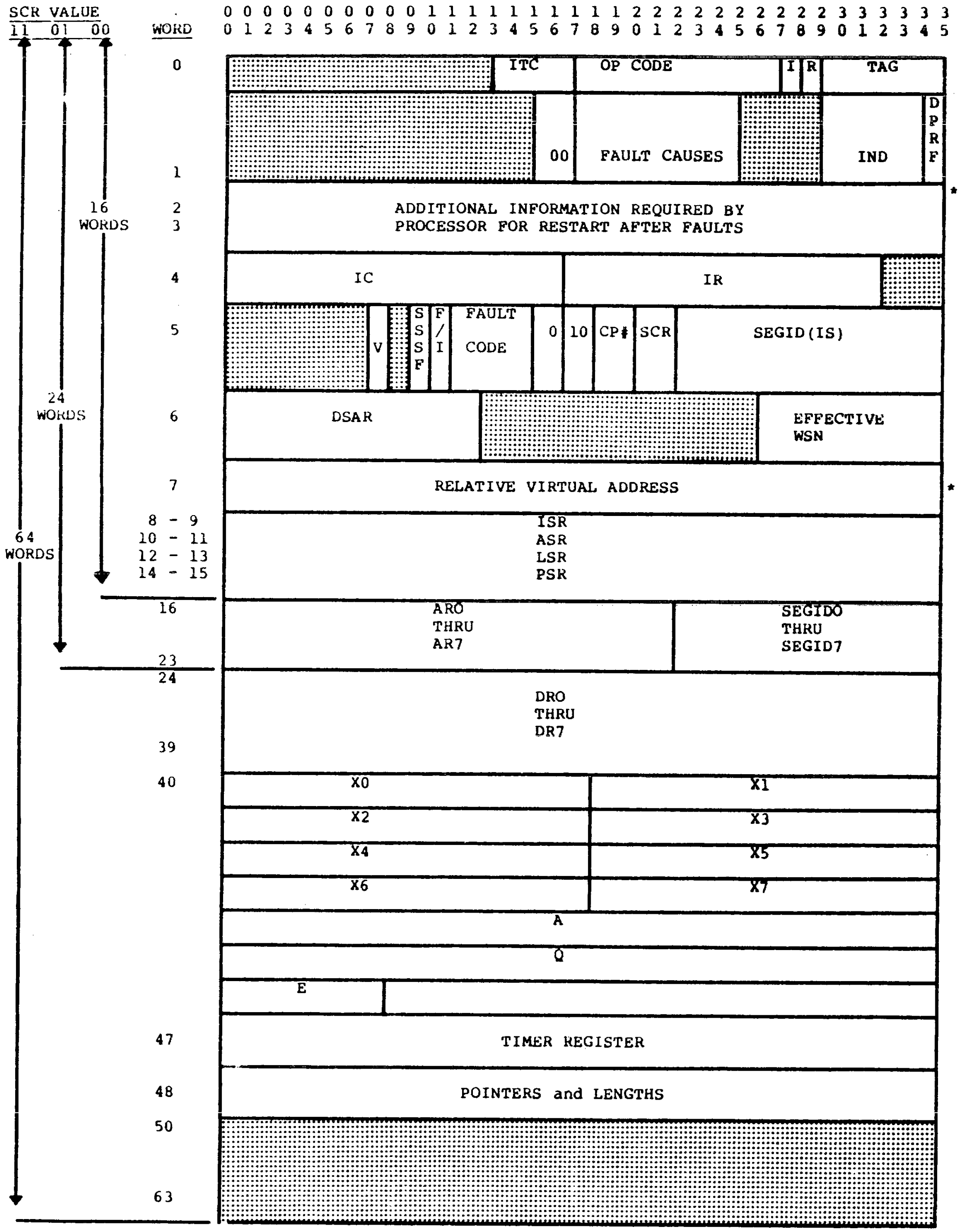


Figure 8-3. Safe Store Stack (DPS 88)

* Stored on faults only.

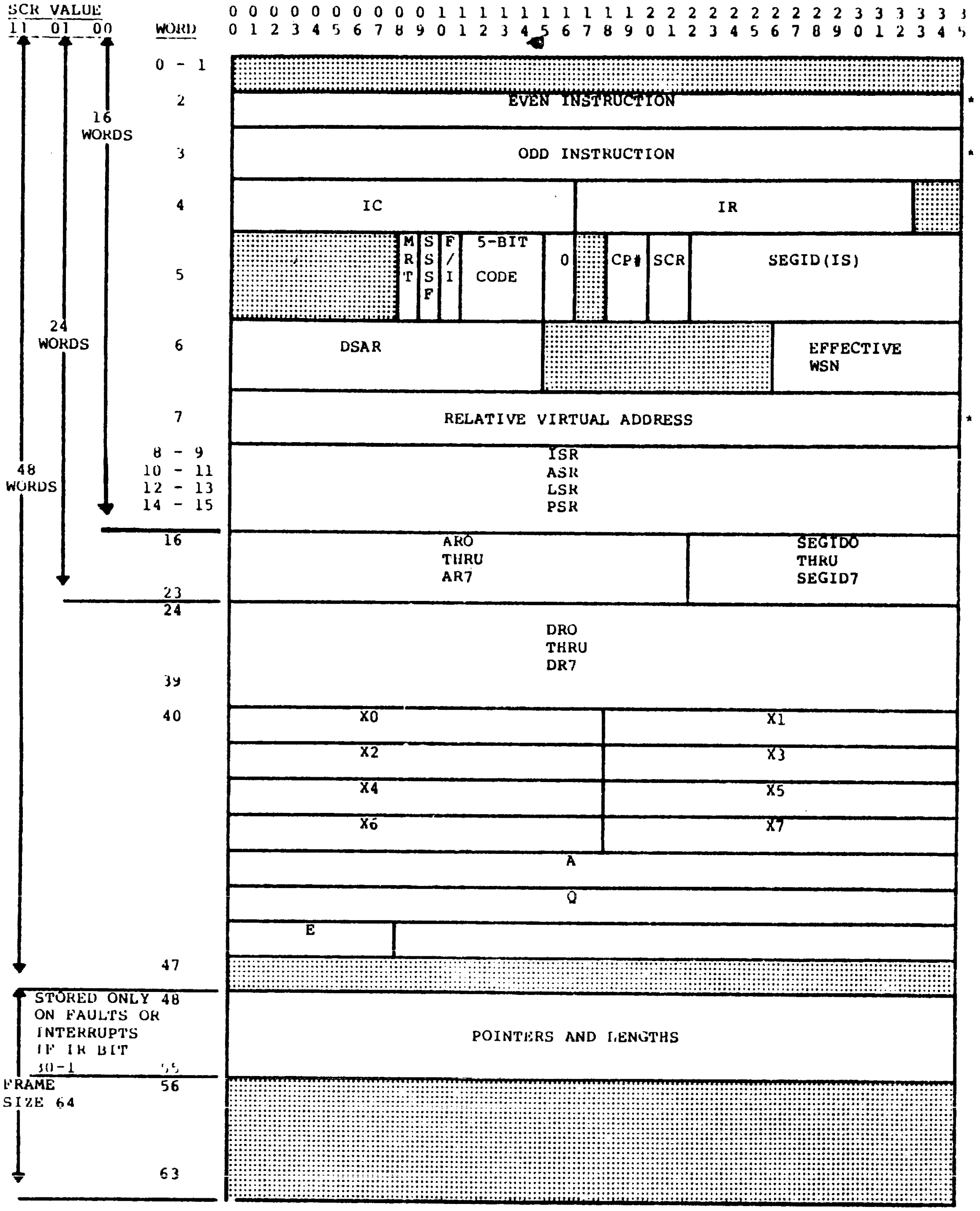


Figure 8-4. Safe Store Stack (DPS 8)

* Stored on faults and interrupts only.

**** DPS 8/20, 8/44: The instruction is stored in word 2 in Figure 8-4. Words 0, 1, and 3 are not used. In word 5, bit 8 is not used, but bits 17-18 contain 00. Word 47 is used for RTMR, word 5, bit 0 is for FRTRY, and words 48-51 is for Mid-Instruction Interrupt Recovery Data for Firmware. ****

The contents of the safe store stack frame following a fault or interrupt for DPS 8 are described in Table 8-8. The designation of the fault group priorities is given in Table 8-1.

The contents of the safestore stack frame following a fault or interrupt for DPS 88 are described in Table 8-9. The designation of the fault group priorities is given in Table 8-2.

Table 8-8. Classes Of Faults And Interrupts (DPS 8)

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 0-1	NOT USED									
INSTR. EV/000 WORDS 2-3	UNDEFINED	FAULTING PAIR						LAST PAIR COMPLETED	PAIR INCLUDING EIS INSTR.	PAIR INCLUDING CLIMB
IC_{00-17} WORD 4	UNDEFINED	IC OF LAST COMPLETED INSTR. +1			IC OF "TRANSFERRED TO" INSTR.	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 TRA	IS OF NEW INSTR.	IS OF CLIMB	CURRENT IS			IS OF CLIMB
DSAR, EWSN RVA WORD 6-7	LAST VALUE OF DSAR; EWSN AND RVA CORRESPOND TO LAST SEGMENT ACCESSED ****DPS 8/70, 8/50, 8/52, 8/62: SEQUENTIAL I FETCHES ARE NOT REFERRENCED****									
ISR WORDS 8-9	CURRENT			ISR OF TRA	ISR OF NEW DOMAIN	ISR OF CLIMB	CURRENT			PRIOR TO ISR 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 48-55	IF IR_{30}=1 AND ENTRY DESCRIPTOR T=11	NO	IF ENTRY DESCRIPTOR T=11	NO					IF ENTRY DESCRIPTOR T=11	NO
EVEN INSTR IS FAULTING INSTR. IF SAFE STORED IC	UNDEFINED	IC_{17}=0	IC_{17}=0	IC_{17}=0				N/A	IC_{17}=0	IF IC_{17}=0 CLIMB WAS EVEN

Table 8-9. Classes Of Faults And Interrupts (DPS 88)

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$_{3-}$ 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 88 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.

**** DPS 8: The definition of the classes of faults and interrupts contained in Table 8-8 is given below:

FAULT 1 - A group II to V fault not covered by FAULT 2 through FAULT 6, including XECs and RPTs. 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 due to 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.

An interrupt that occurs during an interruptible multiword instruction. ****

**** DPS 88: The definition of the classes of faults and interrupts contained in Table 8-9 are given below.

FAULT 1 - A fault, other than Group 1, not covered by FAULT 2 through FAULT 6, including executes and repeats. For faults on instructions which are executed or repeated, the IC and ISR represent the execute or repeat instruction.

FAULT 2 - A fault, other than Group 1, due to an EIS multiword instruction.

FAULT 3 - A fault, other than Group 1, while attempting to fetch "transferred to" instructions resulting from a TRA, TSXn, TSS, RET or a satisfied conditional transfer.

FAULT 4 - A fault, other than Group 1, occurring while attempting to fetch "transferred to" instructions resulting from a CLIMB instruction.

FAULT 5 - A fault, other than Group 1, occurring on a CLIMB instruction prior to fetching "transferred to" instructions.

FAULT 6 - A fault, other than Group 1, occurring on an inline instruction fetch.

INTER 1 - An interrupt occurring any time except during an interruptible EIS multiword instruction.

INTER 2 - An interrupt occurring during an interruptible EIS multiword instruction. ****

**** DPS 8: 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:

a. Programmed CLIMB -

    IC of CLIMB + 2

b. Interrupt during multiword instruction or connect, Timer Runout, or Shutdown faults during multiword instruction -

    IC of the first word of the multiword instruction

c. Interrupt after completed multiword or single-word instruction -

    IC of next instruction

d. Fault while attempting to fetch "transferred to" instructions resulting from a CLIMB instruction -

    IC of "transferred to" instruction

e. Safestore stack fault on programmed CLIMB

    IC of "transferred to" instruction

f. Startup or Execute fault -

    IC undefined

g. Operation Not Completed, Lockup, or Store Memory faults -

    DPS 8: IC of faulting instruction + 1
    DPS 88: IC undefined

h. Connect, Timer Runout, or Shutdown faults after completed multiword or single-word instruction -

    IC of next instruction

i. Any other fault -

    DPS 8: IC of faulting instruction + 1
    DPS 88: IC of faulting instruction

APPENDIX A

OPERATION CODE MAP

This appendix contains the operation code map for the processor 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)

	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017
000		MME	DRL							NOP	PULS1	PULS2	SYNC	CIOC		
020	ADLX0	ADLX1	ADLX2	ADLX3	ADLX4	ADLX5	ADLX6	ADLX7			LDQC	ADL	LDAC	ADLA	ADLQ	ADLAQ
040	ASX0	ASX1	ASX2	ASX3	ASX4	ASX5	ASX6	ASX7					AOS	ASA	ASQ	SSCR= LCCL
060	ADX0	ADX1	ADX2	ADX3	ADX4	ADX5	ADX6	ADX7		AWCA	AWCQ	LREG		ADA	ADQ	ADAQ
100	CMPX0	CMPX1	CMPX2	CMPX3	CMPX4	CMPX5	CMPX6	CMPX7		CWL				CMPA	CMPQ	CMPAQ
120	SBLX0	SBLX1	SBLX2	SBLX3	SBLX4	SBLX5	SBLX6	SBLX7						SBLA	SBLQ	SBLAQ
140	SSX0	SSX1	SSX2	SSX3	SSX4	SSX5	SSX6	SSX7						SSA	SSQ	
160	SBX0	SBX1	SBX2	SBX3	SBX4	SBX5	SBX6	SBX7		SWCA	SWCQ			SBA	SBQ	SBAQ
200	CNAX0	CNAX1	CNAX2	CNAX3	CNAX4	CNAX5	CNAX6	CNAX7		CMK	ABSA		SZNC	CNAA	CNAQ	CNAAQ
220	LDX0	LDX1	LDX2	LDX3	LDX4	LDX5	LDX6	LDX7		RSW= RRES	RMCM=	SZN RIMR	LDA	LDQ	LDAQ	
240	ORSX0	ORSX1	ORSX2	ORSX3	ORSX4	ORSX5	ORSX6	ORSX7						ORSA	ORSQ	STBZ
260	ORX0	ORX1	ORX2	ORX3	ORX4	ORX5	ORX6	ORX7						ORA	ORQ	ORAQ
300	CANX0	CANX1	CANX2	CANX3	CANX4	CANX5	CANX6	CANX7						CANA	CANQ	CANAQ
320	LCX0	LCX1	LCX2	LCX3	LCX4	LCX5	LCX6	LCX7						LCA	LCQ	LCAQ
340	ANSX0	ANSX1	ANSX2	ANSX3	ANSX4	ANSX5	ANSX6	ANSX7					STAC	ANSA	ANSQ	
360	ANX0	ANX1	ANX2	ANX3	ANX4	ANX5	ANX6	ANX7						ANA	ANQ	ANAQ
400		MPF	MPY			CMG				LDE	RIW	RSCR		ADE		
420		UFM		DUFM		FCMG		DFCMG	FSZN	FLD		DFLD		UFA		DUFA
440	SXL0	SXL1	SXL2	SXL3	SXL4	SXL5	SXL6	SXL7	STZ	SMIC	SCPR= SFR		STT	FST	STE	DFST
460		FMP		DFMP					FSTR	FRD	DFSTR	DFRD		FAD		DFAD
500	RPL					BCD	DIV	DVF				FNEG		FCMP		DFCMP
520	RPT	TTES	TTTL	TTU	TTEZ	FDI		DFDI		NEG		NEGL		UFS		DUFS
540										STBA	STBQ	SMCM= LIMR	STC1			
560	RPD					FDV		DFDV				FNO		FSB		DFSB
600	TZE	TNZ	TNC	TRC	TMI	TPL		TTF					TEO	TEU	DIS	TOV
620	EAX0	EAX1	EAX2	EAX3	EAX4	EAX5	EAX6	EAX7	RET		HALT	RCCL	LDI	EAA	EAQ	LDT
640	ERSX0	ERSX1	ERSX2	ERSX3	ERSX4	ERSX5	ERSX6	ERSX7					STACQ		ERSQ	
660	ERX0	ERX1	ERX2	ERX3	ERX4	ERX5	ERX6	ERX7					LCPR	ERA	ERQ	ERAQ
700	TSX0	TSX1	TSX2	TSX3	TSX4	TSX5	TSX6	TSX7	TRA					TSS	XEC	XED
720	LXL0	LXL1	LXL2	LXL3	LXL4	LXL5	LXL6	LXL7		ARS	QRS	LRS		ALS	QLS	LLS
740	STX0	STX1	STX2	STX3	STX4	STX5	STX6	STX7	STC2	STCA	STCQ	SREG	STI	STA	STQ	STAQ
760										ARL	QRL	LRL	GTB	ALR	QLR	LLR

Table A-2. Operation Code Map (Bit 27 = 1)

	000	001	002	003	004	005	006	007	010	011	012	013	014	015	016	017
000					MVNEX					CCAC						
020	MVE				MVNE											
040									STD0	STD1	STD2	STD3	STD4	STD5	STD6	STD7
060	CSL	CSR			SZTL	SZTR	CMPB									
100	MLR	MRL					CMPC		SDR0	SDR1	SDR2	SDR3	SDR4	SDR5	SDR6	SDR7
120	SCD	SCDR			SCM	SCMR										
140									STDSA	SPDBR	STO			STPDW		STPTW
160	MVT				TCT	TCTR	CMPCT		LDDSA	LPDBR	LDO				PAS	
200			AD2D	SB2D			MP2D	DV2D								
220			AD3D	SB3D			MP3D	DV3D								
240			AD2DX	SB2DX			MP2DX	DV2DX								
260			AD3DX	SB3DX			MP3DX	DV3DX								
300	MVN	BTD		CMPN		DTB										
320																
340	MVNX			CMPNX											CCAC0	CCAC1
360	MRF				MMF											
400											EPAT					
420																
440				SAREG				SPL	STP0	STP1	STP2	STP3	STP4	STP5	STP6	STP7
460				LAREG				LPL	LDP0	LDP1	LDP2	LDP3	LDP4	LDP5	LDP6	LDP7
500	A9BD	A6BD	A4BD	ABD				AWD								
520	S9BD	S6BD	S4BD	SBD				SWD	CAMP0	CAMP1	CAMP2	CAMP3				
540	ARA0	ARA1	ARA2	ARA3	ARA4	ARA5	ARA6	ARA7	STTD	STDSD		STTA				
560	AAR0	AAR1	AAR2	AAR3	AAR4	AAR5	AAR6	AAR7		LDDSD						
600	TRTN	TRTF			TMOZ	TPNZ	TTN		LDEA0	LDEA1	LDEA2	LDEA3	LDEA4	LDEA5	LDEA6	LDEA7
620									EPPR0	EPPR1	EPPR2	EPPR3	EPPR4	EPPR5	EPPR6	EPPR7
640	ARN0	ARN1	ARN2	ARN3	ARN4	ARN5	ARN6	ARN7								
660	NAR0	NAR1	NAR2	NAR3	NAR4	NAR5	NAR6	NAR7	LDD0	LDD1	LDD2	LDD3	LDD4	LDD5	LDD6	LDD7
700												CLIMB				
720																
740	SAR0	SAR1	SAR2	SAR3	SAR4	SAR5	SAR6	SAR7	STAS	STPS	STWS	STSS				
760	LAR0	LAR1	LAR2	LAR3	LAR4	LAR5	LAR6	LAR7	LDAS	LDPS	LDWS	LDSS				

APPENDIX B

STANDARD CHARACTER SET

The ASCII column is used to indicate the octal code generated by the assembler for an ASCII pseudo-operation (lowercase characters); however, the statement contains uppercase characters since it is converted before being acted upon by the assembler. The UASCI pseudo-operation allows the assembler to generate uppercase ASCII characters.

Standard Character Set	Internal Machine Code	Octal Code	Hollerith Card Code	Octal Codes Generated by Pseudo-Operations			
				BCD	ASCII	UASCI	EBCDIC
0	000000	00	0	00	060	060	360
1	000001	01	1	01	061	061	361
2	000010	02	2	02	062	062	362
3	000011	03	3	03	063	063	363
4	000100	04	4	04	064	064	364
5	000101	05	5	05	065	065	365
6	000110	06	6	06	066	066	366
7	000111	07	7	07	067	067	367
8	001000	10	8	10	070	070	370
9	001001	11	9	11	071	071	371
[	001010	12	2-8	12	133	133	112
#	001011	13	3-8	13	043	043	173
@	001100	14	4-8	14	100	100	174
:	001101	15	5-8	15	072	072	172
>	001110	16	6-8	16	076	076	156
?	001111	17	7-8	17	077	077	157
b̸	010000	20	(blank)	20	040	040	100
A	010001	21	12-1	21	141	101	301
B	010010	22	12-2	22	142	102	302
C	010011	23	12-3	23	143	103	303
D	010100	24	12-4	24	144	104	304
E	010101	25	12-5	25	145	105	305
F	010110	26	12-6	26	146	106	306
G	010111	27	12-7	27	147	107	307
H	011000	30	12-8	30	150	110	310
I	011001	31	12-9	31	151	111	311
&	011010	32	12	32	046	046	120
.	011011	33	12-3-8	33	056	056	113
]	011100	34	12-4-8	34	135	135	132
(	011101	35	12-5-8	35	050	050	115
<	011110	36	12-6-8	36	074	074	114
\	011111	37	12-7-8	37	134	134	340
^ or ↑	100000	40	11-0	40	136	136	137
J	100001	41	11-1	41	152	112	321
K	100010	42	11-2	42	153	113	322
L	100011	43	11-3	43	154	114	323
M	100100	44	11-4	44	155	115	324
N	100101	45	11-5	45	156	116	325
O	100110	46	11-6	46	157	117	326
P	100111	47	11-7	47	160	120	327
Q	101000	50	11-8	50	161	121	330
R	101001	51	11-9	51	162	122	331

Standard Character Set	Internal Machine Code	Octal	Hollerith Card Code	Octal Codes Generated by Pseudo-Operations			
				BCD	ASCII	UASCI	EBCDIC
-	101010	52	11	52	055	055	140
$	101011	53	11-3-8	53	044	044	133
*	101100	54	11-4-8	54	052	052	134
)	101101	55	11-5-8	55	051	051	135
;	101110	56	11-6-8	56	073	073	136
'	101111	57	11-7-8	57	047	047	175
+	110000	60	12-0	60	053	053	116
/	110001	61	0-1	61	057	057	141
S	110010	62	0-2	62	163	123	342
T	110011	63	0-3	63	164	124	343
U	110100	64	0-4	64	165	125	344
V	110101	65	0-5	65	166	126	345
W	110110	66	0-6	66	167	127	346
X	110111	67	0-7	67	170	130	347
Y	111000	70	0-8	70	171	131	350
Z	111001	71	0-9	71	172	132	351
_ or <-	111-1-	72	0-2-8	72	137	137	155
,	111011	73	0-3-8	73	054	054	153
%	111100	74	0-4-8	74	045	045	154
=	111101	75	0-5-8	75	075	075	176
"	111110	76	0-6-8	76	042	042	177
!	111111	77	0-7-8	77	041	041	117

INDEX

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE: DPS 8 & DPS 88 ASSEMBLY INSTRUCTIONS

ORDER NO. DH03-01

DATED JUNE 1984

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. □

FROM: NAME ______________________ DATE ______________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

Honeywell

Together, we can find the answers.

Honeywell

Honeywell Information Systems
U.S.A.: 200 Smith St., MS 486, Waltham, MA 02154
Canada: 155 Gordon Baker Rd., Willowdale, ON M2H 3N7
Mexico: Av. Constituyentes 900, 11950 Mexico, D.F. Mexico
U.K.: Great West Rd., Brentford, Middlesex TW8 9DH **Italy:** 32 Via Pirelli, 20124 Milano
Australia: 124 Walker St., North Sydney, N.S.W. 2060 **S.E. Asia:** Mandarin Plaza, Tsimshatsui East, H.K.

41483, 7.5C1084, Printed in U.S.A.