H E P

PRINCIPLES OF OPERATION

Denelcor, Inc. 3115 East 40th Avenue Denver, Colorado 80205 This publication reflects, with reasonable accuracy, the specifications in effect at the time it was written. Deneloor, Inc. reserves the right to change these specifications without notice.

Denelcor, Inc. assumes no liability for any damage resulting from or caused by reliance on the information presented. This includes, but is not limited to, typographical errors and the inadvertent or editorial omission of any information.

Copyright © 1981 by Denelcor, Inc.

All rights reserved

No part of this book may be reproduced in any form or by any means without the written permission of Denelcor, Inc.

SECTION	TITLE	PAGE
CHAPTER	1 - INTRODUCTION	
1.1 1.2 1.3 1.4 1.5	PROCESS EXECUTION MODULE	1-3 1-3 1-4
2.1 2.1.1 2.1.2 2.1.3 2.1.4 2.1.5 2.2 2.2.1 2.2.2 2.2.3 2.3 2.4 2.5 2.5.1 2.5.2 2.6 2.6.1	HEP MEMORIES. PROGRAM MEMORY. REGISTER MEMORY. CONSTANT MEMORY. DATA MEMORY. ERROR DETECTION AND CORRECTION DATA TYPES. FLOATING-POINT NUMBERS. INTEGERS. BIT VECTORS. DATA QUALITY. THE PROCESS STATUS WORD (PSW) HEP INSTRUCTION FORMATS. NONSTANDARD INSTRUCTIONS. DATA MEMORY INSTRUCTIONS. TRAPS.	2-2 2-2 2-3 2-3 2-3 2-4 2-5 2-5 2-6 2-7 2-8 2-10 2-10 2-13
	THE TRAP PSW	2-13
3.1 3.2 3.3	PROCESS CREATION AND TERMINATION	3-2
CHAPTER	4 - PRIVILEGED HEP OPERATION	
4.1 4.1.1 4.1.2 4.1.3 4.2 4.3 4.3.1 4.3.2	TASK MANAGEMENT. PROGRAM AND CONSTANT MEMORY ACCESS DATA MEMORY ADDRESSING. TASK STATES. PROCESS SCHEDULING. PROCESS STATUS. THE PROCESS STATUS (PS) FIELD. THE SFU STATUS WORD. TRAPS.	4-2 4-3 4-3 4-3 4-5 4-5

SECTION	TITLE	PAGE
CHAPTER	4 - PRIVILEGED HEP OPERATION (CONTINUED)	
4.5	MAINTENANCE SUPPORT	. 4-15
CHAPTER	5 - INSTRUCTIONS	
5.1	INTEGER ABSOLUTE VALUE	5-2
5.2	ACTIVATE TASK (PRIVILEGED)	
5.3	INTEGER ADD	
5.4	LOGICAL PRODUCT	
5.5	BRANCH	5-6
5.6	BIT CLEAR	5-/
5.7	BIT SETCALL	5-0
5.8 5.9	CLEAR	5-10
5.10	CLEAR ECC COUNTER (PRIVILEGED)	
5.11	CLEAR TASK COUNTER (PRIVILEGED)	
5.12	CREATE	. 5-13
5.13	CREATE PROCESS	
5.14	DEACTIVATE TASK (PRIVILEGED)	
5.15	INTEGER DECREMENT	
5.16	BIT ENCODE, LEFTMOST 0	
5.17	BIT ENCODE, LEFTMOST 1	
5.18	LOGICAL DIFFERENCE	
5.19	EQUIVALENCE	
5.20	FLOATING ADD	. 5-22
5.21	FLOATING DIVIDE	
5.22	FLOAT INTEGER PART	
5.23	CONVERT FLOATING-POINT TO INTEGER	
5.24	CONVERT INTEGER TO FLOATING-POINT	. 5-26
5.25	FLOATING MAXIMUM VALUE	
5 .26	FLOATING MINIMUM VALUE	
5.27	FLOATING MULTIPLY	
5.28	FLOATING SUBTRACT	
5.29	FLOATING COMPARE, INTEGER RESULT	
5.30	FLOATING COMPARE, FLOATING RESULT	5-32
5.31	FLOATING COMPARE, BIT VECTOR RESULT	
5.32	IMPLICATION	
5.33	INTEGER INCREMENT	
5.34	INCREMENT ON CARRY	
5.35	JUMP KILL TASK (PRIVILEGED)	5-3/ 5-29
5.36	LOAD DATA MEMORY	
5.37	LOAD ADDRESS	
5.38	LOAD DATA MEMORY INDIRECT	
5.39 5.40	LOAD DATA MEMORY INDEXED	
5.40	LUAD DATA MEMUKI INDEXED	5-42

SECTION	TITLE	PAGE
CHAPTER 5	- INSTRUCTIONS (CONTINUED)	
5.41	LOAD PSW	5-43
5.42	INTEGER MAXIMUM VALUE	
5.43	INTEGER MINIMUM VALUE	5-45
5.44	MODIFY PSW	5-46
5.45	REGISTER TO REGISTER MOVE	5-48
5.46	MASK REGISTER DESCRIPTOR	5-49
5.47	MOVE REGISTER WITH DESCRIPTOR	5-50
5.48	INTEGER MULTIPLY	5-51
5.49	NAND	5-52
5.50	NO-OPERATION	5-53
5.51	NOR	5-54
5.52	NOT	5-55
5.53	ONE	5-56
5.54	LOGICAL SUM	5-57
5.55	PRIVILEGED CREATE (PRIVILEGED)	5-58
5.56	PRIVILEGED REGISTER TO REGISTER MOVE (PRIVILEGED)	5-59
5.57	PRIVILEGED MASK REGISTER DESCRIPTOR (PRIVILEGED)	5-60
5.58	PRIVILEGED MOVE REGISTER WITH DESCRIPTOR (PRIVILEGED)	5-61
5.59	PRIVILEGED STORE PSW (PRIVILEGED)	5-62
5.60	PRIVILEGED SET REGISTER DESCRIPTOR (PRIVILEGED)	5-63
5.61	QUIT	5-64
5.62	RESET CREATE FAULT (PRIVILEGED)	5-65
5.63	READ CFU CONTROL (PRIVILEGED)	5-66
5.64	READ PEM CLOCK UNDER MASK	5-67
5.65	READ TASK COUNTER (PRIVILEGED)	
5.66	READ ECC COUNTER (PRIVILEGED)	5-69
5.67	READ FLAGS (PRIVILEGED)	
5.68	READ ENABLE REGISTER (PRIVILEGED)	5-71
5.69	READ PROGRAM MEMORY (PRIVILEGED)	
5.70	READ PSW (PRIVILEGED)	
5.71	READ SSW (PRIVILEGED)	5-74
5.72	READ TSW (PRIVILEGED)	
5.73	SHIFT ARITHMETIC	
5.74	SHIFT CIRCULAR	5-77
5.75	SET FLAG (PRIVILEGED)	5-78
5.76	SHIFT LOGICAL	5-79
5.77	STORE PSW	
5.78	STORE PSW AND QUIT	5-81
5.79	SET REGISTER DESCRIPTOR	
5.80	STORE TASK COUNTER	
5.81	STORE DATA MEMORY	
5.82	STORE DATA MEMORY INDIRECT	
5.83	STORE DATA MEMORY INDEXED	
5.84	INTEGER SUBTRACT	5-87

SECTION	TITLE	PAGE
CHAPTER 5	5 - INSTRUCTIONS (CONTINUED)	
5.85 5.86 5.87 5.88 5.89 5.90 5.91 5.92 5.93 5.94 5.95	SUPERVISOR CALL INTEGER COMPARE, INTEGER RESULT INTEGER COMPARE, FLOATING RESULT INTEGER COMPARE, BIT VECTOR RESULT INTEGER MULTIPLY UPPER WAIT ON FLAG (PRIVILEGED) WRITE PROGRAM MEMORY (PRIVILEGED) WRITE PSW (PRIVILEGED) WRITE SSW (PRIVILEGED) WRITE TSW (PRIVILEGED) EXCHANGE PSW	5-89 5-90 5-91 5-92 5-93 5-94 5-95 5-96
APPENDIX	A - GENERALIZED INSTRUCTIONS	
A.1 A.2 A.3 A.4 A.5	GENERALIZED FLOATING ADD INSTRUCTIONS	. A-5 . A-7 . A-9
FIGURES		
1.1 2.2.1 2.4.1 2.5.1	TYPICAL HEP SYSTEM	. 2-4 . 2-7
2.5.2 2.5.3 2.5.4 2.5.5	REGISTER OR CONSTANT MEMORY ADDRESS	2-10 2-11
2.5.6 2.5.7 2.5.8 2.6.1	DATA MEMORY ADDRESS FORMAT PARTIAL WORD ADDRESSING WITH B=0 PARTIAL WORD ADDRESSING WITH B=1 TRAP PSW FORMAT	2-11 2-12 2-12
3.2.1 3.2.2 4.1.1 4.3.1	REGISTER MEMORY ADDRESS FORMAT	. 3-3 . 3-4 . 4-1
4.3.2 4.3.3 4.3.4	SFU STATUS WORDSFU STATESSW REG FIELD FORMAT	. 4-7 . 4-9 4-11
4.4.1	TRAP PSW FORMAT	4-12

SECTION	TITLE	PAGE
TABLES		
2.2.1	FLOATING POINT REPRESENTATIONS	. 2-5
2.3.1	DATA QUALITY	. 2-6
2.4.1	USER TRAP MASK BITS	. 2-8
2.6.1	TRAP PSW PROGRAM COUNTER	2-14
2.6.2	TRAP CODES	2-15
3.3.1	TRAP-HANDLING TASKS	
4.1.1	BASE AND LIMIT ADDRESS FORMATION	. 4-2
4.3.1	PS FIELD CONTENTS	. 4-6
4.3.2	SSW CONTENTS	
4.3.3	SSW TRAP SUBFIELDS	4-10
4.3.4	SSW REG SUBFIELDS	4-11
4.4.1	TRAP PSW FIELDS	4-12
4.4.2	TRAP-HANDLING TASKS	4-13
4.4.3	TRAP CODES	4-13
5.79	REGISTER DESCRIPTOR CONTENTS	5-82

CHAPTER 1 - INTRODUCTION

CHAPTER 1

INTRODUCTION

The Heterogeneous Element Processor (HEP) is a large-scale, scientific digital computer system. It can execute a number of sequential or parallel programs simultaneously, and can be configured to support extremely large applications. It features 64-bit, floating-point arithmetic; up to 128 million words of physical memory; single-error correcting, double-error detecting (SECDED) capability; a high-performance file subsystem; and a modular design that allows for both field expansion of the system and programmable reconfiguration to keep most of the system available while a part of it is being maintained.

The major modules of the HEP system are listed below with some of their specifications.

Process Execution Module

2048 64-bit general purpose registers
4096 64-bit constant registers
32K-1024K 64-bit words of program memory, SECDED
ECL MSI technology
10 million instructions per second
1-60 user processes
1-60 supervisor processes
Relocation and protection for 16 tasks
Up to 16 modules per system

Data Memory Module

32K-1024K 64-bit words, SECDED ECL technology 20 million reads or writes per second Byte, quarterword, halfword and fullword addressable Up to 128 modules per system

I/O Cache Module

1-16 million 64-bit words, SECDED MOS technology

CHAPTER 1 - INTRODUCTION

20 million reads or writes per second 1-32 channels Up to 2.5 million bytes per second per channel Up to four modules per system

External I/O Module

32-1024 analog input channels

32-1024 analog output channels

32-1024 discrete input channels

32-1024 discrete output channels

Up to four modules per system

Figure 1.1 shows a typical medium-sized HEP system with eight million words of program memory, 16 million words of data memory, and four million words of I/O cache. This system executes 80 million instructions per second.

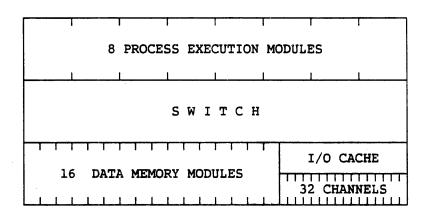


Figure 1.1 - Typical HEP System

1.1 PROCESS EXECUTION MODULE

Each HEP Process Execution Module (PEM) contains its own registers and program memory, and supports as many as 120 parallel processes executing simultaneously. Of these 120 processes, 50 are available for executing user programs; the rest perform operating system functions.

The 50 available processes can be distributed arbitrarily among protection domains called tasks. Each task has base and limit registers that relocate register and memory references and protect processes in differ-

CHAPTER 1 - INTRODUCTION

ent tasks from interfering with one another. It is also possible to allow processes in different tasks to communicate by establishing overlapping or identical task allocations. This is normally done only in data memory, to allow processes executing in different PEMs to share data.

A PEM supports multiple processes by time-multiplexing its control and execution hardware. Each process is periodically given a chance to execute an instruction; the logic of the PEM is pipelined so that several processes are in different phases of instruction execution at any moment. A user may create and terminate processes dynamically without operating system intervention. A minimum of eight processes are required to keep the PEM execution logic fully occupied, and to achieve 10 million instructions per second.

1.2 DATA MEMORY MODULE

To the user, data memory appears as one contiguous block of memory regardless of the number of modules used to implement it. The Switch connects all data memory modules to each PEM of the system to allow access by any process. A data memory module may also be connected to a single PEM by a direct data path; when this is the case, that data memory module is said to be "local" to the PEM.

Each data memory word has an associated access state, which may be either "full" or "empty". Store operations set the access state full, while load operations may set the access state empty. A store operation can be forced to wait until the word is empty before writing it and setting it full, and similarly a load can be forced to wait until the word is full before reading it and setting it empty. Access state testing is entirely under the control of the programmer. It synchronizes the flow of data between cooperating parallel processes. The general-purpose registers in each PEM also have the full-empty property to allow tightly coupled parallel computation, especially for the evaluation of arithmetic expressions.

1.3 I/O CACHE MODULE

The I/O Cache module is used as a buffer to match the 80 megabyte per second transfer rate of the Switch to the much lower transfer rates provided by disk or tape mass storage devices. The I/O Cache is addressable from the Switch and from as many as 32 channels, each of which can support disks or tapes having data rates up to 2.5 megabytes per second.

CHAPTER 1 - INTRODUCTION

Given a large number of disks and a nominal physical record size of 40K bytes, the system can achieve sustained I/O rates approaching 32 megabytes per second. The large size of the I/O cache allows many files to be open simultaneously; this allows a user to distribute data across files for higher performance.

1.4 EXTERNAL I/O MODULE

The External I/O (EIO) module contains analog conversion devices and discrete analog and digital signal interfaces to allow a HEP system to communicate with external hardware. The EIO module also contains a flexible programmable real time clock, and a facility that allows HEP processes to wait for external events by executing load instructions on certain locations associated with the EIO module. The result is similar to that obtained using vectored interrupts on nonparallel computers, but is much faster because it is unnecessary to save and restore processor state to handle the interrupt.

1.5 SWITCH

The modules of a HEP system are interconnected by the Switch, a high-speed packet switching network. The Switch nominally accepts 10 million 128-bit messages per second from each module connected to it, and distributes these messages among the other modules of the system.

Each node of the network has three full-duplex ports that connect it to modules or to other nodes. The nodes contain routing tables that are loaded during system initialization. These tables specify the optimal message routing (i.e. output port) for every possible destination module address. Message conflicts for the same output port at a node are resolved by sending all but one of the messages nonoptimally; these messages are incremented in priority to give them preference in subsequent conflicts.

The propagation delay of a switch node is 50 nanoseconds.

CHAPTER 2 - BASIC HEP OPERATION

CHAPTER 2

BASIC HEP OPERATION

Conventional computers execute a single instruction at a time on a single stream of data. This architecture, called SISD (for Single Instruction Stream Single Data Stream), is a subset of the HEP architecture. A program executing (or ready to execute) in the HEP system is called a process. HEP can support many processes executing simultaneously, but each process operates as if it were executing alone in an SISD computer, except for a few process synchronization operations. This chapter describes the SISD-compatible subset of HEP operations, which includes most instructions and capabilities.

HEP processes execute in a hardware unit called a Process Execution Module (PEM), which consists of a control unit, function units that implement the instruction set, and several memories. Instructions may access either these memories or a global memory accessible to all PEMs in the HEP system. Since the global memory contains most of the data, it is normally irrelevant which PEM contains a particular process.

2.1 HEP MEMORIES

A HEP system has four types of memory, all of which store 64-bit words:

- * Program Memory
- * Register Memory
- * Constant Memory
- * Data Memory

Program, register and constant memory are integral to each PEM; they are accessible only to those processes executing in the PEM in which they are installed. Data memory is global. It can be accessed from any

CHAPTER 2 - BASIC HEP OPERATION

PEM, and the address of a given location is the same regardless of which PEM accesses it.

2.1.1 PROGRAM MEMORY

Program memory is execute-only to nonprivileged ("user") processes and is used to store executable programs. It can accommodate programs up to 1 million instructions long.

Program memory space is allocated to a task by the operating system as part of the task definition when the task is loaded. Program memory can be read or written only by privileged insructions, which can be executed only by privileged ("supervisor") processes.

2.1.2 REGISTER MEMORY

Register memory consists of 2048 64-bit general purpose registers that are used to store operands and results for computations. Space in register memory is allocated to a task by the operating system as part of the task definition when the task is loaded. Space may be allocated to individual processes within a task to provide for reentrant programs.

Access to individual register memory addresses can be controlled for reading and writing by access control codes, which are described in detail in Chapter 3. These codes synchronize data that flows between processes executing parts of the same problem in parallel.

2.1.3 CONSTANT MEMORY

Constant memory contains 4096 64-bit registers that are read-only to user processes. Space in constant memory is allocated to a task and its processes in a manner similar to register memory. Constant memory is used interchangeably with register memory for operands to be used in computations, but can be used for storing results only by supervisor processes executing privileged instructions. Normally, constant memory is initialized by a supervisor process as the task is loaded.

CHAPTER 2 - BASIC HEP OPERATION

2.1.4 DATA MEMORY

Data memory can be read or written by all PEMs, but is not an integral part of any of them. It is used for storing most of the data in the system, and for communication between processes that are executing in parallel in different PEMs. Up to 128 million 64-bit words of this global data memory may be installed in a HEP system.

Data memory is also byte and partial word addressable. When partial-word or byte addressing is used, the low order three bits of the address specify the portion of the word to be read or written. The upper half of the data memory address identifies the I/O space, a 256-million-word area that can be addressed by supervisor processes as if it were data memory. This space supports I/O devices, I/O caches, supervisor communications, and other privileged functions.

2.1.5 ERROR DETECTION AND CORRECTION

Program memory and data memory use. a single-error correcting, double-error detecting (SECDED) code. Register and constant memory are parity-checked to detect single errors.

2.2 DATA TYPES

HEP supports three types of data in constant, register and data memory. These three types are:

- * Floating-point numbers
- * Integers
- * Bit vectors

CHAPTER 2 - BASIC HEP OPERATION

2.2.1 FLOATING-POINT NUMBERS

Floating-point numbers have the format shown in Figure 2.2.1.

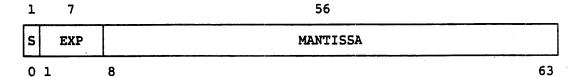


Figure 2.2.1 - Floating-Point Number Format

The meanings of the fields are:

S Sign bit. Set to 'l' if the number is negative.

EXP Exponent. To obtain the proper magnitude the floating-point number, the mantissa must be multiplied by the power of 16 represented by the exponent in bits 1-7. The exponent is stored as an excess-64 number; (40 hexadecimal) must be subtracted from the representation to arrive at the true exponent. This allows representation of exponent values in the range -64 through +63, so the magnitudes of numbers that can be represented range from 16**-65 to (16**63)(1-16**-14), or about 10**-78 (decimal) to 10**75 (decimal).

MANTISSA The mantissa is normally expressed in hexadecimal digits, each represented by four binary bits. The 14 hexadecimal digits occupy bits 8-63 of the data word. The radix point of the mantissa is defined to be immediately to the left of the high order digit.

The overall representation of the floating-point number is sign-magnitude; bits 1-63 represent the magnitude of the number, and the sign (S) bit is set to 1 if the number is negative.

Floating-point numbers are conventionally presumed to be normalized. In normalizing a floating-point number, the mantissa digits of a result are shifted left to remove leading zeroes and the exponent is decremented accordingly. Table 2.2.1 contains several examples of normalized and unnormalized floating-point representations.

CHAPTER 2 - BASIC HEP OPERATION

NUMBER	FLOATING-POINT REPRESENTATION	NORMALIZED
0	00000000000000	YES
0	2B0000000000000	NO
0	80000000000000	МО
1	411000000000000	YES
1	4E0000000000001	МО
-1	C11000000000000	YES
PI	413243F6A8885A31	YES
MAX	7 FFFFFFFFFFFF	YES
MIN	001000000000000	YES
-MIN	801000000000000	YES
-MAX	FFFFFFFFFFFFF	YES

Table 2.2.1 - Floating-Point Representations

Floating-point operations in HEP include add, subtract, multiply, divide, absolute value, maximum, minimum, comparison and conversion to integer.

2.2.2 INTEGERS

Integers are stored in 64-bit binary, two's-complement notation. For a positive number, the sign bit (bit 0, as for a floating-point number) has the value '0'; for a negative number, the sign bit has the value '1'. The negative of any integer (except -2**63, which has no positive counterpart) is obtained by complementing (inverting) all bits of the number and adding one to the result.

Integer operations in HEP include add, subtract, multiply, arithmetic shift, increment, decrement, absolute value, maximum, minimum, comparison and conversion to floating-point. Two separate multiply instructions give access to all 128 bits of the product of two 64-bit integers.

2.2.3 BIT VECTORS

Bit vectors are merely unsigned 64-bit strings. The instructions that operate on this data type implement all sixteen logical functions of two variables, logical shift, circular shift, operations that determine the numeric bit position of the leftmost 1 or 0, and operations that set or clear a bit at any specified numeric position.

CHAPTER 2 - BASIC HEP OPERATION

2.3 DATA QUALITY

The result of a HEP computation has two parts. The major part is the data value, which is one of the data types just defined. The other part is called the data quality. The data quality is a three-bit tag associated with the data in a result location (in addition to the 64-bit data value). The data quality of a result indicates how usable the associated data value is. It can be explicitly checked by the user program, or can be used later to generate an error trap when the data in the result is used or stored in data memory. The possible values for data quality and the interpretation of these values are in Table 2.3.1. Traps are discussed later in this chapter.

The only HEP memories that record data quality are register and constant memory. When a value is stored in data memory and reloaded into register memory, the original data quality is lost and the new one is zero. If several different data quality values might be expected to occur in one instruction, only the highest data quality is stored. Thus, integer overflow masks carry. Similarly, parity error masks all other results.

DATA	QUALITY	MEANING	GENERATING OPERATIONS
	000	Good Data (normal result)	Any
	001	Carry	Integer Operations
	010	Underflow	Floating-Point Operations
	011	Loss of Significance	Floating-Point Add, Subtract and Integer to Floating-Point Conversion
	100	Overflow	Floating-Point and Integer Operations
	101	Indefinite Result	Floating-Point Divide
	110	Parity Error in Operand	Any
	111	Unused	None

Table 2.3.1 - Data Quality

CHAPTER 2 - BASIC HEP OPERATION

2.4 THE PROCESS STATUS WORD (PSW)

Each process in a HEP PEM is defined and controlled by a 64-bit Process Status Word (PSW). The PSW format is shown in Figure 2.4.1.

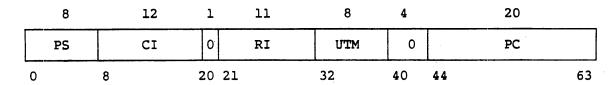


Figure 2.4.1 - Process Status Word Format

The meanings of the fields are:

- PS Privileged field. The PS field is accessible only to privileged instructions executed by supervisor processes. If a PSW is accessed by a user process, the PS field value is always zero. The PS field is described in Figure 4.3.1.
- CI Constant Memory Index field. When indexed addressing is specified in constant memory, the address is calculated by adding the CI value to a displacement supplied in the instruction. The CI field can be read and written by user processes.
- RI Register Memory Index field. This field is similar to CI, but applies to register memory rather than constant memory.
- UTM User Trap Mask. This field specifies which data exception traps can occur. Each bit in the UTM field corresponds to a particular data quality value. (Thus, bit 0 of the UTM corresponds to data quality 000, bit 1 to 001, and so on.) A trap occurs when a bit in the UTM is '1' and an instruction operand has the corresponding data quality value. Table 2.4.1 shows the correspondence between PSW bits and data quality.

CHAPTER 2 - BASIC HEP OPERATION

PSW BIT	UTM BIT	DATA QUALITY	MEANING
32	.0	0	Good Data (Normal)
33	1	1	Carry
34	2	2	Underflow
35	3	3	Loss of Significance
36	4	4	Overflow
37	-5	5	Indefinite Result
38	6	6	Parity Error
39	7	7	Unused

Table 2.4.1 - User Trap Mask Bits

If all bits in the UTM are set, any instruction with operands (virtually every HEP instruction) generates a trap. The UTM field can be read and written by user processes.

PC Program Counter. The program counter is used to address the next instruction in program memory to be executed. The program counter is incremented as each instruction is executed. The PC field can be modified by PSW instructions, effectively causing the process to jump to a different place in the program.

2.5 HEP INSTRUCTION FORMATS

All HEP instructions occupy a 64-bit word in program memory. Most of them are three-address instructions. The format of these instructions is shown in Figure 2.5.1.

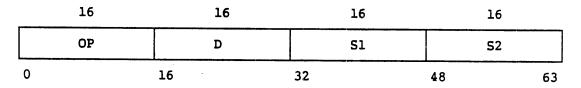


Figure 2.5.1 - Three Address Instruction Format

The meanings of the fields are:

OP The 16-bit operation code field for the operation to be performed.

CHAPTER 2 - BASIC HEP OPERATION

- D Destination. The address of the location in register memory or constant memory in which the result of the instruction is to be stored. If D specifies a constant memory location, the executing process must be a supervisor process.
- Source 1; the address of the source location in register of constant memory containing the first operand of the instruction.
- Source 2; the address of the source location in register or constant memory containing the second operand of the instruction.

The format of the D, Sl and S2 fields of the instruction is shown in Figure 2.5.2.

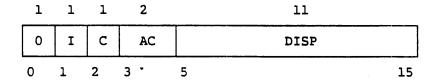


Figure 2.5.2 - Register or Constant Memory Address

The meanings of the fields are:

- I Indexed addressing specification bit.
 - I = 1 means add the index value (CI or RI) from the PSW when calculating the effective address.
 - I = 0 means do not add the index value from the PSW when calculating the effective address.
- C Memory type specification bit.
 - C = 1 means the address is in constant memory. (This is illegal
 in the destination field for a user process.)
 - C = 0 means the address is in register memory.
- AC This field is concatenated with DISP for addresses in constant memory, to provide a 13-bit displacement.

CHAPTER 2 - BASIC HEP OPERATION

For addresses in register memory, AC is the access control specification field. The interpretation of this field is described in Figure 3.2.1.

The values in the locations specified by the S1 and S2 fields are never modified by the instruction unless D=S1 or D=S2. Not all instructions require all 3 fields; unused fields must be set to zero.

2.5.1 NONSTANDARD INSTRUCTIONS

Certain instructions deviate from the standard form. For example, the branch instruction contains a 20-bit program memory address and has the format shown in Figure 2.5.3.

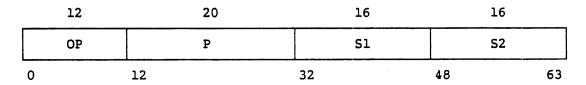


Figure 2.5.3 - Branch Instruction Format

The OP field is reduced to 12 bits to make room for the address P, a 20-bit field whose contents may conditionally replace the PC field of the PSW. S1 and S2 are the same as in standard instructions.

2.5.2 DATA MEMORY INSTRUCTIONS

Some data memory instructions are also nonstandard. In these instructions, a 32-bit data memory displacement field replaces two contiguous 16-bit fields normally used to hold register or constant memory addresses. The formats for these instructions are shown in Figures 2.5.4 and 2.5.5. Figure 2.5.4 shows the format for the Load (LOD) and Load Address (LODA) instructions; Figure 2.5.5 shows the format of the Store (STO) instruction.

CHAPTER 2 - BASIC HEP OPERATION

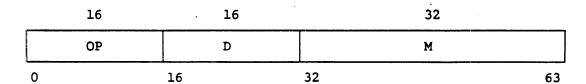


Figure 2.5.4 - Instruction Format for LOD and LODA

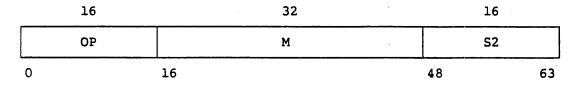


Figure 2.5.5 - Instruction Format for STO

The 32-bit data memory displacement M is actually the concatenation of a 29-bit word offset W and a 3-bit partial word specifier PW. For these instructions, The data memory displacement is combined with three other fields obtained from the low order 5 bits of the OP field to form the data memory address (See Figure 2.5.6).

1	2	2	29	3	
В	SAC	DAC	W	PW	

Figure 2.5.6 - Data Memory Address Format

In Figure 2.5.6 SAC and DAC are access control specifications used for synchronizing parallel operations and are described in Section 3.3. The W field is the displacement of a 64-bit word in data memory, and the B and PW fields specify the portion of the data memory word to be addressed, as described below.

If B=0 a quarter-word (16 bits), half-word (32 bits), or full-word (64 bits) is addressed. The PW field specifies which bits are addressed (See Figure 2.5.7).

CHAPTER 2 - BASIC HEP OPERATION

If B=1 an 8-bit byte of the data memory word is addressed. The specific byte addressed is defined by the PW field (See Figure 2.5.8).

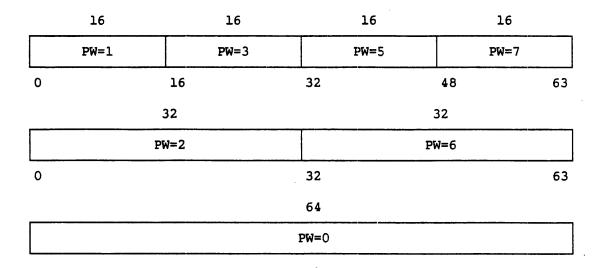


Figure 2.5.7 - Partial Word Addressing with B=0

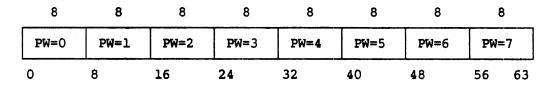


Figure 2.5.8 - Partial Word Adressing with B=1

The operation of the PW field allows a data memory displacement to address successive portions of words by adding the size of the portion in bytes. For example, successively adding 2 to a quarter-word displacement addresses successively higher quarter-words, automatically incrementing the W (displacement) field by 1 when PW changes from 7 to 1.

The data memory displacement appears in the instruction itself for the instructions LOD, LODA, and STO (as shown in Figures 2.5.4 and 2.5.5). It appears in the low order 32 bits of the word addressed by Sl for the remaining data memory reference instructions: Load Indirect (LODI),

CHAPTER 2 - BASIC HEP OPERATION

Store Indirect (STOI), Load Indexed (LODX), and Store Indexed (STOX). For these instructions, the data memory address may be formed by concatenating the B, SAC and DAC subfields with the data memory displacement (as described above), or may be contained in bits 27-31 of the word addressed by S1. The LODX and STOX instructions form an effective address by adding the low order 32 bits of the word addressed by S2 to the displacement in S1 modulo 2**32; for the other instructions, the displacement itself is the effective address.

The STOX instruction is nonstandard because the left-most address field is a source field rather than a destination field, and is referred to as Source 0 (S0).

2.6 TRAPS

All error and exception conditions in HEP are handled in a uniform way. The PSW of the process generating the exception is saved, and a new PSW is created. Both the old and new PSW contain information about the exception condition. The new PSW corresponds to a supervisor process that normally executes operating system code, but the information in the new and old PSW is made available to the user for diagnostic purposes.

The old PSW has the format described in Section 2.5. If bit 20 of the new PSW is 1, the PC field of the old PSW points to the instruction following the one that generated the exception. If bit 20 of the new PSW is 0, the old PSW points to the generating instruction.

2.6.1 THE TRAP PSW

The new (trap) PSW format is shown in Figure 2.6.1.

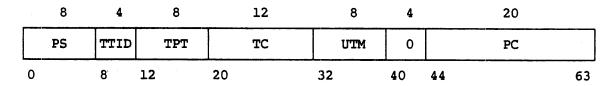


Figure 2.6.1 - Trap PSW Format

CHAPTER 2 - BASIC HEP OPERATION

The PS, TTID and TPT fields contain information that is either privileged or associated with processes executing in parallel. These fields are discussed in Chapter 4. The UTM is set to zero.

The PC field defines the type of trap. User programs can generate traps with the PC values shown in Table 2.6.1.

PC	MEANING
00010	Memory error - ECC hardware detected an uncorrectable error in a program memory or data memory location.
00012	Illegal Instruction - The opcode of the instruction is not in the HEP instruction set.
00014	Protection Violation - An illegal memory access was attempted.
00016	Privileged Instruction - A user process attempted to execute a privileged instruction.
0001A	Supervisor Call - The operating system is to perform some function on behalf of the user.
0001C	Simulated Instruction - The opcode of the instruction is to be interpreted and carried out by software.
0001E	Data Exception - A source data quality specified by the user's UTM was encountered.
	Table 2.6.1 - Trap PSW Program Counter

Other traps are possible, but they relate to parallel processing and privileged operations (See Chapter 4).

The TC field is the trap code. If the trap is a data exception (with a trap PC of 0001E), then the trap code is interpreted as three four-bit subfields specifying which operand data qualities caused the trap. If the high bit of a given subfield is zero, the remaining three bits in that subfield are undefined; if the high bit is one, the remaining three bits specify a data quality that caused the trap. The left-most subfield (trap PSW bits 20-23) is significant only when a data exception occurs in Source 0 of a STOX instruction. Note that bit 20 will be set in this case, so the PC field of the old PSW points to the instruction following the STOX that generated the trap. The remaining two subfields

CHAPTER 2 - BASIC HEP OPERATION

(trap PSW bits 24-27 and 28-31) are significant when a data exception occurs in Source 1 or Source 2 of an instruction respectively.

If the trap is not a data exception, then the trap code conveys information about the source of the error. When multiple traps are possible, the trap with the smallest trap PC is generated. As a result, a memory error or illegal instruction trap takes priority over a protection violation. The trap codes that can result from these three types of trap are described in Table 2.6.2. The trap codes for privileged instruction and simulated instruction traps are always zero. The trap code for an SVC instruction is obtained from the D field of the instruction.

TC	BIT	PSW BIT	MEANING IF SET
	0	20	Data memory uncorrectable error or protection violation.
	1-5	21-25	(unused)
	6	26	Unimplemented data memory location.
	7	27	Data memory protection violation.
	8	28	D or SO protection violation.
	9	29	Sl protection violation.
	10	30	S2 protection violation.
	11 .	31	Program memory protection violation.

Table 2.6.2 - Trap Codes

Protection violations can result either from an attempt to use a constant memory location as the destination of an instruction executed by a user process or from an attempt to reference a memory location not allocated to the task containing the process. Tasks are described in Section 3.3.

CHAPTER 3 - PARALLEL HEP OPERATION

CHAPTER 3

PARALLEL HEP OPERATION

In a conventional "multiprocessing" computer, several independent or cooperating processes exist simultaneously within the machine, but only one of these processes is executing at any given time. The remaining processes are inactive, managed by a software process scheduler that periodically stops the executing process and starts a previously inactive one. In these systems, a "process" is a software artifact; creating and deleting processes is a function of the operating system.

In HEP, multiple processes exist simultaneously in hardware. A single HEP PEM can have up to 128 simultaneously-active PSW's at any one time. Each PSW controls a separate instruction stream, and all instruction streams execute in parallel, multiplexed under hardware control. With this architecture, creating and deleting processes is a hardware function implemented by nonprivileged instructions.

3.1 PROCESS CREATION AND TERMINATION.

Processes in the HEP are created in one of three ways. The normal method is for an existing process to execute the CREATE instruction. One of the operands of this instruction is a register memory or constant memory location containing a PSW. The value in the location is placed in the hardware process queue; the instruction stream pointed to by its PC field subsequently begins executing. The process executing the CREATE proceeds to its next sequential instruction. Options exist in CREATE to allow fields in the newly-created PSW to be inherited from the PSW of the creating process instead of the instruction operand.

The other two ways of creating processes are by a trap or by performing Initial Program Load, an external function. These are discussed in Section 3.3.

CHAPTER 3 - PARALLEL HEP OPERATION

HEP processes terminate by executing a quit (QT or SQT) instruction. After executing QT or SQT, the process executes no further instructions. The hardware process queue location associated with the process is freed, and may be reused by a create in another process. There is no way for a user process to terminate another process; to be terminated, a process must execute a quit instruction. A privileged operation does exist to terminate entire groups of processes. This is discussed in Chapter 4.

3.2 PROCESS SYNCHRONIZATION

Normally, HEP processes execute independently. To allow a single computing problem to be solved by several processes in parallel (thus reducing the solution time), hardware mechanisms allow processes to synchronize and communicate.

Register memory and data memory locations possess an attribute called the "access state". Each register or data memory location may be "full" or "empty". In addition, register memory locations may be "reserved". The access state of a location is independent of the value in the location and may be manipulated without changing the value. Constant memory and program memory do not have access states.

The reserved state prevents any use of a location as the source or destination of an instruction. This effect is automatic and not under the programmer's control. A register memory location is set reserved when an instruction begins executing with that location as its destination operand. The state changes to full when the result value is stored. In the meantime, no other process may access the location.

The full/empty state of operands is under the programmer's control. Register memory address fields contain a 2-bit access control (AC) field that controls the testing and setting of the register memory location access state. Figure 3.2.1 shows the format of a register memory address.

CHAPTER 3 - PARALLEL HEP OPERATION

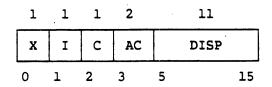


Figure 3.2.1 - Register Memory Address Format

For sources the AC field has the following meaning:

AC	EFFECT	ASSEMBLER MNEMONIC
00	read if not reserved, do not alter state	none
01	read if not reserved, set state to empty	: U
10	read only if full, leave full	:W
11	read only if full, set state to empty	:U:W

For destinations the AC field has the following meaning:

AC	EFFECT	ASSEMBLER MNEMONIC
00	write if not reserved, set full	none
01	write only if empty, set full	:E
10	write only if full, leave full	:F
11	invalid (instruction will never execute)	:E:F

Note that the destination is always set full; a source location may or may not be emptied.

Data memory locations also possess the access state attribute, but may only be full or empty. Data memory access control information is contained in the data memory address. Figure 3.2.2 shows the format of a data memory address.

CHAPTER 3 - PARALLEL HEP OPERATION

1	2	2	29	3
В	SAC	DAC	W	PW

Figure 3.2.2 - Data Memory Address Format

The SAC (Source Access Control) field has the following meaning:

SAC	EFFECT	ASSEMBLER MNEMONIC
00	read, do not alter state	none
01	read, set state to empty	:U
10	read only if full, leave full	: W
11	read only if full, set state to empty	W:U:

The contents of the SAC field are interpreted only by the load operations LOD, LODI and LODX.

The DAC (Destination Access Control) field has the following meaning:

DAC	EFFECT	ASSEMBLER MNEMONIC
00	write, set full	none
01	write only if empty, set full	: E
10	write only if full, leave full	: F
11	invalid (instruction will never execute)	:E:F

The DAC field is interpreted only by the store operations STO, STOI and STOX. The access state of data memory applies to the entire word; if a partial word is stored, the entire word is set full, and if a partial word is loaded with SAC=01 or SAC=11, the entire word is set empty.

When the operands of an instruction are in an incorrect access state (reserved, or in a state failing a test), the instruction is not executed, none of the operands is modified in any way, and the process is placed at the tail of the process queue. After a delay whose length depends on machine loading, the process reaches the head of the process queue and the execution of the instruction is reattempted. This sequence continues until the operands are in the correct access state, at which time the instruction is executed.

CHAPTER 3 - PARALLEL HEP OPERATION

Processes use access control to pass data to each other. A receiving ("consumer") process can wait to read data from an initially empty location using AC or SAC=11. A sending ("producer") process writes the data with AC or DAC=01. When the producer fills the location, the consumer, whose instruction has been waiting, obtains the data and sets the location empty again. If the consumer was not ready for the data and does not read and empty the location before the producer attempts to write a second time, the location is left full. Thus the producer is prevented from placing more data in the location until the consumer empties it.

When a PEM has local data memory, the operations STO and STOI on full words with SAC and DAC both zero do not modify the access state of a location local to the PEM. In particular, the location is not set full. The STOX operation, however, always sets full.

3.3 TASKS

Each process in a HEP PEM belongs to a set of processes called a task. To make HEP programs relocatable, and to protect different users from each other, each task has base and limit addresses for each memory type. These addresses are held in the hardware, and are used to map separate problems into nonoverlapping partitions.

Each PEM supports 16 tasks numbered 0 through 15. Task 0 is not used. Tasks 1 through 7 are called user tasks, and contain user processes. Tasks 8 through 15 are called supervisor tasks and contain supervisor processes. A task may contain any number of processes up to 64. No more than 64 user processes and 64 supervisor processes can exist at any moment.

For processes in each task, the base address for each memory type is added to each effective address to form a real memory address. Thus all code is written as if its memory spaces begin at location zero; the real memory address for the task is transparent to the user. For each access (except for constant memory), the real memory address is compared with the limit address. If the real address is greater, a protection violation trap is generated. The absence of a constant memory limit address prevents the detection of illegal constant memory references. But user tasks cannot modify constant memory so destructive task interaction is impossible, though a task may read constants not part of its memory allocation.

A task in HEP has two states: active and dormant. Processes in an active task are scheduled for execution and make computational progress in the normal way. Processes in a dormant task are not selected for execu-

CHAPTER 3 - PARALLEL HEP OPERATION

tion, and do not make progress. Tasks move from one state to the other via certain privileged instructions and by generating traps.

When a trap occurs, the trap hardware makes the trapping task dormant and creates a process in the task that handles the trap. The trap handling task is shown in Table 3.3.1.

TRAPPING TASK TRAP HANDLING TASK

0	8
1	9
2	10
3	11
4	12
5	- 13
6	14
7	15
8	8
9	8
10	8
11	8
12	8
13	8
14	8
15	8

Table 3.3.1 - Trap-Handling Tasks

This relationship between trapping and trap-handling tasks is normally used to implement a three-level operating system in which each user task (1-7) is managed by a corresponding supervisor task (9-15). The supervisor tasks are managed by a kernel task (8) that handles overall resource management and communication. Task 0 is not used since its trap structure is asymmetric (its "supervisor" is the kernel). Although task 8 traps to task 8, this is a fatal error, since the trap leaves the trapping task dormant, and the trap-handling process cannot progress.

CHAPTER 4 - PRIVILEGED HEP OPERATION

CHAPTER 4

PRIVILEGED HEP OPERATION

Instructions and facilities described in this section are available only to supervisor processes; they control the hardware mechanisms for task and process management, input/output, and system initialization, configuration, and maintenance.

4.1 TASK MANAGEMENT

Each task in HEP is allocated regions in program, register, constant and data memory. The regions are defined by a 128-bit Task Status Word (TSW) associated with the task. The instructions RTSW and WTSW, which allow supervisor processes to read and write TSWs, treat the TSW as two independent, 64-bit words (as shown in Figure 4.1.1).

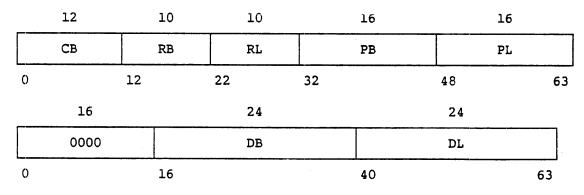


Figure 4.1.1 - TSW

The fields of these two words are used to compute a base and limit address for each memory type except constant memory, which has no limit address. Base and limit addresses are formed by appending an appropri-

CHAPTER 4 - PRIVILEGED HEP OPERATION

ate number of zero bits to the right of the base and the same number of one bits to the right of the limit. Table 4.1.1 shows the number of bits appended to each field. In data memory, base and limit addresses are byte addresses; in other memories they are word addresses.

TSW	FIELD	FIELD	NO. OF BITS	BIT VALUE	ADDRESS
FIELD	LENGTH	NAME	APPENDED	APPENDED	LENGTH
СВ	12	Constant Memory Base	1	0	· 13
RB	10	Register Memory Base	1	0	11
RL	10	Register Memory Limit	1	1	11
PB	16	Program Memory Base	4	0	20
PL	16	Program Memory Limit	4	1	20
DB	24	Data Memory Base	8	0 -	32
DL	24	Data Memory Limit	8	1	32

Table 4.1.1 - Base and Limit Address Formation

The base address is added to each effective address generated by a process to yield a real (physical) memory address; if a carry occurs a protection violation trap is generated. The real address is in turn compared to the appropriate limit address. A real address greater than the limit generates a protection violation trap. A base address greater than the limit denies the task access to the associated memory type.

The exception to this scheme is constant memory. Since the constant memory base address is 13 bits long and there is no CL field, a task can access any constant memory location with a real address greater than or equal to the base. (The effect of reading a location with a real address greater than 2**12-1, however, is undefined.) In addition, when a real constant memory address is calculated, a carry does not generate a trap.

4.1.1 PROGRAM AND CONSTANT MEMORY ACCESS

Since user processes may not write either program memory or constant memory, supervisor processes must perform these functions. Supervisor processes may write constant memory via any instruction with a D field; they may read and write program memory via the instructions RPM and WPM. Base and limit addresses are not applied to the program memory address arguments of RPM and WPM.

CHAPTER 4 - PRIVILEGED HEP OPERATION

4.1.2 DATA MEMORY ADDRESSING

Data memory effective address space is split. Effective addresses in the lower half (byte locations 0 to 2**31-1) are based and limited by DB and DL, but effective addresses in the upper half (byte locations 2**31 to 2**32-1) are not. Thus the upper half of the space is identical to real address space and may only be accessed by supervisor processes. It contains locations associated with input-output devices, the I/O cache, and system support processor interfaces. A user process can address these locations only if DL is sufficiently large (greater than 800000 hexadecimal) to include some of them.

The uppermost effective addresses allow supervisor processes read-only access to the lower 2**28 bytes of real data memory address space by specifying effective addresses in the range 2**32-2**28 to 2**32-1. That is, the last sixteenth of the effective address space addresses the same locations as the first sixteenth of the real address space.

4.1.3 TASK STATES

Tasks may be live or dead, and active or dormant. A task that has no processes associated with it is said to be dead; such a task can become live only by having processes created in it. This can be done by another task, via a trap, or by a supervisor process, via a Privileged Create (PCR) instruction. A live task dies in one of two ways: either all of its processes execute quit (QT or SQT) instructions or a supervisor process executes a KILL instruction for the task.

Independent of whether it is live or dead, a task may also be active or dormant. Processes in an active task are allowed to execute instructions normally, but processes in a dormant task are prevented from executing. The instructions ACT (Activate) and DACT (Deactivate) make a task active or dormant, respectively. In addition, a trap causes the task responsible to become dormant. The states of all tasks are available through the RCTL (Read CFU Control) instruction.

4.2 PROCESS SCHEDULING

A HEP PEM automatically schedules processes for execution by inserting and removing Process Tags (PTs) in one of 16 task queues. Each PT is the address of a PSW in the "process queue" (which is not, in fact, a queue but a 128-location memory).

CHAPTER 4 - PRIVILEGED HEP OPERATION

When a task is to be made dormant, it is only necessary for the hardware to inhibit the removal of PTs from the associated queue. In creating a process, the hardware automatically allocates a PT value not currently in use, writes the desired PSW information at the process queue location pointed to by that PT, and inserts the PT in the appropriate task queue. When a process quits, the PT value is made available for the creation of other processes and is not reinserted in the queue.

The PEM logic associated with the task queues selects PTs from active, live tasks in a round-robin fashion, thereby ensuring that each task obtains a fair share of the execution resources. When a PT is selected, it is removed from its queue and used to select a PSW from the process queue. The task queue that was used determines the TSW, and the PSW and TSW together determine the next instruction to be fetched and its real source and destination addresses.

For all but SFU and quit instructions, each PT is reinserted in its task queue only after a fixed delay. The delay is such that a minimum of eight active processes are required to avoid the wasted cycles that occur when all task queues of active tasks are empty. The delay ensures that a function unit has eight cycles to compute and store a result in register or constant memory before the next instruction accesses the result.

Two function units, the Divider and the SFU, do not compute results in eight cycles. The Divider needs 16 or 17 cycles to complete. If the destination of an FDIV instruction is in register memory, the destination location remains reserved until the result is written. This prevents any use of the value until it is available. In constant memory, however, it is possible for a supervisor process to access a location that does not yet contain the value from a previous FDIV. For this reason supervisor process programs must have at least two instructions between an FDIV with a constant-memory destination and an instruction that uses the result of that FDIV.

The SFU instructions (LOD, LODA, LODI, LODX, RSSW, STO, STOI, STOX, and WSSW) require an indeterminate amount of time to complete. When one of these instructions is executed, the PT is not reinserted in its task queue but is instead sent to the SFU. The SFU also contains 16 queues for PTs, one for each task, and an SFU queue addressed by those PTs. The SFU queue holds 128-bit SFU Status Words (SSWs). Each SSW contains enough information about a pending data memory reference instruction to allow its re-execution by the SFU when the access state of the specified location does not satisfy the access control specified in the instruction.

CHAPTER 4 - PRIVILEGED HEP OPERATION

The PT associated with the process executing the SFU instruction remains in the SFU until the instruction is successful. When this occurs, the PT is reinserted in its task queue, removed, and the next instruction is fetched. By this time, the result of the SFU instruction has been written in register or constant memory and is available to the next instruction. If the SFU generates a trap, the PT of the process generating the trap remains in its queue in the SFU.

4.3 PROCESS STATUS

Supervisor processes can read and write the PSW or SSW corresponding to a given PT using the instruction RPSW, WPSW, RSSW, or WSSW. Since a PSW or SSW may be modified by instructions executed by the process it describes, care should be taken in the use of these instructions on processes belonging to an active task. None of these instructions actually creates a PT for a process; the only mechanisms that can create a PT are create instructions and traps.

4.3.1 THE PROCESS STATUS (PS) FIELD

Bits 0-7 of the PSW -- the "PS field" -- can be read and written only by supervisor processes; the contents of PS are shown in Figure 4.3.1 and described in Table 4.3.1.

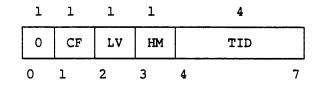


Figure 4.3.1 - PS Field of the PSW

CHAPTER 4 - PRIVILEGED HEP OPERATION

BITS	NAME	DESCRIPTION
0	-	Must be zero.
1	CF	Create Fault enable. When set, allows this process to run in create fault mode.
2	ΓΛ	Live. When set, a PT has been allocated for this process.
3	НМ	Hardware Maintenance. When set, certain trap and wave off conditions have no effect on this process.
4-7	TID	Task Identifier. Used by PCR to specify the task into which a process is created, and to indicate the task to which a process belongs when the PSW is read.

Table 4.3.1 - PS Field Contents

The TID field contains the task identifier and indicates which task the process belongs to. When a PCR instruction is executed, the TID field of the PSW operand specifies the task queue into which the newly allocated PT is to be inserted. Never change the value of TID for a live process (a process having LV=1); the task queue to which the process belongs would disagree with its TID.

The CF bit is set to allow the process to execute in create fault mode. When too many processes are created, a create fault trap occurs and the PEM enters user or supervisor create fault mode, depending on the task that caused the problem. In create fault mode, only processes with a CF value of 1 are allowed to execute; other processes are waved off. The process created by a create fault trap executes in Task 8 and has the CF bit set. The PCR instruction may also be used to create processes with CF set (or reset) in arbitrary tasks.

Although a create fault occurs when the number of live user or supervisor processes exceeds 56, there may be a greater number of live processes of either type when the trap is complete. For this reason, a process handling a supervisor create fault must not create other supervisor processes until it has verified that there are less than 64 PT values in use. Note also that unlike other traps, create fault does not make the trapping task dormant. Whether or not a PEM is in a create fault mode can be discovered using the RCTL instruction, and either kind of create fault mode can be reset using the RCF instruction.

The LV bit is a read-only bit that, when set, asserts that the process

CHAPTER 4 - PRIVILEGED HEP OPERATION

is live, i.e., a PT has been allocated to it. If LV is zero, the PSW does not correspond to an executing process. LV is set when a process is created, and reset when the process quits or when the task is killed.

The HM bit is primarily used for hardware maintenance; when it is set, the process will not trap for any reason unless a create fault trap occurs, create fault mode is in effect, or the trap is due to the SFU. In addition, whether a SFLG or WFLG instruction cause a wait is independent of the state of HM. Processes created by an Initial Process Load (IPL) trap have the HM bit set, and PCR may be used to create processes with HM set (or reset) in arbitrary tasks.

4.3.2 THE SFU STATUS WORD

The SSW format is shown in Figure 4.3.2.

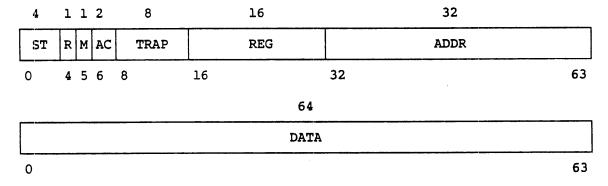


Figure 4.3.2 - SFU Status Word

The instructions RSSW and WSSW read and write the SSW as two independent 64-bit words. The fields of the SSW are described in Table 4.3.2.

CHAPTER 4 - PRIVILEGED HEP OPERATION

BITS	NAME	DESCRIPTION
0-3	ST	State. Describes the SFU operation in progress. If ST>=8, a trap occurs.
4	R	Read. When set, a LOD, LODA, or LODX instruction was executed.
5	М	Address Mode. When set, a byte operation is to be performed.
6-7	AC	Access Control. Corresponds to SAC if R=1, DAC if R=0.
8-15	TRAP	Trap information. Contains either the UTM from the PSW if ST<8 or information describing the trap if ST>=8.
16-31	REG	Register or Constant Address. Contains the real address computed from the D or SO field of the SFU instruction. The location addressed is read when ST=6 and written when ST=2 or 5. This field also contains information about partial word justification and access state.
32-63	ADDR	Data Memory Address. The real address of the location where the operation is to be performed.
0-63	DATA	Data. The data to be moved by the operation.

Table 4.3.2 - SSW Contents

The ST field of the SSW describes the state of the SFU operation in progress. The meanings of these states are summarized in Figure 4.3.3. The idle state (ST=0) indicates that no operation is in progress in the When an SFU instruction other than RSSW or WSSW is executed, the ST field changes value to 1 for LOD, LODI, and LODX, 2 for LODA, 5 for STO and STOI, 6 for STOX, and 8 if a data memory protection violation trap is to be generated. When ST=1, a data memory read operation is attempted. If the access control requirements are not met by the access state of the addressed location, the ST field (in fact, the entire SSW) is left unchanged. If an uncorrectable error occurs in the location or if an unimplemented address was specified, ST changes to 9. Finally, if the data memory read succeeds, the ST value changes to 2 and the data just read are available in the data field of the SSW so they can be written in register or constant memory. Other state sequences perform similar operations for the other instructions. The contents of source 0 (SO) of the STOX instruction is read into the SSW when ST is 6; a data

CHAPTER 4 - PRIVILEGED HEP OPERATION

quality in SO corresponding to a non-zero bit in the UTM will cause a data exception trap.

Values of ST not described in Figure 4.3.3 generate traps if ST>=8 and are forced to 0 if ST<8. In addition, when a task is killed, all ST values for SSWs allocated to that task are forced to 0 and the PTs are reinserted in the task queue so that they may be deallocated as they are removed.

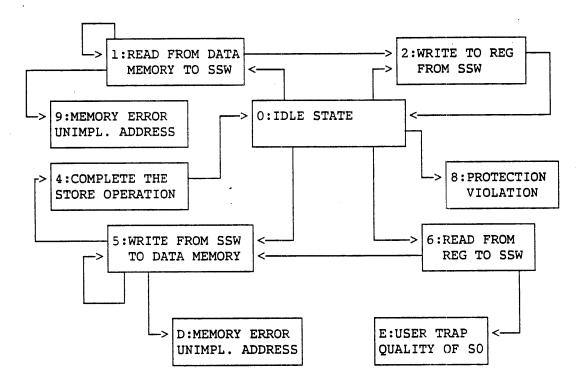


Figure 4.3.3 - SFU State

The R, AM, and AC fields contain further information about the operation to be performed by the SFU. The R (read) bit is l if a LOD, LODI, or LODX is to be attempted and is 0 for LODA, STO, STOI, and STOX. M is the address mode bit from the data memory address, and AC equals SAC if R=1 and DAC if R=0, where SAC and DAC also come from the data memory address.

The trap field contains the UTM if ST is 6, the data quality causing the data exception trap if ST is E, and information about data memory errors or protection violations if ST is 9 or D. For other values of ST, the contents of the trap field are undefined. These facts are summarized in Table 4.3.3.

CHAPTER 4 - PRIVILEGED HEP OPERATION

				SSW	BIT			
ST	8	9	10	11	12	13	14	15
0				UNDE	FINED			
1				UNDE	FINED			
2				UNDE	FINED			
4				UNDE	FINED			
5				UNDE	FINED			
6	UTM							
8	UNDEFINED							
9	ואט	DEFI	NED		DE	0	SE	INV
D	מט	DEFI	NED	•	DE	0	SE	INV
E	0	0	0	0	0		DQ	

- DQ Data quality of SO causing a data exception trap.
- DE Double error in data memory causing a
 memory error trap.
- SE Single error in data memory (no trap).
- INV- Invalid address causing a protection
 violation (unless DE is also set).

Table 4.3.3 - SSW Trap Subfields

The REG field contains a real address in register or constant memory which is used by STOX and by LOD, LODI, and LODX. It also contains bits relating to partial word justification and access state. The REG field is described in Figure 4.3.4 and Table 4.3.4.

CHAPTER 4 - PRIVILEGED HEP OPERATION

1 .	1	1	12	1
AC	С	LJ	RADR	SX
16	17	18	19	31

Figure 4.3.4 - SSW REG Field Format

- AC Access Control Bit. If the instruction was LOD, LODI, or LODX, AC is set if bits 8 and 9 of the instruction were both set. For STO, STOI, STOX, and LODA, AC is set if the access state of the register memory location specified by the D (or SO) field of the instructions was empty before the instruction set it reserved.
- C Constant Bit. Agrees with bit 18 of the instruction.
- LJ Left Justify Bit. Set if bits 8 and 9 of the instruction contain 1 and 0 respectively.

RADR The real register or constant memory address.

SX Sign Extended Bit. Agrees with bit 9 of the instruction except for LODA; for LODA, SX is always zero.

Table 4.3.4 - SSW REG Subfields

The ADDR field of the SSW contains the real data memory address or I/O address, and the DATA field contains the information to be loaded or stored. The LODA instruction generates a DATA field that is in all respects ready to write in the destination with ST=2. For LOD, LODI, and LODX, the DATA field is correctly justified in ST=2 but has not been extended. The DATA field contents when ST is 5 or 4 is the unmodified value from the register or constant memory location to be stored by the STO, STOI, or STOX instruction. For other values of ST, the DATA field is undefined.

CHAPTER 4 - PRIVILEGED HEP OPERATION

4.4 TRAPS

Two traps discussed in the preceding section were not described in Section 2.7. These traps are the initial program load trap and the create fault trap. Figure 4.4.1 shows the format of the trap PSW, and Table 4.4.1 describes each field.

1	1	1	1	4	4	8	12	8	4	20	
	LV	CF	нм	TID	TTID	TPT	TC	UTM	0	PC	
_	1	2	3	4	8	12	20	32	40	44	63

Figure 4.4.1 - Trap PSW Format

LV Set.

CF Set if a create fault trap occurred.

HM Set if an IPL trap occurred.

TID Set to 8 if a create fault trap or an IPL trap occurred; set in accordance with Table 4.4.2 otherwise.

TTID Set to zero for an IPL trap; set to the TID of the trapping task otherwise. Except for create fault traps, this task is made dormant.

TPT Undefined for an IPL trap; set to the PSW and SSW address (i.e., the PT) of the process causing the trap otherwise.

TC Set in accordance with Table 4.4.3 if a memory error, illegal instruction, or protection violation occurred. Set in accordance with Table 4.4.4 if a data exception trap occurred. Set to the contents of the D field of the instruction if an SVC occurred. Set to zero otherwise.

UTM Set to zero.

PC Set in accordance with Table 4.4.5.

Table 4.4.1 - Trap PSW Fields

CHAPTER 4 - PRIVILEGED HEP OPERATION

TRAPPING TASK TRAP HANDLING TASK

0 .	8
1	9
2	10
3	11
4	12
5	13
6	14
7	15
8	8
9	8
10	8
11	8
12	. 8
13	8
14	8
15	8

Table 4.4.2 - Trap-Handling Tasks

TC	BIT	PSW BIT	MEANING IF SET
	0	20	Data memory uncorrectable error or protection violation.
	1-5	21-25	(unused)
	6	26	Unimplemented data memory location.
	7	27	Data memory protection violation.
	8	28	D or SO protection violation.
	9	29	S1 protection violation.
	10	30	S2 protection violation.
	11	31	Program memory protection violation.

Table 4.4.3 - Trap Codes

CHAPTER 4 - PRIVILEGED HEP OPERATION

TC BIT	PSW BIT	MEANING IF SET
0	20	If set, trap was due to SO (STOX only)
1-3	21-23	Data quality from SO, if relevant
4	24	If set, trap was due to Sl
5-7	25-27	Data Quality from Sl, if relevant
8	28	If set, trap was due to S2
9-11	29-31	Data quality from S2, if relevant

Table 4.4.4 - Data Exception Trap Codes

PC	MEANING
00001	IPL - An initial program load was generated externally.
00010	Memory error - ECC hardware detected an uncorrectable error in a program memory or data memory location.
00012	Illegal Instruction - The opcode of the instruction is not in the HEP instruction set.
00014	Protection Violation - An illegal memory access was attempted.
00016	Privileged Instruction - A user process attempted to execute a privileged instruction.
00018	Create Fault - Too many processes were created in user tasks or supervisor tasks.
0001A	Supervisor Call - The operating system is to perform some function on behalf of the user.
0001C	Simulated Instruction - The opcode of the instruction is to be interpreted and carried out by software.
0001E	Data Exception - A source data quality specified by the user's UTM was encountered.
	Table 4.4.5 - Trap PSW Program Counter

The program counter of the PSW that caused the trap is still pointing to

CHAPTER 4 - PRIVILEGED HEP OPERATION

the instruction that caused the trap unless either the most significant bit of the TC field in the trap PSW (PSW bit 20) is set or unless a create fault trap occurred. In these latter cases, the PC of the PSW that caused the trap has been incremented to point to the next instruction. If a create fault trap occurs, the PSW addressed by TPT is the PSW that executed the create instruction or caused that trap that generated the create fault, not the PSW created thereby.

4.5 MAINTENANCE SUPPORT

PEMs communicate with the Diagnostic and Maintenance Processor (DMP) using program memory, since both the PEM and the DMP can read and write data or programs there. Each PEM has 64 communications flags and 64 enable bits to synchronize these operations. If a flag is set and its corresponding enable bit is also set, an interrupt is generated to the DMP. Similarly, the WFLG instruction allows a process in the PEM to wave off until a particular flag is cleared (irrespective of the enable bit setting).

The DMP can only clear flags, while the PEM can only set them (using SFLG). The enable bits are only modifiable from the DMP, but flags and enable bits can be read either by the DMP or by the PEM (using RFLG and RNABL).

The WFLG, SFLG, RFLG, and RNABL instructions are supported by the Hardware Access (HA) function unit, and do not require that the PEM be connected to the rest of the system. This feature is useful when a PEM must be isolated for preventive maintenance.

The number of single or double errors found by the program memory error correction hardware is logged in a counter for each PEM. Instructions are provided to read this counter (using RECC) and clear it (using CLRECC). The counter helps gather information about error rates so that timely maintenance can be performed on program memory to keep its reliability high.

-			

CHAPTER 5 - INSTRUCTIONS

CHAPTER 5

INSTRUCTIONS

This chapter discusses each HEP PEM instruction individually. Each instruction description includes the assembler statement format for the instruction, the internal structure of the instruction, a description of the operation formed by the instruction, a description of exception conditions which may occur during execution, and the function unit used to execute the instruction.

The instruction descriptions are presented in alphabetic sequence with respect to the assembler mnemonic.

5.1 INTEGER ABSOLUTE VALUE

ASSEMBLER: ABS <D>, <S1>

FUNCTION UNIT: IFU

	16	16	16	16	
	610B	В	D	Sl	
0		16	32	48 63	•

DESCRIPTION: The contents of D are replaced by the integer absolute

value of the contents of S1.

EXCEPTIONS: Overflow

Overflow is set when S1 contains -2**63. The result is

-2**63.

5.2 ACTIVATE TASK (PRIVILEGED)

ASSEMBLER: ACT <S2>

FUNCTION UNIT: CFU

	16	16	16	16
	9901	0000	0000	S2
0		16	32	48 . 63

DESCRIPTION: The task specified by the integer contents of S2 is acti-

vated, allowing processes in that task to execute instruc-

tions.

5.3 INTEGER ADD

ASSEMBLER: ADD <D>, <S1>, <S2> FUNCTION UNIT: IFU

	16 .	16	16	16
	7126	D	S1	S 2
_) .	16	32	48 63

DESCRIPTION: The contents of D are replaced by the integer sum of the

contents of S1 and S2.

EXCEPTIONS: Carry; overflow

Carry is set if there was a carry out of the most significant bit of the result and overflow does not occur.

Overflow is set if both operands are of the same sign and the result is of the opposite sign. The resulting integer is too small by 2**64 if its sign is negative and too large by 2**64 if its sign is positive.

5.4 LOGICAL PRODUCT

ASSEMBLER: AND <D>, <S1>, <S2> FUNCTION UNIT: IFU

16	16	16	16	
711E	D	Sl	S2	
0	16	32	48 6	 i3

DESCRIPTION: The contents of D are replaced by the bit vector 'and' of

the contents of S1 and S2. A bit position in the result is set to one if the corresponding bit positions of both S1 and S2 contain a one; otherwise, the result bit is set

to zero.

E	BEQ BGE BGT BLE		HEP PRINCIPLES OF OPERATION CHAPTER 5 - INSTRUCTIONS 5.5 BRANCH				
	BLT BNE		5.5	BKANCH		BLT BNE	
F	ASSEMBLER:	B <	<addr>,,<s1> <addr>,<s2>,<s1></s1></s2></addr></s1></addr>		FUNCTION UNIT:	CFU	
	8	4	20	16	16		
	38		ADDR	51	S2		

32

8

12

DESCRIPTION: The contents of S2 are compared with zero. If the condition specified by <c> is true, the program counter of the executing PSW is set to ADDR. Additionally, the executing PSW is modified, field by field, by adding the corresponding fields of the contents of Sl modulo the length of each This is done for the CI, RI, UTM and PC fields. The PS field is not modified. If field modification is not desired, S1 must be the address of a constant integer 0. Note the reversal of <S2> and <S1>.

48

63

The possible test modes and the corresponding values for C are:

B <c></c>	•	С
	(never)	0
BLT	(less than)	1
BEQ	(equal to)	2
BLE	(less than or equal to)	3
BGT	(greater than)	4
BNE	(not equal to)	5
BGE	(greater than or equal to)	6
В	(always)	7
BE	(empty)	8
BF	(full)	9
	(RI equal to zero)	A
	(RI not equal to zero)	В
	(CI equal to zero)	C
	(CI not equal to zero)	D
	(even parity)	E
	(odd parity)	F

5.6 BIT CLEAR

ASSEMBLER: BITCLR <D>, <S1>, <S2> FUNCTION UNIT: HA

	16	16	16	16	
	7728	D	Sl	S2	
0		16	32	48 6	3

DESCRIPTION: The contents of D are replaced by the result obtained by

setting a zero in the contents of S1 in the bit position specified by the low order six bits of the contents of S2,

i.e., the contents of S2 modulo 64.

5.7 BIT SET

ASSEMBLER: BITSET <0>,<S1>,<S2> FUNCTION UNIT: HA

16	16	16	1,6	_
7720	D	Sl	S2	
0	16	32	48 63	3

DESCRIPTION: The contents of D are replaced by the result obtained by

setting a one in the contents of S1 in the bit position specified by the low order six bits of the contents of S2,

i.e., the contents of S2 modulo 64.

CALL	HEP PRINCIPLES OF OPERATION	CALL
CALLEQ		CALLEQ
CALLGE	CHAPTER 5 - INSTRUCTIONS	CALLGE
CALLGT		CALLGT
CALLLE		CALLLE
CALLLT	5.8 CALL	CALLLT
CALLNE		CALLNE

ASSEMBLER: CALL <D>,<S1>

CALL<c> <D>,<S1>,<S2>

8	4	4	16	16	16	
79	С	1	D	Sl	S2	
0	8	12	16	32	48	63

DESCRIPTION: The contents of D are replaced by the executing PSW, with the PS field set to zero and the PC field incremented to point to the next instruction. In addition, the contents of S2 are compared with zero. If the condition specified by <c> is true, the PSW in S1 replaces the PC field of the executing PSW. CALL is equivalent to XPSW with an <action code list> of RPC.

The possible test modes and the corresponding values for C are:

CALL <c></c>	•	С
	(never)	0
CALLLT	(less than)	1
CALLEQ	(equal to)	2
CALLLE	(less than or equal to)	3
CALLGT	(greater than)	4
CALLNE	(not equal to)	5
CALLGE	(greater than or equal to>	6
CALL	(always)	7
CALLE	(empty)	8
CALLF	(full)	9
	(RI equal to zero)	A
	(RI not equal to zero)	В
	(CI equal to zero)	C
	(CI not equal to zero)	D
	(even parity)	E
	(odd parity)	F

EXCEPTIONS: None.

FUNCTION UNIT: CFU

5.9 CLEAR

ASSEMBLER: CLR <D>

FUNCTION UNIT: IFU

16	16	16	16	
411C	D	0000	0000	
0	16	32	48 6	53

DESCRIPTION: The contents of D are replaced by all zeros.

5.10 CLEAR ECC COUNTER (PRIVILEGED)

ASSEMBLER: CLRECC

FUNCTION UNIT: HA

	16	16	16	16	
	8701	0000	0000	0000	
0		16	32	48	63

DESCRIPTION: This instruction clears the ECC Counter (See RECC).

EXCEPTIONS: None.

5-11

5.11 CLEAR TASK COUNTER (PRIVILEGED)

ASSEMBLER: CLTC <S1>,<S2> FUNCTION UNIT: SPI

16	16	16	16	
3601	0000	Sl	S2	
0	16	32	48 6	3

DESCRIPTION: The task counter specified by the contents of S1 for the task number specified by the integer contents of S2 is set to zero. The task counter determined by the contents of S1 is given below.

- Sl Task Counter
 - 0 wave off counter
 - floating-point instruction counter
 - 2 data memory instruction counter
 - 3 other instruction counter

Floating-point instructions are those whose mnemonics begin with the letter 'F'. Data memory instructions are LOD, LODI, LODX, STO, STOI and STOX. The sum of counters 1, 2 and 3 is the number of instructions executed by the task since the last time the counters were cleared. (See RDCT, STCT.)

HEP PRINCIPLES OF OPERATION	CR
	CREQ
CHAPTER 5 - INSTRUCTIONS	CRGE
	CRGT
	CRLE
5.12 CREATE	CRLT
_	CRNE
	CHAPTER 5 - INSTRUCTIONS

ASSEMBLER: CR <S1>

FUNCTION UNIT: CFU

CR<c> <S1>,<S2>

8	4	4	16	16	16	
39	С	5	0000	S1	S2	
0	8	12	16	32	48 63	•

DESCRIPTION: The contents of S2 are compared with zero. If the condition specified by <c> is true, the PS, CI, and UTM fields from the executing PSW are combined with the RI and PC fields of the PSW in Sl to form a new PSW. This PSW is placed in the first available location in the process queue, creating a process with the new PSW. If the process queue location used is the last available location, the create fault condition is raised in the PEM and a trap occurs. CR is equivalent to CREATE with an <action code list> of RRI, RPC.

The test modes and the corresponding values for C are:

CR <c></c>	•	С
CDI M	(never)	0
	(less than)	1
_	(equal to)	2
CRLE	(less than or equal to)	3
CRGT	(greater than)	4
CRNE	(not equal to)	5
CRGE	(greater than or equal to>	6
CR	(always)	7
CRE	(empty)	8
CRF	(full)	9
	(RI equal to zero)	A
	(RI not equal to zero)	В
	(CI equal to zero)	С
	(CI not equal to zero)	D
	(even parity)	E
	(odd parity)	F

EXCEPTIONS: Create fault.

CREATE	HEP PRINCIPLES OF OPERATION	CREATE
CREATEGE		CREATEGE
CREATEGT	CHAPTER 5 - INSTRUCTIONS	CREATEGT
CREATELE		CREATELE
CREATELT		CREATELT
CDFATFNT	5.13 CREATE PROCESS	CREATENE

ASSEMBLER: CREATE <Sl>(<action code list>) FUNCTION UNIT: CFU CREATE<c> <Sl>(<action code list>),<S2>

8	4	4	16	16	16
39	С	ACT	0000	Sl	S2

32

0

8

12

16

DESCRIPTION: The contents of S2 are compared with zero. If the condition specified by <c> is true, the fields of Sl, a PSW, or fields of the executing PSW are selected according to the action to form a new PSW. This PSW is placed in the first available location in the process queue, creating a process associated with the new PSW. The PS (privileged) field of the PSW created is the same as the currently executing PSW. The <action code list> specifies ACT, and consists of a sequence of <action code> items separated by commas. The meaning of each <action code> and its corresponding ACT value is given below; when more than one <action code> is specified, ACT values are additive. If the process queue location used by the create is the last available location, the create fault condition is raised in the PEM and a trap occurs.

48

63

<action code=""></action>	ACT	MODIFICATION BY S1
RCI	8	REPLACE CI
RRI	4	REPLACE RI
RUTM	2	REPLACE UTM
RPC	1	REPLACE PC

For fields other than PC, if no <action code> is specified the PSW of the created process matches the PSW of the executing process in that field. If RPC is not specified, the PC of the executing process is incremented to form the PC of the created process.

The possible test modes and the corresponding values for C are:

CREATE CREATEGE CREATEGT CREATELE CREATELT CREATENE	HEP PRINCIPLES CHAPTER 5 -	OF OPERATION INSTRUCTIONS	CREATE CREATEGE CREATELE CREATELT CREATENE
	CREATE <c></c>	•	С
		(never)	0
	CREATELT	(less than)	1
		(equal to)	2
		(less than or equal to)	3
		(greater than)	4
		(not equal to)	5
		(greater than or equal to>	6
	CREATE	(always)	7
	CREATEE	(empty)	8
	CREATEF	(full)	9
		(RI equal to zero)	A
		(RI not equal to zero)	В
		(CI equal to zero)	С
		(CI not equal to zero)	D
		(even parity)	E
		(odd parity)	F

EXCEPTIONS: Create fault.

5.14 DEACTIVATE TASK (PRIVILEGED)

ASSEMBLER: DACT <S2>

FUNCTION UNIT: CFU

	16	16	16	16	
	9902	0000	0000	S2 ·	
0		16	32	48	63

DESCRIPTION: The task specified by the integer contents of S2 is deac-

tivated, preventing processes in that task from executing

instructions.

5.15 INTEGER DECREMENT

ASSEMBLER: DEC <D>, <S1> FUNCTION UNIT: IFU

	16	16	16	16
	612F	D	Sl	0000
0		16	32	48 63

DESCRIPTION: The contents of D are replaced by the integer result obta-

ined by subtracting an integer one (1) from the contents

of Sl.

EXCEPTIONS: Carry, overflow

Carry is set if there was no borrow out of the most significant bit of the result and overflow does not occur. This will be the case if S1 does not contain either zero (causing borrow) or -2**63 (causing overflow).

Overflow is set when S1 contains -2**63. The result is 2**63-1.

5.16 BIT ENCODE, LEFTMOST 0

ASSEMBLER: ENCODEO <D>, <S1> FUNCTION UNIT: HA

16	16	16	16	
6718	D	Sl	0000	
0	16	32	48	63

DESCRIPTION: The contents of D are replaced by the bit position number

(0=leftmost bit) of the leftmost zero bit in the contents of Sl. If there are no zeros in Sl, the contents of D are

set to the negative integer -2**63.

5.17 BIT ENCODE, LEFTMOST 1

ASSEMBLER: ENCODE1 <D>, <S1> FUNCTION UNIT: HA

	16	16	16	16	
	6710	D	Sl	0000	
0		16	32	48	63

DESCRIPTION: The contents of D are replaced by the bit position number

(0=leftmost bit) of the leftmost one bit in the contents of Sl. If there are no ones in Sl, the contents of D are

set to the negative integer -2**63.

5.18 LOGICAL DIFFERENCE

ASSEMBLER: EOR <D>, <S1>, <S2> FUNCTION UNIT: IFU

_	16	16	16	16	
	7119	D	Sl	S 2	
•	0	16	32	48	63

DESCRIPTION: The contents of D are replaced by the bit vector 'exclusive or' of the contents of S1 and S2. A bit position in the result is set to one if the corresponding bit positions of either S1 or S2 but not both contain a one;

otherwise, the result bit is set to zero.

5.19 EQUIVALENCE

ASSEMBLER: EQV <D>, <S1>, <S2> FUNCTION UNIT: IFU

	16	16	16	16
	7116	D	S1	S2
0)	16	32	48 63

DESCRIPTION: The contents of D are replaced by the bit vector 'equivalence' of the contents of S1 and S2. A bit position in the result is set to one if the corresponding bit positions of S1 and S2 are either both zero or both one;

otherwise, the result bit is set to zero.

5.20 FLOATING ADD

FUNCTION UNIT: ADDER ASSEMBLER: FADD <D>,<S1>,<S2>

	16	16	16	16
	7200	D	Sl	S2
0		16	32	48 63

DESCRIPTION: The contents of D are replaced by the normalized sum of the floating-point contents of S1 and S2. Each 14-hexadecimal digit mantissa is expanded to 16 digits by placing a guard digit to the right and a carry digit to the left. The mantissa corresponding to the smaller exponent is shifted right, discarding rightmost digits, by an amount equal to the difference in exponent magnitudes. The two mantissas are added or subtracted as required by their signs and the 16-digit result is normalized, correcting the exponent as required, or clearing it if the result is zero. Unnormalized floating-point numbers may be normalized by adding zero to them. The 16-digit result mantissa is shifted left by the number of leading hexadecimal zeros and truncated on the right to 14 hexadecimal digits.

EXCEPTIONS: Overflow, underflow.

Overflow is set if the exponent of the normalized result is greater than 63; the result is set to maximum magnitude ((16**63)*(1-16**-14)) with the proper sign.

Underflow is set if the exponent of the normalized result is less than -64 and the result is non-zero; the result is set to zero.

5.21 FLOATING DIVIDE

ASSEMBLER: FDIV <D>, <S1>, <S2> FUNCTION UNIT: DIVIDER

	16	16	16	16
	7 F 00	D	Sl	S2
•	0	16	32	48 63

DESCRIPTION: The contents of D are replaced by the normalized floating-point result obtained by dividing the contents of S1 by the contents of S2. If S1 or S2 does not contain a normalized floating-point number the result is undefined. The result mantissa is the most significant 56 bits of the

quotient of the operand mantissas.

EXCEPTIONS: Overflow, indefinite result.

Overflow is set if the exponent of the result is greater than 63 or if the contents of S2 but not S1 are zero; the result is set to maximum magnitude ((16**63)*(1-16**-14)) with the proper sign.

Underflow is set if the exponent of the result is less than -64 and the result is non-zero; the result is set to zero.

Indefinite result is set if both operands are zero; the result is set to zero.

5.22 FLOAT INTEGER PART

ASSEMBLER: FIP <D>, <S1>

FUNCTION UNIT: IFU

	16	16	16	16	
	610A	D	S1	0000	
_)	16	32	48 6	3

DESCRIPTION: The contents of D are replaced by the floating-point re-

presentation of the integer part of the floating-point contents of S1. FIP is equivalent to the sequence FIX,FLT

unless FIX sets overflow.

5.23 CONVERT FLOATING-POINT TO INTEGER

ASSEMBLER: FIX <D>, <S1>

FUNCTION UNIT: IFU

0	16	32	48 63	 }
6108	D	Sl	0000	
16	16	16	16	_

DESCRIPTION: The contents of D are replaced by the integer part of the

floating-point contents of Sl.

EXCEPTIONS: Overflow

Overflow is set if the value of the operand is greater than or equal to $2^{**}63$ or less than $-2^{**}63$. The result is

undefined if overflow is set.

5.24 CONVERT INTEGER TO FLOATING-POINT

ASSEMBLER: FLT <D>,<S1>

FUNCTION UNIT: IFU

	16	16	16	16	
	6109	D	Sl	0000	
0		16	32	48	63

DESCRIPTION: The contents of D are replaced by the floating-point re-

presentation of the integer contents of S1.

EXCEPTIONS: Loss of significance.

Loss of significance is set if the absolute value of the contents of Sl is greater than or equal to 2**56. The result mantissa is the most significant bits of the integer

operand.

FUNCTION UNIT: IFU

CHAPTER 5 - INSTRUCTIONS

5.25 FLOATING MAXIMUM VALUE

ASSEMBLER: FMAX <D>, <S1>, <S2>

	16	16	16	16
	7179	D	Sl	S 2
0		16	32	48 63

DESCRIPTION: The contents of D are replaced by the larger of the two

floating-point contents of S1 and S2. Since this instruction does not normalize either operand and merely performs a 64-bit sign-magnitude comparison, the result may be in-

correct for unnormalized operands.

5.26 FLOATING MINIMUM VALUE

ASSEMBLER: FMIN <D>, <S1>, <S2>

FUNCTION UNIT: IFU

16	16	16	16	
7Í7C	D	Sl	S2	
0	16	32	48 63	3

DESCRIPTION: The contents of D are replaced by the smaller of the two

floating-point contents of S1 and S2. Since this instruction does not normalize either operand and merely performs a 64-bit sign-magnitude comparison, the result may be in-

correct for unnormalized operands.

5.27 FLOATING MULTIPLY

ASSEMBLER: FMUL <D>, <S1>, <S2> FUNCTION UNIT: MULTIPLIER

	16	16	16	16	_
	7300	D	Sl	S2	
0)	16	32	48 63	

DESCRIPTION: The contents of D are replaced by the normalized floating-point product of the contents of S1 and S2. If either operand does not contain a normalized floating-point number the result is undefined. The result mantissa is the most significant 56 bits of the product of

the operand mantissas.

EXCEPTIONS: Overflow, underflow.

Overflow is set if the exponent of the result is greater than 63; the result is set to maximum magnitude ((16**63)*(1-16**14)) with the proper sign.

Underflow is set if the exponent of the result is less than -64 and the result is non-zero; the result is set to zero.

5.28 FLOATING SUBTRACT

ASSEMBLER: FSUB <d>,<s1>,<s2> FUNCTION UNIT: A</s2></s1></d>
--

	16	16	16	16	
	7201	D	Sl	S2	
0		16	32	48 6	3

DESCRIPTION: The contents of D are replaced by the normalized floating-point result obtained by subtracting the contents of S2 from the contents of S1. The contents of S2 are neand a floating-point add is performed. Each 14-hexadecimal digit mantissa is expanded to 16 digits by placing a guard digit to the right and a carry digit to the left. The mantissa corresponding to the smaller exponent is shifted right, discarding rightmost digits, by an amount equal to the difference in exponent magnitudes. The two mantissas are added or subtracted as required by their signs and the 16-digit result is normalized, correcting the exponent as required, or clearing it if the result is zero. Unnormalized floating-point numbers may be normalized by adding zero to them. The 16-digit result mantissa is shifted left by the number of leading hexadecimal zeros and truncated on the right to 14 hexadecimal digits.

EXCEPTIONS:

Overflow, underflow.

Overflow is set if the exponent of the normalized result is greater than 63; the result is set to maximum magnitude ((16**63)*(1-16**-14)) with the proper sign.

Underflow is set if the exponent of the normalized result is less than -64 and the result is non-zero; the result is set to zero.

FTNE	5.29 FLOATING COMPARE, INTEGER RESULT	FTNE
FTLT		FTLT
FTLE		FTLE
FTGT	CHAPTER 5 - INSTRUCTIONS	FTGT
FTGE		FTGE
FTEQ	HEP PRINCIPLES OF OPERATION	FTEQ

ASSEMBLER: FT<C> <D>, <S1>, <S2> FUNCTION UNIT: IFU

8	4	4	16	16	16	
71	4	С	D	S1	52	
0 .	8	12	16	32	48 63	

DESCRIPTION: The floating-point contents of S1 and S2 are compared according to <c>; if the condition is true the contents of D are replaced by positive integer one; if the condition is false the contents of D are replaced by zero.

The possible test modes and the corresponding values for ${\tt C}$ are:

FT <c< th=""><th>•</th><th>С</th></c<>	•	С
FTEQ FTGE FTLT FTNE	<pre>(never) (greater than) (equal to) (greater than or equal to) (less than) (not equal to) (less than or equal to) (always)</pre>	8 9 A B C D E F

FTFEQ	HEP PRINCIPLES OF OPERATION	FTFEQ
FTFGE		FTFGE
FTFGT	CHAPTER 5 - INSTRUCTIONS	FTFGT
FTFLE		FTFLE
FTFLT		FTFLT
FTFNE 5.30	FLOATING COMPARE, FLOATING RESULT	FTFNE

ASSEMBLER: FTF<c> <D>, <S1>, <S2> FUNCTION UNIT: IFU

8	4	4	16	16	16	
71	6	С	D	Sl	S2	
0	8	12	16	32	48	63

DESCRIPTION: The floating-point contents of S1 and S2 are compared according to <c>; if the condition is true the contents of D are replaced by positive floating-point one; if the condition is false the contents of D are replaced by zero.

The possible test modes and the corresponding values for C are:

FTF <c></c>	•	С
FTFEQ FTFGE FTFLT FTFNE	<pre>(never) (greater than) (equal to) (greater than or equal to (less than) (not equal to)</pre>	8 9 A B C D
FTFLE	(less than or equal to) (always)	E F

FTLEQ	HEP PRINCIPLES OF OPERATION	FTLEQ
FTLGE		FTLGE
FTLGT	CHAPTER 5 - INSTRUCTIONS	FTLGT
FTLLE		FTLLE
FTLLT		FTLLT
FTLNE	5.31 FLOATING COMPARE, BIT VECTOR RESULT	FTLNE

ASSEMBLER: FTL		L <c></c>	<d>,<s1>,<s2></s2></s1></d>		FUNCTION UNIT:	IFU	
	8	4	4	16	16	16	
	71	5	С	D	Sl	S2	
	0	8	12	16	32	48	63

DESCRIPTION: The floating-point contents of S1 and S2 are compared according to <c>; if the condition is true the contents of D are replaced by the bit vector having all bits set; if the condition is false the contents of D are replaced by the bit vector with all bits zero.

The possible test modes and the corresponding values for C are:

FTL <c< th=""><th>></th><th>С</th></c<>	>	С
FTLEQ FTLGE FTLLT FTLNE	<pre>(never) (greater than) (equal to) (greater than or equal to) (less than) (not equal to) (less than or equal to)</pre>	8 9 A B C D E
	(always)	F

5.32 IMPLICATION

ASSEMBLER: IMPL <D>, <S1>, <S2> FUNCTION UNIT: IFU

	16	16	16	16	
	7112	D	Sl	S2	\Box
0		16	32	48 6	 i3

DESCRIPTION: The contents of D are replaced by the bit vector 'implication' of the contents of Sl and S2. A bit position in the result is set to one if the corresponding bit position of Sl is zero or the corresponding bit position in S2 is one or both; otherwise, the result bit is set to

zero.

5.33 INTEGER INCREMENT

ASSEMBLER: INC <D>, <S1> FUNCTION UNIT: IFU

	16	16	16	16	
	6130	D	Sl	S2	
0		16	32	48	63

 ${\tt DESCRIPTION:} \quad {\tt The \ contents \ of \ D \ are \ replaced \ by \ the \ integer \ sum \ of \ the}$

contents of S1 and the integer one (1).

EXCEPTIONS: Carry, overflow

Carry is set if there was a carry out of the most significant bit of the result. This will occur when Sl contains

Overflow is set when S1 contains 2**63-1. The result is -2**63.

5.34 INCREMENT ON CARRY

ASSEMBLER: IOC <D>, <S1>, <S2> FUNCTION UNIT: IFU

16 16 16 16

710F D S1 S2

0 16 32 48 63

DESCRIPTION: The contents of D are replaced by the integer sum of the

contents of S1 and the carry bit from the data quality of

S2.

EXCEPTIONS: Carry, overflow

Carry is set if there was a carry out of the most significant bit of the result and overflow does not occur.

Overflow is set when the contents of S1 is 2**63-1 and S2 has carry data quality. The result is -2**63.

JUMP JUMPEQ JUMPGE JUMPGT JUMPLE JUMPLT JUMPNE		JUMP JUMPEQ JUMPGE JUMPGT JUMPLE JUMPLT JUMPNE		
ASSEMBLER:	JUMP <: JUMP <c></c>	S1> <s1>,<s2></s2></s1>		FUNCTION UNIT: CFU
8	4 4	16	16	16

000F

The contents of S2 are compared with zero. If the condi-DESCRIPTION: tion specified by <c> is true, the PC field of the executing PSW is replaced by the PC of the PSW in S1. Jump is equivalent to MOD with an <action code list> of RPC.

32

The possible test modes and the corresponding values for Care:

Sl

JUMP <c></c>	•	С
JUMPLT JUMPEQ JUMPLE JUMPGT JUMPNE	<pre>(never) (less than) (equal to) (less than or equal to) (greater than) (not equal to) (greater than or equal to> (always) (empty) (full) (RI equal to zero) (RI not equal to zero) (CI equal to zero) (CI not equal to zero) (even parity)</pre>	C 0 1 2 3 4 5 6 7 8 9 A B C D E
	(odd parity)	F

EXCEPTIONS: None.

В8

0

C

12

16

8

S2

63

48

5.36 KILL TASK (PRIVILEGED)

ASSEMBLER: KILL <S2>

FUNCTION UNIT: CFU

16	16	16	16 S2	
9900	0000	0000		
0	16	32	48	63

DESCRIPTION: The task specified by the integer contents of S2 is

killed. When the task is active, this results in the destruction of every process which attempts to begin execution of an instruction in the task. The process queue locations used by these processes are available for reuse.

5.37 LOAD DATA MEMORY

ASSEMBLER: LOD <atlist> <D>, <M> FUNCTION UNIT: SFU

8	1	1	1	1	2	2	16	32	
4 C	С	N	R	В	SAC	DAC	D	м	
0	8				12	14	16	32	63

DESCRIPTION: The contents of D are replaced by the contents of the addressed data memory location. The displacement part of the address M and the B, SAC, and DAC parts are obtained from the instruction. The instruction bits C, N, and R are specified by <atlist>, a sequence of <attributes>. Instruction bits are set when their corresponding <attribute> is present, with the following meanings:

BIT OF CORRESPINSTRUCTION ATTRI		MEANING:
8 :0	i a n F	Copy access state. If present and :N is absent, set the access state of D to agree with the access state of the data memory location before the load took place; otherwise set the access state to D to full.
9 :N	r v j E	Numeric. If present and :C is absent, right justify and sign fill partial words loaded into D; if present and :C is present, left justify and zero fill partial words loaded into D; if absent, right justify and zero fill partial words loaded into D.
10 :F	R I	Register address. Must be absent.

5.38 LOAD ADDRESS

ASSEMBLER: LODA <atlist> <D>, <M> FUNCTION UNIT: SFU

8	1 1	1]	L]	. 2	2	16		32	
CC	C	N	R	SAC	DAC	D		М	
0	8			12	14	16	32		63

DESCRIPTION: The contents of D are replaced by the right justified data memory address. The displacement part of the address M and the B, SAC, and DAC parts are obtained from the instruction. The instruction bits C, N, and R are specified by <atlist>, a sequence of <attributes>. Instruction bits are set when their corresponding <attribute> is present, with the following meanings:

BIT OF INSTRUCTION	CORRESPONDING ATTRIBUTE	MEANING:
8	:C	Copy access state. Must be absent.
9	: N	Numeric. If present, the most significant bit of the displacement, M, i.e., the sign of the displacement treated as a 32-bit integer, is extended to the left to fill 64 bits, logically oring with B, SAC, and DAC; if absent, no sign extension occurs.
10	:R	Register address. Must be absent.

5.39 LOAD DATA MEMORY INDIRECT

ASSEMBLER: LODI <atlist> <D>, <S1> FUNCTION UNIT: SFU

8	3	1	1	1	1	2	2	16		16	16	
60	;	С	N	R	В	SAC	DAC	D		S1	0000	
0 .		8				12	14	16	32		48	63

DESCRIPTION: The contents of D are replaced by the contents of the addressed data memory location. The displacement part of the address is the contents of Sl modulo 2**32. The B, SAC, and DAC parts are obtained from the instruction or from the contents of Sl, depending on instruction bit R. Instruction bits C, N, and R are specified by <atlist>, a sequence of <attributes>. Instruction bits are set when their corresponding <attribute> is present, as follows:

BIT OF INSTRUCTION	CORRESPONDING ATTRIBUTE	MEANING:
8	:C	Copy access state. If present and :N is absent, set the access state of D to agree with the access state of the data memory location before the load took place; otherwise set D full.
9	: N	Numeric. If present and :C is absent, right justify and sign fill partial words loaded into D; if present and :C is present, left justify and zero fill partial words loaded into D; if absent, right justify and zero fill partial words loaded into D.
10	:R	Register address. If present, the B, SAC, and DAC subfields are obtained from bits 27 through 31 of the contents of S1; if absent, they are obtained from instruction bits 11 through 15.

5.40 LOAD DATA MEMORY INDEXED

AS:	SEMBLER:	L	מככ	X •	<a1< th=""><th>tlis</th><th>t> •</th><th><d>,<s1>,<s2></s2></s1></d></th><th></th><th>FUNCTION</th><th>UNIT:</th><th>SFU</th></a1<>	tlis	t> •	<d>,<s1>,<s2></s2></s1></d>		FUNCTION	UNIT:	SFU
	8	1	1	1	1	2	2	16	16		16	
	7C	С	N	R	В	SAC	DAC	D	Sl		S2	
	0	8				12	14	16	32	48	63	

DESCRIPTION: The contents of D are replaced by the contents of the addressed data memory location. The displacement part of the address is the sum of the contents of Sl and the contents of S2 modulo 2**32. The B, SAC, and DAC parts are obtained from the instruction or from the contents of Sl, depending on instruction bit R. Instruction bits C, N, and R are specified by <atlist>, a sequence of <attributes>. Instruction bits are set when their corresponding <attribute> is present, as follows:

BIT OF INSTRUCTION	CORRESPONDING ATTRIBUTE	MEANING:
8	:C	Copy access state. If present and :N is absent, set the access state of D to agree with the access state of the data memory location before the load took place; otherwise set D full.
9	: N	Numeric. If present and :C is absent, right justify and sign fill partial words loaded into D; if present and :C is present, left justify and zero fill partial words loaded into D; if absent, right justify and zero fill partial words loaded into D.
10	:R	Register address. If present, the B, SAC, and DAC subfields are obtained from bits 27 through 31 of the contents of S1; if absent, they are obtained from instruction bits 11 through 15.

LPSW	HEP PRINCIPLES OF OPERATION	LPSW
LPSWEQ		LPSWEQ
LPSWGE	CHAPTER 5 - INSTRUCTIONS	LPSWGE
LPSWGT	·	LPSWGT
LPSWLE		LPSWLE
LPSWLT	5.41 LOAD PSW	LPSWLT
LPSWNE		LPSWNE

ASSEMBLER: LPSW <Sl>

FUNCTION UNIT: CFU

LPSW<c> <S1>,<S2>

8	4	4	16	16	10	6
В8	С	0	03FF	Sl		52
0	8	12	16	32	48	63

DESCRIPTION: The contents of S2 are compared with zero. If the condition specified by <c> is true, the executing PSW is replaced by the PSW contained in S1 with the exception of the PS field, which is left unmodified. LPSW Ts equivalent to MOD with an <action code list> of RCI, RPI, RUTM,

RPC.

The possible test modes and the corresponding values for C are:

LPSW <c></c>	•	С
	(never)	0
LPSWLT	(less than)	1
LPSWEQ	(equal to)	2
LPSWLE	(less than or equal to)	3
LPSWGT	(greater than)	4
LPSWNE	(not equal to)	5
LPSWGE	(greater than or equal to>	6
LPSW	(always)	7
LPSWE	(empty)	8
LPSWF	(full)	9
	(RI equal to zero)	A
	(RI not equal to zero)	В
	(CI equal to zero)	С
	(CI not equal to zero)	D
	(even parity)	E
	(odd parity)	F

5.42 INTEGER MAXIMUM VALUE

ASSEMBLER: MAX <D>, <S1>, <S2>

FUNCTION UNIT: IFU

	. 16	16	16	16	
	7171	D	Sl	S 2]
0		16	32	48 63	-

DESCRIPTION: The contents of D are replaced by the larger of the two

integer contents of S1 or S2.

5.43 INTEGER MINIMUM VALUE

ASSEMBLER: MIN <D>, <S1>, <S2>

FUNCTION UNIT: IFU

	16	16	16	16	
	7174	D	Sl	S2	
0		16	32	48	63

DESCRIPTION: The contents of D are replaced by the smaller of the two

integer contents of S1 or S2.

MOD	HEP PRINCIPLES OF OPERATION	MOD
MODEQ		MODEQ
MODGE	CHAPTER 5 - INSTRUCTIONS	MODGE
MODGT		MODGT
MODLE		MODLE
MODLT	5.44 MODIFY PSW	MODLT
MODNE		MODNE

ASSEMBLER: MOD <Sl>(<action code list>)

FUNCTION UNIT: CFU

MOD<c> <S1>(<action code list>),<S2>

	8	4	4	16	16	16	
	В8	С	0	ACT	S1	S2	
_)	8	12	16	32	48	63

DESCRIPTION: The contents of S2 are compared with zero. If the condition specified by <c> is true, the executing PSW is modified, field by field, by the contents of S1 according to

the operations specified by ACT. The contents of S1 are in PSW format. The <action code list> specifies ACT and consists of a sequence of <action code> items separated by commas; each <action code> has a prefix (A, E or R) specifying an action and a suffix (CI, RI, UTM or PC) specifying the affected field. The PS field is not modified. At most one <action code> of a given suffix can appear in

<action code list>. The meaning of each <action code> and its corresponding ACT value is given below; when more than one <action code> is specified, ACT values are addi-

tive.

<action code> ACT MODIFICATION BY S1

ACI	0100	ADD TO CI
ECI	0200	EXCLUSIVE OR TO CI
RCI	0300	REPLACE CI
ARI	0040	ADD TO RI
ERI	0800	EXCLUSIVE OR TO RI
RRI	00C0	REPLACE RI
AUTM	0010	MTU OT DAA
EUTM	0020	EXCLUSIVE OR TO UTM
RUTM	0030	REPLACE UTM
APC	0005	ADD TO PC
EPC	A000	EXCLUSIVE OR TO PC
RPC	000F	REPLACE PC

For fields other than PC, if no <action code> is specified the PSW of the executing process is left unmodified in

MOD	HEP PRINCIPLES OF OPERATION	MOD
MODEQ		MODEQ
MODGE	CHAPTER 5 - INSTRUCTIONS	MODGE
MODGT		MODGT
MODLE		MODLE
MODLT		MODLT
MODNE		MODNE

modifies the PC, the PC field is incremented.

The possible test modes and the corresponding values for ${\tt C}$ are:

<pre>(never) MODLT (less than) MODEQ (equal to) MODLE (less than or equal to) MODGT (greater than) MODRE (not equal to) MODGE (greater than or equal to> 6 MOD (always) MODE (empty) MODF (full) (RI equal to zero) (RI not equal to zero) (CI not equal to zero) (CI not equal to zero) (Even parity) (Odd parity)</pre>

5.45 REGISTER TO REGISTER MOVE

ASSEMBLER: MOV <D>, <S1>

FUNCTION UNIT: IFU

	16	16	16	16	
	611F	D	Sl	ó000	
,	0	16	32	48	63

DESCRIPTION: The contents of D are replaced by the contents of Sl.

5.46 MASK REGISTER DESCRIPTOR

ASSEMBLER: MRD <D>, <S1>, <S2> FUNCTION UNIT: IFU

	16	16	16	16
	710C	ם	Sl	S2
0		16	32	48 63

DESCRIPTION: The contents of D are replaced by the logical 'and' of a register descriptor derived from S2 and the contents of S1 (the mask). The format of the register descriptor is as follows:

DESCRIPTOR BITS	MEANING
0-55	0
56	Access state: empty if 1, full if 0
57-58	0
59	Parity check bit: contents have even parity if 1, odd parity if 0
60	0
61-63	Data quality

5.47 MOVE REGISTER WITH DESCRIPTOR

ASSEMBLER: MRG <D>, <S1>

FUNCTION UNIT: IFU

16	16	16	16	_
610E	D	Sl	0000	
0	16	32	48 63	,

DESCRIPTION: The contents of D are replaced by the contents of S1. access state, parity check bit, and data quality of D are

set equal to those of S1.

5.48 INTEGER MULTIPLY

ASSEMBLER: MUL <D>, <S1>, <S2>

FUNCTION UNIT: Multiplier

	16	16	16	16
	7320	D	Sl	52
0		16	32	48 63

DESCRIPTION: The contents of D are replaced by the least significant

half of the 128-bit integer product of the contents of S1

and S2. (See UMUL.)

EXCEPTIONS: Overflow

Overflow is set if the most significant 65 bits of the 128-bit product are not all identical. The resulting integer is the least significant half of the 128-bit pro-

duct.

5.49 NAND

ASSEMBLER: NAND <D>, <S1>, <S2> FUNCTION UNIT: IFU

16	16	16	16	
7111	D	Sl	S 2	
0	16	32	48	— 53

DESCRIPTION: The contents of D are replaced by the bit vector 'nand' of

the contents of S1 and S2. A bit position in the result is set to one if the corresponding bit positions of either S1 or S2 or both contain a zero; otherwise, the result

bit is set to zero.

5.50 NO-OPERATION

ASSEMBLER: NOP

FUNCTION UNIT: CFU

16	16	16	16	
0000	0000	0000	0000	
0	16	32	48	63

DESCRIPTION: No operation takes place.

5.51 NOR

ASSEMBLER: NOR <D>, <S1>, <S2>

FUNCTION UNIT: IFU

_	16	16	16	16
	7114	ם	Sl	S2
Ī	0	16	32	48 63

DESCRIPTION: The contents of D are replaced by the bit vector 'nor' of

the contents of S1 and S2. A bit position in the result is set to one if the corresponding bit positions of both S1 and S2 contain a zero; otherwise, the result bit is

set to zero.

5.52 NOT

ASSEMBLER: NOT <D>,<S1>

FUNCTION UNIT: IFU

 16
 16
 16
 16

 6110
 D
 51
 0000

 0
 16
 32
 48
 63

DESCRIPTION: The contents of D are replaced by the bit vector 'not' of

the contents of S1. A bit position in the result is set

to one if the corresponding bit position of Sl is zero;

otherwise, the result bit is set to zero.

5.53 ONE

ASSEMBLER: ONE <D>

FUNCTION UNIT: IFU

	16	16	16	16	
	4113	D	0000	0000	
,	0	16	32	48	63

DESCRIPTION: The contents of D are replaced by all ones.

5.54 LOGICAL SUM

ASSEMBLER: OR <D>, <S1>, <S2> FUNCTION UNIT: IFU

	16	16	16	16
	711B	D	Sl	S2
0		16	32	48 63

DESCRIPTION: The contents of D are replaced by the bit vector 'or' of

the contents of S1 and S2. A bit position in the result is set to one if the corresponding bit positions of either S1 or S2 contain a one; otherwise, the result bit is set

to zero.

5.55 PRIVILEGED CREATE (PRIVILEGED)

ASSEMBLER: PCR <D>, <S1> FUNCTION UNIT: CFU

	16	16	16	16	
	E97F	D	Sl	0000	
0		16	32	48	63

DESCRIPTION: The contents of Sl, a PSW, are placed in the first avail-

able location in the process queue, which activates a process associated with the new PSW. The PS field of the new PSW is not inherited from the executing PSW, but is obtained from the corresponding bits of the contents of S1. The contents of D are replaced by the address (PT) of the new PSW. If the queue location used by the create is the last available location, the create fault condition is ra-

ised in the PEM and a trap occurs.

EXCEPTIONS: Create fault.

5.56 PRIVILEGED REGISTER TO REGISTER MOVE (PRIVILEGED)

ASSEMBLER: PMOV <D>, <S1>

FUNCTION UNIT: IFU

	16	16	16	16
	E 11F	D	Sl	NOT USED
0		16	32	48 63

DESCRIPTION: The contents of D are replaced by the contents S1. The

instruction ignores reserved state on D and S1 unless the access state is specifically tested using :W, :F, or :E.

The access state of D is set full.

5.57 PRIVILEGED MASK REGISTER DESCRIPTOR (PRIVILEGED)

ASSEMBLER: PMRD <D>, <S1>, <S2> FUNCTION UNIT: IFU

	16	16	16	16	
	F101	ם	Sl	S2	
,	0	16	32	48 63	_ }

DESCRIPTION: The contents of D are replaced by the logical 'and' of a register descriptor derived from S2 and the contents of S1 (the mask). The instruction ignores reserved state on D, S1, and S2 unless the access state is specifically tested using :W, :F, or :E. The access state of D is set full. The format of the register descriptor is as follows:

DESCRIPTOR BITS	MEANING
0-55	0
56-57	Access state: empty if 10, full if 00, reserved if 11 or 01
58	0
59	Parity check bit: contents have even parity if 1, odd parity if 0
60	0
61-63	Data quality

5.58 PRIVILEGED MOVE REGISTER WITH DESCRIPTOR (PRIVILEGED)

ASSEMBLER: PMRG <D>, <S1>

FUNCTION UNIT: IFU

	16	16	16	16	
	ElOE	D	Sl	NOT USED	
•	0 .	16	32	48 6	3

DESCRIPTION: The contents of D are replaced by the contents of Sl. The instruction ignores reserved state on D and Sl unless the access state is specifically tested using :W, :F, or :E.The access state, parity check bit, and data quality of

D are set equal to those of S1.

EXCEPTIONS: Any data quality may be set.

5.59 PRIVILEGED STORE PSW (PRIVILEGED)

ASSEMBLER: PSPSW <>> FUNCTION UNIT: CFU

	16	16	16	16	
	C900	D	0000	0000	
0		16	32	48	63

DESCRIPTION: The contents of D are replaced by the executing PSW. All

fields, including the privileged (PS) field, are stored. The PSW that is stored contains a Program Counter (PC) field that has been incremented to point to the next in-

struction. The executing PSW is not affected.

5.60 PRIVILEGED SET REGISTER DESCRIPTOR (PRIVILEGED)

ASSEMBLER: PSRD <D>, <S1>, <S2> FUNCTION UNIT: IFU

	16	16	16	16
	FlOD	ם	Sl	S2
0		16	32	48 63

DESCRIPTION: The contents of D are replaced by the contents of S1 with the access state, parity check bit, and data quality formed from a register descriptor in S2. The format of the register descriptor is as follows:

DESCRIPTOR BITS	MEANING
0-55	0
56-57	Access state: empty if 10, full if 00, reserved if 11 or 01
58	0
59	Parity check bit: contents have even parity if 1, odd parity if 0
60	0
61-63	Data quality

EXCEPTIONS: Any data quality may be set.

In addition, the parity check bit of D may not agree with its contents.

QT	HEP PRINCIPLES OF OPERATION	QT
QTEQ		QTEQ
OTGE	CHAPTER 5 - INSTRUCTIONS	QTGE
OTGT		QTGT
QTLE		QTLE
QTLT	5.61 QUIT	QTLT
QTNE	·	QTNE

ASSEMBLER: QT

QT<c> <S2>

FUNCTION UNIT: CFU

8	4	4	16	16	16	
19	С	0	0000	0000	52	
0	8	12	16	32	48 63	}

DESCRIPTION: The contents of S2 are compared with zero. If the condition specified by <c> is true, the executing PSW is removed from the process queue and the associated process stops executing. The queue location then becomes available.

The possible test modes and the corresponding values for C are:

QT <c></c>	•	С
	(never)	0
QTLT	(less than)	1
QTEQ	(equal to)	2
QTLE	(less than or equal to)	3
QTGT	(greater than)	4
QTNE	(not equal to)	5
QTGE	(greater than or equal to>	6
QT	(always)	7
	(empty)	8
QTF	(full)	9
	(RI equal to zero)	A
	(RI not equal to zero)	В
	(CI equal to zero)	С
	(CI not equal to zero)	D
	(even parity)	E
	(odd parity)	F

Exceptions: None.

5.62 RESET CREATE FAULT (PRIVILEGED)

ASSEMBLER: RCF <S2>

FUNCTION UNIT: CFU

	16	16	16	16	j
	9903	0000	0000	52	
0		16	32	48	63

DESCRIPTION: The user create fault condition is reset if S2 contains

the integer zero, and the supervisor create fault condi-

tion is reset if S2 contains the integer 1.

5.63 READ CFU CONTROL (PRIVILEGED)

ASSEMBLER: RCTL <D> FUNCTION UNIT: CFU

16	16	16	16	
4A 00	D	0000	0000	7
0	16	32	48 6:	3

DESCRIPTION: The contents of D are replaced by the contents of the CFU Task Control Register, which has the format:

	30	1	1	16	16	
	0000000	S	ט	Live	Dormant	
0		30	31	32	48	63

where:

S = SCF If set to 1 means supervisor create fault condition
 exists.

U = UCF If set to 1 means user create fault condition exists.

Live Contains a bit corresponding to each task with the leftmost bit corresponding to Task 0 and the rightmost bit corresponding to Task 15; a one means the task has processes associated with it.

Dormant Contains a bit corresponding to each task with the leftmost bit corresponding to Task 0 and the rightmost bit corresponding to Task 15; a one means the task is dormant (e.g., a trap has occurred), and a zero means the task is active.

5.64 READ PEM CLOCK UNDER MASK

ASSEMBLER: RDCLK <D>, <S1>

FUNCTION UNIT: HA

	16	16	16	16	
ſ	670 4	D	S1	0000	
_	0	16	32	48 63	3

DESCRIPTION: The contents of D are replaced by the bit vector 'and' of

the contents of S1 (the mask) and the present value of the PEM's hardware clock. The clock is 64 bits; the low-order 48 bits comprise a cycle counter whose value is incremented every 100 nanosecond cycle; the high-order 16 bits uniquely identify the PEM in a multi-PEM system.

5.65 READ TASK COUNTER (PRIVILEGED)

ASSEMBLER: RDTC <D>,<S1>,<S2> FUNCTION UNIT: SPI

	16	16	16	16
ſ	7600	D	Sl	.S2
	0	16	32	48 63

DESCRIPTION: The contents of D are replaced by the Task Counter specified by the contents of S1 for the task specified by the integer contents of S2. The task counter determined by the contents of S1 is given below.

S1	Task Counter
0	wave off counter
1	floating-point instruction counter
2	data memory instruction counter
3	other instruction counter

Floating-point instructions are those whose mnemonics begin with the letter 'F'. Data memory instructions are LOD, LODI, LODX, STO, STOI and STOX. The sum of counters 1, 2 and 3 is the number of instructions executed by the task since the last time the counters were cleared. (See STCT, CLCT.)

5.66 READ ECC COUNTER (PRIVILEGED)

ASSEMBLER: RECC <D>, <S1>

FUNCTION UNIT: HA

	16	16	16	16	
	E705	D	Sl	0000	
L	0	16	32	48	63

DECCRIPTION: The contents of D are replaced by the logical 'AND' of the contents of S1 (the mask) and the current integer value of the 17-bit ECC Counter.

The ECC Counter counts the number of errors (corrected or uncorrected) detected in program memory; if the counter reaches 2**17-1, the high order bit remains a one while the rest of the counter starts over at zero. As a result of this, an ECC counter value lying between 2**16 and 2**17-1, inclusive, is only accurate modulo 2**16.

5.67 READ FLAGS (PRIVILEGED)

ASSEMBLER: RFLG <D>, <S1>

FUNCTION UNIT: HA

16	16	16	16	
E706	D	Sl	0000	
0	16	32	48	63

DESCRIPTION: The contents of D are replaced by the logical 'and' of the Communication Flags Register and the contents of Sl (the mask). The Communication Flags Register contains 64 flags which are used for synchronization between the PEM which contains them and the Diagnostic and Maintenance Processor. When a flag and its corresponding enable bit are both set, an interrupt occurs on the Diagnostic and Maintenance Processor. (See RNABL, SFLG, WFLG.)

5.68 READ ENABLE REGISTER (PRIVILEGED)

FUNCTION UNIT: HA ASSEMBLER: RNABL <D>, <S1>

	16	16	16	16
Ī	E707	D	Sl	0000
(0	16	32	48 63

DESCRIPTION: The contents of D are replaced by the logical 'and' of the Enable Register and the contents of S1 (the mask). The Enable Register is 64 bits wide and is used to enable interrupts on the Diagnostic and Maintenance Processor by communication flags. When a communication flag is set, if the corresponding enable bit is set an interrupt occurs on the Diagnostic and Maintenance Processor via the Low Speed The Enable Register can be modified only by the Diagnostic and Maintenance Processor. (See RFLG, SFLG, WFLG.)

5.69 READ PROGRAM MEMORY (PRIVILEGED)

ASSEMBLER: RPM <D>, <S2>

FUNCTION UNIT: HA

	16	16	16	16	
	D700	D	0000	S2	
0		16	32	48	63

DESCRIPTION: The contents of D are replaced by the contents of the real

program memory location addressed by the contents of S2. The program base and program limit values from the TSW are not used to relocate or check the program memory address.

(See WPM.)

5.70 READ PSW (PRIVILEGED)

ASSEMBLER: RPSW <D>, <S2>

FUNCTION UNIT: CFU

	16	16	16	16	
	5800	D	0000	S2	
0		16	32	48	63

DESCRIPTION: S2 contains the PT of the PSW to be read. The contents of

D are replaced by the PSW contained in the Process Queue

location addressed by the PT.

EXCEPTIONS: This instruction generates a wave-off during the next

cycle in the PEM.

5.71 READ SSW (PRIVILEGED)

ASSEMBLER: RSSW <D>,<S1>

FUNCTION UNIT: SFU

16	16	16	16	
EC00	D	S1	NOT USED	
0	16	32	48	63

DESCRIPTION: The SSW half specified by the contents of Sl replaces the

contents of D. Bits 56 through 62 from Sl contain the PT and bit 63 specifies whether D is to be read from the control half of the SSW (bit 63=0) or the data half

(bit 63=1).

5.72 READ TSW (PRIVILEGED)

ASSEMBLER: RTSW <D>, <S2>

FUNCTION UNIT: HA

_	16	16	16	16	
	5700	D	0000	S2	
•	0	16	32	48 63	•

DESCRIPTION: The contents of D are replaced by the contents of the TSW half specified by the contents of S2. The contents of S2 are divided by two; the quotient determines the task and the remainder (the least significant bit in S2) the TSW

half as shown below. (See WTSW.)

REMAINDER

TSW HALF

		12	10	10		16		16
0		СВ	RB	RL		PB	P	L
	0		12	22	32		48	63
		16		24			24	
1		0000		DB			DL	
	0		16			40		63

5.74 SHIFT CIRCULAR

ASSEMBLER: SC <D>, <S1>, <S2> FUNCTION UNIT: IFU

	16	16	16	16	
	7106	D	Sl	S2	
0		16	32	48 6	 3

DESCRIPTION: The contents of D are replaced by the result obtained by

shifting the contents of S1 the number of bit positions specified by the contents of S2. A positive shift count indicates a left shift and a negative shift count indicates a right shift. In either case, bits shifted off the end fill the bit positions vacated at the opposite end.

5.75 SET FLAG (PRIVILEGED)

ASSEMBLER: SFLG <flag> FUNCTION UNIT: HA

	0700	0000	flag	0000	
1	0	16	32	48	 63

DESCRIPTION: The instruction waits until the communication flag register bit position specified by flag contains a zero; then a one is set into the bit position. The value of flag is taken to be modulo 64, but the high order bit of flag (instruction bit 32) must be set.

> If the corresponding enable bit is set when the instruction completes, an interrupt occurs on the Diagnostic and Maintenance Processor. A flag can be cleared only by the Diagnostic and Maintenance Processor. (See RNABL, RFLG, WFLG.)

5.76 SHIFT LOGICAL

ASSEMBLER: SL <D>,<S1>,<S2> FUNCTION UNIT: IFU

	16	16	16	16	
	7104	D	Sl	S2	
0		16	32	48	63

DESCRIPTION: The contents of D are replaced by the result obtained by shifting the contents of Sl the number of bit positions

shifting the contents of S1 the number of bit positions specified by the contents of S2. A positive shift count indicates a left shift and a negative shift count indicates a right shift. In either case, bits shifted off the end are lost and bit positions vacated at the opposite end

are zero-filled.

5.77 STORE PSW

ASSEMBLER: SPSW <D>

FUNCTION UNIT: CFU

	16	16	16	16	
	5900	D	0000	0000	
0		16	32	48	63

DESCRIPTION: The contents of D are replaced by the executing PSW, with

the PS field set to zero and the PC field incremented to point to the next instruction. SPSW is the same as a con-

ditional SQT with an always failing condition.

SQT	HEP PRINCIPLES OF OPERATION	SQT
SQTEQ		SQTEQ
SQTGE	CHAPTER 5 - INSTRUCTIONS	SQTGE
SOTGT	·	SQTGT
SOTLE		SQTLE
SOTLT	5.78 STORE PSW AND QUIT	SQTLT
SQINE	<u>-</u>	SQTNE

ASSEMBLER: SQT <D>

SQT<c> <D>,<S2>

FUNCTION UNIT: CFU

8	4	4	16	16	16	
59	С	0	. D	0000	S2	
0	8	12	16	32	48	63

DESCRIPTION: The contents of D are replaced by the executing PSW, with the PS field set to zero and the PC field incremented to point to the next instruction. In addition, the contents of S2 are compared with zero. If the condition specified by <c> is true, the executing PSW is removed from the Process Queue and the associated process stops executing. The queue location then becomes available. Note that storing of the PSW always takes place, whether or not the

process quits.

The possible test modes and the corresponding values for C are:

```
SQT<c>
                                 C
SPSW (never)
SQTLT (less than)
SQT (equal to)
SQTLE (less than or equal to)
                                 3
SQTGT (greater than)
SQTNE (not equal to)
                                 5
SQTGE (greater than or equal to>
                                6
                                 7
SQT
      (always)
SQTE (empty)
                                 9
SQTF
      (full)
      (RI equal to zero)
                                 Α
      (RI not equal to zero)
                                 В
      (CI equal to zero)
                                 С
      (CI not equal to zero)
                                 D
      (even parity)
                                 E
      (odd parity)
```

5.79 SET REGISTER DESCRIPTOR

ASSEMBLER: SRD <D>,<S1>,<S2> FUNCTION UNIT: IFU

	16	16	16	16 .	
	710D	D	Sl	S 2	7
0		16	32	48 6	3

DESCRIPTION: The contents of D are replaced by the contents of Sl with the access state, parity check bit, and data quality formed from a register descriptor in S2. The format of the register descriptor is as follows:

DESCRIPTOR BITS	MEANING
0-55	0
56	Access state: empty if 1, full if 0
57-58	0
59	Parity check bit: contents have even parity if 1, odd parity if 0
60	0
61-63	Data quality

Table 5.79 - Register Descriptor Contents

EXCEPTIONS: Any data quality may be set.

In addition, the parity check bit of D may not agree with its contents.

5.80 STORE TASK COUNTER

ASSEMBLER: STTC <D>, <S1>

FUNCTION UNIT: SPI

	16	16	16	16	
	6600	D	S1	0000	
0		16	32	48	63

DESCRIPTION: The contents of D are replaced by the contents of the task counter specified by the contents of S1 for the currently executing task. The task counter determined by the contents of S1 is given below.

Sl	Task	counter	
J.L.	145/1	COULT C C L	

- 0 wave off counter
- floating-point instruction counter
- 2 Data memory instruction counter
- 3 other instruction counter

Floating-point instructions are those whose mnemonics begin with the letter 'F'. Data memory instructions are LOD, LODI, LODX, STO, STOI and STOX. The sum of counters 1, 2 and 3 is the number of instructions executed by the task since the last time the counters were cleared. (See RDCT, CLCT.)

5.81 STORE DATA MEMORY

ASSEMBLER: STO <atlist> <S2>,<M> FUNCTION UNIT: SFU 8 1 1 1 1 2 2 32 16 9C C N R B SAC DAC M **S2** 0 8 12 14 16 48 63

Note the reversal of assembler operands.

DESCRIPTION: The contents of the addressed data memory location are replaced by the contents of S2. The displacement part of the address M and the B, SAC, and DAC parts are obtained from the instruction or from the contents of S1, depending on instruction bit R. Instruction bits C, N, and R are specified by <atlist>, a sequence of <attributes>. Instruction bits are set when their corresponding <attribute> is present, with the following meanings:

BIT OF INSTRUCTION	CORRESPONDING ATTRIBUTE	MEANING:
8	: C	Copy access state. See :N.
9	: N	Numeric. If present and :C is present, store left justified partial words from S2; otherwise, store right justified partial words from S2.
10	:R	Register address. Must be absent.

5.82 STORE DATA MEMORY INDIRECT

ASS	SEMBLER:	S'	ro:	I •	<at< th=""><th>llist</th><th>t> <</th><th><s2>,<s1></s1></s2></th><th></th><th>FUNCTION</th><th>UNIT:</th><th>SFU</th></at<>	llist	t> <	<s2>,<s1></s1></s2>		FUNCTION	UNIT:	SFU
	8	1	1	1	1	2	2	16	16		16	
	ВС	С	N	R	В	SAC	DAC	0000	Sl		S2	
	0	8						16	32	48	6	3

Note the reversal of assembler operands.

DESCRIPTION: The contents of the addressed data memory location are replaced by the contents of S2. The displacement part of the address is the contents of S1 modulo 2**32. The B, SAC, and DAC parts are obtained from the instruction or from the contents of S1, depending on instruction bit R. Instruction bits C, N, and R are specified by <atlist>, a sequence of <attributes>. Instruction bits are set when their corresponding <attribute> is present, with the following meanings:

BIT OF INSTRUCTION	CORRESPONDING ATTRIBUTE	MEANING:
8	: C	Copy access state. See :N.
9	: N	Numeric. If present and :C is present, store left justified partial words from S2; otherwise, store right justified partial words from S2.
10	:R	Register address. If present, the B, SAC, and DAC subfields of the data memory address are obtained from bits 27 through 31 of the contents of S1; if absent, they are obtained from instruction bits 11 through 15.

5.83 STORE DATA MEMORY INDEXED

ASS	SEMBLER:	S	roz	Κ <	<at< th=""><th>:list</th><th>t> <</th><th><pre><s0>,<s1>,<s2)< pre=""></s2)<></s1></s0></pre></th><th></th><th>FUNCTION</th><th>UNIT:</th><th>SFU</th></at<>	:list	t> <	<pre><s0>,<s1>,<s2)< pre=""></s2)<></s1></s0></pre>		FUNCTION	UNIT:	SFU
	8	1	1	1	1	2	2	16	16		16	
	FC	c	N	R	В	SAC	DAC	S0	Sl		S2	
	0	8				12	14	16	32	48	6	 i3

DESCRIPTION: The contents of the addressed data memory location are replaced by the contents of SO. The displacement part of the address is the sum of the contents of Sl and the contents of S2 modulo 2**32. The B, SAC, and DAC parts are obtained from the instruction or from the contents of Sl, depending on instruction bit R. Instruction bits C, N, and R are specified by <atlist>, a sequence of <attributes>. Instruction bits are set when their corresponding <attribute> is present, with the following meanings:

BIT OF INSTRUCTION	CORRESPONDING ATTRIBUTE	MEANING:
8	: C	Copy access state. See :N.
9	:N	Numeric. If present and :C is present, store left justified partial words from S2; otherwise, store right justified partial words from S2.
10	:R	Register address. If present, the B, SAC, and DAC subfields of the data memory address are obtained from bits 27 through 31 of the contents of S1; if absent, they are obtained from instruction bits 11 through 15.

5.84 INTEGER SUBTRACT

ASSEMBLER: SUB <D>, <S1>, <S2> FUNCTION UNIT: IFU

_	16	16	16	16
ſ	7139	D	Sl	S2
	0	16	32	48 63

DESCRIPTION: The contents of D are replaced by the integer result obta-

ined by subtracting the contents of S2 from the contents

of Sl.

EXCEPTIONS: Carry, overflow

Carry is set if there was no borrow out of the most significant bit of the result and overflow does not occur.

Overflow is set if both operands are of opposite signs, and the sign of the result is the same as the sign of S2. The resulting integer is too small by 2**64 if its sign is negative and too large by 2**64 if its sign is positive.

5.85 SUPERVISOR CALL

ASSEMBLER: SVC <code> FUNCTION UNIT: IFU

	16	16	16	16	
	0900	code	0000	0000	
0		16	32	48	63

DESCRIPTION: The task to which the executing process belongs is made

dormant. A process is created in the corresponding supervisor task; the code field of the instruction replaces the RI field of the supervisor PSW and indicates the type of supervisor function requested. The PC field of the su-

pervisory PSW is set to 0001A, and the UTM is zero.

TEQ	HEP PRINCIPLES OF OPERATION	TEQ
TGE		TGE
TGT	CHAPTER 5 - INSTRUCTIONS	TGT
TLE		TLE
TLT		TLT
TNE	5.86 INTEGER COMPARE, INTEGER RESULT	TNE

ASSEMBLER: T<c> <D>, <S1>, <S2> FUNCTION UNIT: IFU

8	4	4	16	16	16	
71	4	С	D	S1	S2	
0	8	12	16	32	48	63

DESCRIPTION: The integer contents of S1 and S2 are compared according to <c>; if the condition is true the contents of D are replaced by positive integer one; if the condition is false the contents of D are replaced by zero.

The possible test modes and the corresponding values for C are:

T <c></c>	С
(never)	0
TGT (greater than)	1
TEQ (equal to)	2
TGE (greater than or equal to)	3
TLT (less than)	4
TNE (not equal to)	5
TLE (less than or equal to>	6
(always)	7

TFEQ	HEP PRINCIPLES OF OPERATION	TFEQ
TFGE		TFGE
TFGT	CHAPTER 5 - INSTRUCTIONS	TFGT
TFLE		TFLE
TFLT		TFLT
TFNE	5.87 INTEGER COMPARE, FLOATING RESULT	TFNE
	• • • • • • • • • • • • • • • • • • • •	

ASSEMBLER: TF<c> <D>, <S1>, <S2> FUNCTION UNIT: IFU

8	4	4	16	16	16	
71	6	С	D	Sl	S2	
0	8	12	16	32	48 6	3

DESCRIPTION: The integer contents of S1 and S2 are compared according to <c>; if the condition is true the contents of D are replaced by positive floating-point one; if the condition is false the contents of D are replaced by zero.

The possible test modes and the corresponding values for C are:

TF <c></c>	С
(always)	0
TFGT (greater than)	1
TFEQ (equal to)	2
TFGE (greater than or equal to) 3
TFLT (less than)	4
TFNE (not equal to)	5
TFLE (less than or equal to>	6
(never)	7

TLEQ	HEP PRINCIPLES OF OPERATION	TLEQ
TLGE		TLGE
TLGT	CHAPTER 5 - INSTRUCTIONS	TLGT
TLLE		TLLE
TLLT		TLLT
TLNE	5.88 INTEGER COMPARE, BIT VECTOR RESULT	TLNE

ASSEMBLER: TL<c> <D>, <S1>, <S2> FUNCTION UNIT: IFU

8	4	4	16	16	16	
71	5	С	D	Sl	S2	
0	8	12	16	32	48 - 63	

DESCRIPTION: The integer contents of S1 and S2 are compared according to <c>; if the condition is true the contents of D are replaced by the bit vector having all bits set; if the condition is false the contents of D are replaced by the bit vector with all bits zero.

The possible test modes and the corresponding values for C are:

TL <c></c>	•	С
	(never)	0
TLGT	(greater than)	1
TLEQ	(equal to)	2
TLGE	(greater than or equal to)	3
TLLT	(less than)	4
TLNE	(not equal to)	5
TLLE	(less than or equal to>	6
	(always)	7

5.89 INTEGER MULTIPLY UPPER

ASSEMBLER: UMUL <D>, <S1>, <S2>

FUNCTION UNIT: Multiplier

16	16	16	16
7310	D	Sl	52
0	16	32	48 63

DESCRIPTION: The contents of D are replaced by the most significant

half of the 128-bit integer product of the contents of Sl

and S2. (See MUL.)

5.90 WAIT ON FLAG (PRIVILEGED)

ASSEMBLER: WFLG <flag>

FUNCTION UNIT: HA

	16	16	16	16	
	8700	0000	flag	0000	
0		16	32	48	63

DESCRIPTION: The instruction waits until the communication flags regis-

ter bit position specified by flag contains a zero. The value of the flag is taken modulo 64, but the high order bit of flag (instruction bit 32) must be set. A flag can be set only by the Diagnostic and Maintenance Processor.

(See RNABL, RFLG, SFLG.)

5.91 WRITE PROGRAM MEMORY (PRIVILEGED)

ASSEMBLER: WPM <S1>,<S2> FUNCTION UNIT: HA

16	16	16	16	
B700	0000	Sl	S2	
0	16	32	48 63	•

DESCRIPTION: The contents of the real program memory location addressed

by the contents of S2 are replaced by the contents of S1. The program base and program limit values from the TSW are not used to relocate or check the program memory address.

(See RPM.)

5.92 WRITE PSW (PRIVILEGED)

ASSEMBLER: WPSW <S1>, <S2>

FUNCTION UNIT: CFU

	16	16	16	16	
	3B7F	0000	Sl	S2	
0		16	32	48	63

DESCRIPTION: S2 contains the PT of the Process Queue location where the

PSW in Sl is to be written. The contents of the Process

Queue location are replaced by the contents of S1.

CHAPTER 5 - INSTRUCTIONS

5.93 WRITE SSW (PRIVILEGED)

ASSEMBLER: WSSW <S1>, <S2>

FUNCTION UNIT: SFU

	16	16	16	16
	3C00	0000	Sl	. S2
0		16	32	48 63

DESCRIPTION: The contents of S2 are stored in the SSW half specified by

the contents of Sl. Bits 56 through 62 from Sl contain the PT and bit 63 specifies whether S2 is to be stored in the control half of the SSW (bit 63=0) or the data half (bit 63=1).

EXCEPTIONS: None.

CHAPTER 5 - INSTRUCTIONS

5.94 WRITE TSW (PRIVILEGED)

ASSEMBLER: WTSW <S1>,<S2> FUNCTION UNIT: HA

	16	16	16	16	
	3700	0000	Sl	S2	
0		16	32	48	63

DESCRIPTION: The contents of the TSW half specified by the integer contents of S2 are replaced by the contents of S1. The contents of S2 are divided by two; the quotient determines the task and the remainder (the least significant bit in S2) the TSW half as shown below. (See RTSW.)

REMAI	NDER		TSW	HALF		
	12	10	10	16	16	5
0	СВ	RB	RL	PB	PI	•
	0	12	22	32	48	63
	16		24		24	
1	0000		DB		DL	
	0	16	1.0	40		63

EXCEPTIONS: None.

XPSW	HEP PRINCIPLES OF OPERATION	XPSW
XPSWEQ		XPSWEQ
XPSWGE	CHAPTER 5 - INSTRUCTIONS	XPSWGE
XPSWGT		XPSWGT
XPSWLE		XPSWLE
XPSWLT	5.95 EXCHANGE PSW	XPSWLT
XPSWME		XPSWNE

ASSEMBLER: XPSW <D>, <Sl>(<action code list>) FUNCTION UNIT: CFU

XPSW<c> <D>,<Sl>(<action code list>),<S2>

8	4	4	16	16	16	
79	С	ACT	D	Sl	S2	
0	8	12	16	32	48	 53

DESCRIPTION:

The contents of D are replaced by the executing PSW with the PS field set to zero and the PC field incremented to point to the next instruction. In addition, the contents of S2 are compared with zero. If the condition specified by <c> is true, the Process Queue location containing the executing PSW is modified by the fields of Sl, a PSW, according to the action specified by ACT. Therefore the process associated with the old PSW stops executing and a new process starts executing associated with the new PSW. The PS field of the new PSW is the same as the old PSW. The <action code list> specifies ACT, and consists of a sequence of <action code> items separated by commas. The meaning of each <action code> and its corresponding ACT value is given below; when more than one <action code> is specified, ACT values are additive.

<action code=""></action>	ACT	MODIFICATION BY S1
RCI	8	REPLACE CI
RRI	4	REPLACE RI
RUTM	2	REPLACE UTM
RPC	1	REPLACE PC

For fields other than PC, if no <action code> is specified the PSW of the created process matches the PSW of the executing process in that field. If RPC is not specified, the PC of the executing process is incremented to form the PC of the created process.

The possible test modes and the corresponding values for C are:

XPSW XPSWEQ	HEP PRINCIPLES		XPSWEQ	
XPSWGE XPSWGT XPSWLE XPSWLT	CHAPTER 5 - 1	INSTRUCTIONS	XPSWGE XPSWLE XPSWLE XPSWLT	
XPSWNE			XPSWNE	
	XPSW <c></c>	•	с	
		(never)	0	
	YDCWI T	(less than)	1	
•		(equal to)	2	
		(less than or equal to)	3	
		(greater than)	4	
		(not equal to)	5	
		(greater than or equal to>	6	
	XPSW	(always)	7	
	XPSWE		8	
	XPSWF	(full)	9	
		(RI equal to zero)	A	
		(RI not equal to zero)	В	
•		(CI equal to zero)	Ċ	
		(CI not equal to zero)	D	
		(even parity)	E	
		(odd parity)	F	

EXCEPTIONS: None.

APPENDIX A - GENERALIZED INSTRUCTIONS

APPENDIX A

GENERALIZED INSTRUCTIONS

The instructions described in this appendix are primarily of interest to numerical analysts, compiler writers and systems programmers.

The generalized floating-point instructions give the programmer the ability to manipulate signs of floating-point operands and to control significance in floating-point addition. Generalized integer and bit vector instructions are useful to compiler writers and systems programmers. The generalized program memory instructions are used to write program memory error diagnostics. Important special cases of these instructions are discussed separately in Chapter 5.

APPENDIX A - GENERALIZED INSTRUCTIONS

A.1 GENERALIZED FLOATING ADD INSTRUCTIONS

	(No Mnemonic)						ADDER	
	4	4	4	4	16	16	16	
	n	2	los	sign	ם	Sl	S2	
0)	4	8	12	16	32	48 6	 3

DESCRIPTION: The contents of D are replaced by the floating-point sign of the contents of S1 and S2 modified by sign control as described below. Both normalized and unnormalized operation with significance control are supported. An unnormalized quantity may be normalized by performing a normalized add with zero. A normalized quantity may be unnormalized by performing an unnormalized add with unnormalized zero (i.e., zero mantissa, non-zero exponent). For a further discussion, see FADD or FSUB. The operation of each field of the instruction is as follows:

- n = 7 Normalized
 - = F Unnormalized

los = Significance limit. A normalization shift greater than 15-los causes loss of significance to be set in the result. A normalization shift of zero means a carry has occurred in the addition of the aligned mantissas; a normalization shift of 15 is the largest possible.

sign = Sign control, as shown in the following table:

	SIGN	RESULT		
0	(0000)	(S1)	+	(S2)
1	(0001)	(S1)	-	(S2)
2	(0010)	(S1)	-	(S2)
3	(0011)	(S1)	+	(S2)
4	(0100)	(S1)	+	(S2)

APPENDIX A - GENERALIZED INSTRUCTIONS

A prenormalized mantissa of zero never causes loss of significance.

In unnormalized arithmetic, loss of significance occurs only if los is F (hexadecimal) and no carry occurred in adding the aligned mantissas.

Members of this set of instructions described separately are FADD (7200) and FSUB (7201).

APPENDIX A - GENERALIZED INSTRUCTIONS

A.2 GENERALIZED FLOATING MULTIPLY/DIVIDE INSTRUCTIONS

(No Mnemonic)

MULTIPLIER/DIVIDER

	4	4	4	4	16	16	16	
	7	unit	0	sign	D	Sl	S2	
•	0	4	8	12	16	32	48	63

DESCRIPTION: The contents of D are replaced by the result of the operation specified by unit; the sign of the result is determined by sign control. See FMUL or FDIV. The operation

of each field in these instructions is as follows:

unit = 3 Multiply
= F Divide

sign = Sign control, as follows:

	SIGN	RESULT		
0	(0000)	(S1)	* or /	(S2)
1	(0001)	(S1)	* or /	[- (S2)]
2	(0010)	(S1)	* or /	[- (S2)]
3	(0011)	(S1)	* or /	(S2)
4	(0100)	(S1)	* or /	(S2)
5	(0101)	(S1)	* or /	[- (S2)]
6	(0110)	(S1)	* or /	[- (S2)]
7	(0111)	(S1)	* or /	(S2)
8	(1000)	(S1)	* or /	(52)
9	(1001)	(S1)	* or /	[- (S2)]
A	(1010)	(S1)	* or /	[- (S2)]

APPENDIX A - GENERALIZED INSTRUCTIONS

EXCEPTIONS: See FMUL or FDIV.

Members of this set of instructions described separately are FMUL (7300) and FDIV (7F00).

APPENDIX A - GENERALIZED INSTRUCTIONS

A.3 GENERALIZED INTEGER ARITHMETIC INSTRUCTIONS

(No Mnemonic) IFU

4 4 4 4 16 16 16

op 1 cry func D S1 S2

DESCRIPTION: The contents of D are replaced by the result of the integer operation described by func on the operands contained in S1 and S2. The value of op depends on func. If cry = 2, carry-in is zero; if cry = 3, carry-in is 1.

32

The table below summarizes the operation of this group of instructions.

48

63

	FUNC	RESULT	OP
0	(0000)	(S1) + carry-in	6
1	(0001)	(S1) + [(S1) .ANDNOT. (S1)] + carry-in	7
2	(0010)	(S1) + [(S1) .AND. (S2)] + carry-in	7
3	(0011)	2 * (S1) + carry-in	6
4	(0100)	[(S1) .OR. (S2)] + carry-in	7
5	(0101)	[(S1) .OR. (S1)] + [(S1) .ANDNOT. (S2)] + carry-in	7
6	(0110)	(S1) + (S2) + carry-in	7
7	(0111)	(S1) + [(S1) .OR. (S2)] + carry-in	. 7
8	(1000)	[(S1) .ORNOT. (S1)] + carry-in	7
9	(1001)	(S1) - (S2) - 1 + carry-in	7
A	(1010)	[(S1) .ORNOT. (S1)] + [(S1) .AND. (S2)] + carry-in	7
В	(1011)	(S1) + [(S1) .ORNOT. (S1)] + carry-in	7
C	(1100)	-l + carry-in	4
D	(1101)	[(S1) .ANDNOT. (S1)] -1 + carry-in	7
		[(S1) .AND. (S2)] -1 + carry-in	7
F	(1111)	(S1) -1 + carry-in	6

EXCEPTIONS: Carry, Overflow

Carry is set if there was a carry out of the most significant bit of the result and overflow is not set.

Overflow is set according to the following table:

0

4

8

12

16

APPENDIX A - GENERALIZED INSTRUCTIONS

Sign of Sl Sign of S2 Sign of D FUNC VALUES	Sign of	Sl	Sign	of	S2	Sign	of	D	FUNC VALUES
---	---------	----	------	----	-----------	------	----	---	-------------

+	+	-	0,1,2,3,4,5,6,7	(OXXX)
+	-	-	0,1,2,3,8,9,A,B	(XOXX)
-	-	+	2,3,6,7,A,B,E,F	(XX1X)
_	+	+	1.3.5.7.9.B.D.F	(XXX1)

If one of the four sign combinations above occurs on an operation which has a corresponding func value, overflow is set.

Members of this set of instructions described separately are:

DEC 612F INC 6130 ADD 7126 SUB 7139

APPENDIX A - GENERALIZED INSTRUCTIONS

A.4 GENERALIZED BIT VECTOR OPERATIONS

16	

IFU

4	_	4		16	16	16	
op	1	1	func	D	Sl	52	
0	4	8	12	16	32	48	63

DESCRIPTION: The contents of D are replaced by the result of the bit vector operation described by func on the operands contained in S1 and S2. The value of op depends on func. The table below summarizes the operation of this group of instructions.

	FUNC	RESULT	op
0	(0000)	.NOT. (S1)	6
1	(0001)	.NOT. (S1) .ORNOT. (S1)	7
2	(0010)	.NOT. (S1) .OR. (S2)	7
3	(0011)	-1	4
4	(0100)	.NOT. (S1) .AND.NOT. (S2)	7
5	(0101)	.NOT. (S2)	5
6	(0110)	.NOT. [(S1) .XOR. (S2)]	7
7	(0111)	(S1) .ORNOT. (S1)	7
8	(1000)	.NOT. (S1) .AND. (S2)	7
9	(1001)	(S1) .XOR. (S2)	7
A	(1010)	(S2)	5
В	(1011)	(S1) .OR. (S2)	7
С	(1100)	0	4
D	(1101)	(S1) .ANDNOT. (S1)	7
E	(1110)	(S1) .AND. (S2)	7
F	(1111)	(S1)	6

EXCEPTIONS: None.

Members of this set of instructions described separately are:

AND	711E	EOR	7119	NAND	7111	ONE	4113
CLR	411C	IMPL	7112	NOR	7114	OR	711B
EOV	7116	MOV	611F	NOT	7110		

APPENDIX A - GENERALIZED INSTRUCTIONS

A.5 GENERALIZED PROGRAM MEMORY INSTRUCTIONS

(No Mnemonic)

HA

_	4	4	4	4	16	16	16	
	op	7	func	0	D	Sl	S 2	
(0	4	8	12	16	32	48 63	,

DESCRIPTION: If op = B, the contents of the real program memory addressed by the contents of S2 are replaced by the contents of S1. The parity check bits are not written if func = 4, and are written as computed from the contents of S1 if func = 0. If op = D, the contents of D are replaced by the contents of the real program memory location addressed by the contents of S2. If func = C, the parity check bits of the program memory location replace the contents of D. If func = 4, no error correction is performed on the value replacing the contents of D; if func = 0, error correction is performed on the value replacing the contents of D.

EXCEPTIONS: None.

Members of this set of instructions described separately are:

RPM D700 WPM B700