88951100

CONTROL DATA CORPORATION

CDC® HARDWARE FLOATING POINT UNIT 1781-1

GENERAL DESCRIPTION
FUNCTIONAL DESCRIPTION
OPERATING INSTRUCTIONS
INSTRUCTION DESCRIPTIONS
PROGRAMMING INFORMATION

HARDWARE REFERENCE MANUAL

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(3/77)	
Publication No. 88951100	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:

Control Data Corporation
Publications and Graphics Division
4455 Eastgate Mall
La Jolla, California 92037

or use Comment Sheet in the back of this manual.

MANUAL TO EQUIPMENT LEVEL CORRELATION SHEET

This manual reflects the equipment configurations listed below.

EXPLANATION: Locate the equipment type and series number, as shown on the equipment FCO log, in the list below. Immediately to the right of the series number is an FCO number. If that number and all of the numbers underneath it match all of the numbers on the equipment FCO log, then this manual accurately reflects the equipment.

EQUIPMENT TYPE	SERIES	WITH FCOs	COMMENTS
BT221-A	01		

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV	PAGE	REV	PAGE	REV	PAGE	REV	PAGE	REV
Cover	-								
Title Page	-								
ii	A								
iii/iv	A								
v/vi	A								
vii/viii	A								
ix	A								
1-1 thru 1-4	A								
2-1 thru 2-4	A								
3-1 thru 3-4	A								
4-1 thru 4-12	A								
5-1 thru 5-5	A								
A-1 thru A-3	A								
B-1 thru B-3	A								
Comment Sheet	A								
Back Cover	-								

PREFACE

This manual refers to the CDC® 1781-1 Hardware Floating Point Unit used with the CYBER 18-17 (System 17) Computer system. The manual assumes the user is familiar with CYBER 18-17 hardware and software operation.

The following Control Data Corporation publications may be referenced for additional information:

Publication	Publication No.
1784 Computer Reference Manual	89633400
1784 Computer Input/Output Specifications	89673100
CYBER 18-17 Installation Manual	88996000

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

APPENDIXES

FIGURES

TABLES

SYSTEM DESCRIPTION

The CDC® 1781-1 Hardware Floating Point Unit (HFPU) is a hardware option that provides CYBER 18-17/System 17 systems with floating point arithmetic capability and greatly reduces the software overhead required to do the same function.

It processes floating point numbers in single precision or double precision mode; operates in absolute, relative, and/or index address mode; and functions in either word, block, or hog DSA priority mode under program control.

This manual provides a description and programming information necessary to operate the hardware floating point unit.

PHYSICAL CHARACTERISTICS

Figure 1-1 shows a typical configuration of the system. The HFPU consists of seven 50-pack CYBER 18-17/System 17 type printed wiring assemblies that are inserted into assigned card positions (figure 1-2) in the 1783-1 Expansion Enclosure (part of the CYBER 18-17/System 17 hardware). The printed wiring assemblies consist of a combination of TTL logic, read only memory, and micro processors that interface directly to the CYBER 18-17/System 17 internal A/Q and DSA bus. The printed wiring assembly utilize three connector assemblies that fit over the assigned backplane area to provide routing of interboard signals (figure 1-3). No external interface cable assemblies are required.

The HFPU obtains all of its operating voltages and cooling from the expansion enclosure it resides in.

A 1785-1 A/Q Channel Expansion and a 1785-2 DSA Channel Expansion are required in addition to the expansion enclosure whenever a hardware floating point unit is to be a part of the system.

FUNCTIONAL CHARACTERISTICS

The seven printed wiring assemblies that make up the HFPU are:

- A/Q Interface (A/Q)
- Direct storage access (DSA) interface
- Timing and exponent
- Floating point hardware micro processor
- Address ALU
- Single precision ALU
- Double precision ALU

The A/Q and DSA printed wiring assemblies interface directly with the CPU internal A/Q and DSA bus via the 1785-1/2 Channel Expansions. These two interfaces are responsible for establishing and maintaining control between the central processing unit (CPU) and the HFPU during arithmetic operation (figure 1-4). The DSA activity can be programmed for three different modes of priority: word, block, and hog mode. The primary function of the address ALU is to establish and maintain proper DSA address generation and sequencing during the HFPU operation. These addresses are associated with the HFPU program counter, command code retrieval, operand (data) retrieval and storage, indexing, and the stop/save address register. The addressing is done in four different modes: absolute, relative, indexed, and indexed unmultiplied. The addressing allows the user to access all memory locations within a 65K word memory.

The single precision ALU and double precision ALU decode 16 standard and nine special floating point command codes, perform floating point operations, and compare the results

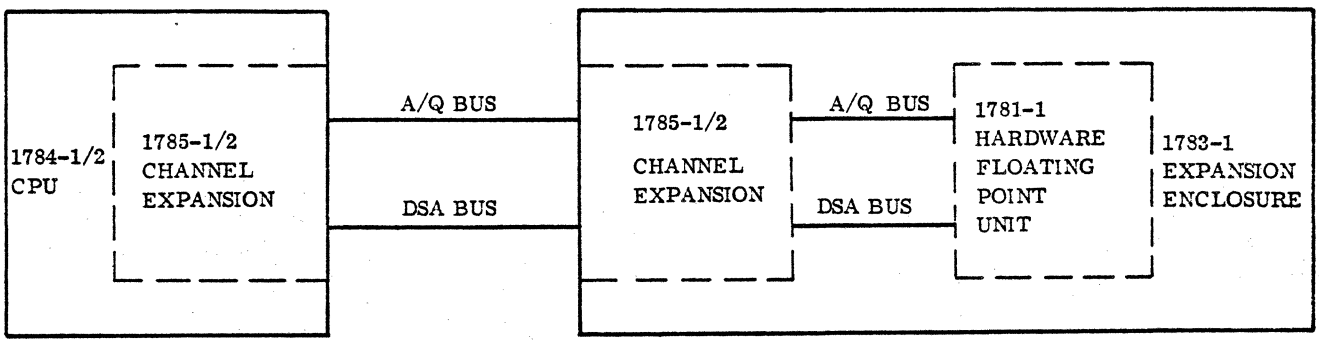


Figure 1-1. System Configuration

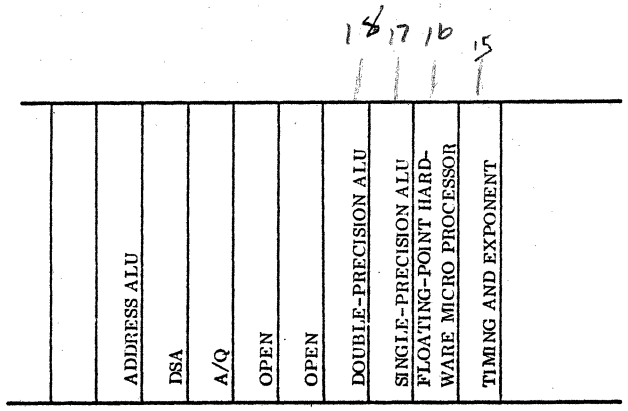


Figure 1-2. Slot Assignment

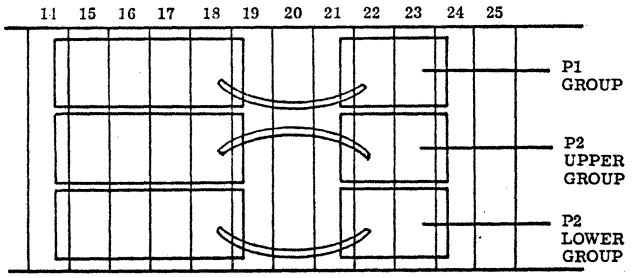


Figure 1-3. Backplane Connector Board Locations

of the mantissa (coefficient) portion of the floating point accumulator. Both the single precision ALU and the double precision ALU contain portions of the floating point. Two accumulator levels of accuracy are available to the user: single precision with a 32-bit operand and double precision with a 48-bit operand. The timing and exponent printed wiring assembly performs floating point operation on the exponent portion of the floating point data word. Sign and magnitude checks are also performed on the timing and exponent printed wiring assembly.

The primary purpose of the floating point hardware micro processor is to provide control and timing signals to the HFPU operation and to detect fault conditions such as overflow, underflow, and divide faults.

SYSTEM FUNCTIONAL CHARACTERISTICS

The HFPU interfaces directly with the CYBER 18-17/System 17 A/Q and DSA bus system and responds to A/Q commands only when it is presented with an equipment address (Q10 through Q07) during a write/read function that is identical to that selected at the equipment (for system signal flow, see figure 1-5).

The HFPU address is selected by inserting the appropriate jumpers (Q10 through Q07) on the A/Q interface board (position 21). Refer to section 3 for information on setting up the equipment code. Specific A/Q write commands are sent to the HFPU to prepare it for operation and to cause it to execute. The following functions can be accomplished via A/Q write commands:

Function 0	–	Load/clear function/status register
Function 1	–	Load/clear command code register
Function 2	–	Load/clear index register
Function 6	–	Load/clear floating point accumulator bits 00 through 15
Function 7	–	Load/clear floating point accumulator bits 16 through 31
Function 8	–	Load/clear floating point accumulator bits 32 through 47
Function 3, 4	–	Load single/double precision cold-start address
Function 5, 9	–	Load restart/stop address

Write functions 3, 4, 5, and 9, when executed, cause activity to occur on the DSA bus. The HFPU uses the address loaded by these functions as a pointer in memory from which a command code is obtained. The subsequent command code may require additional DSA accesses.

Specific A/Q read commands can be sent to the HFPU so the user can access and monitor various registers within the HFPU. The following registers can be statused via A/Q read commands:

Function 0	–	Samples the function/status register
Function 1	–	Samples the command code register
Function 2†	–	Samples the index register
Function 3, 4, 5†	–	Samples the program count register
Function 6†	–	Samples floating point accumulator bits 00 through 15
Function 7†	–	Samples floating point accumulator bits 16 through 31
Function 8†	–	Samples floating point accumulator bits 32 through 47
Function 9	–	Samples the stop save address register (function/status register status at time of stop function)

† Can be sampled only when the HFPU is not in an active state (bit 15); i.e., in a condition where these registers are not in a changing state.

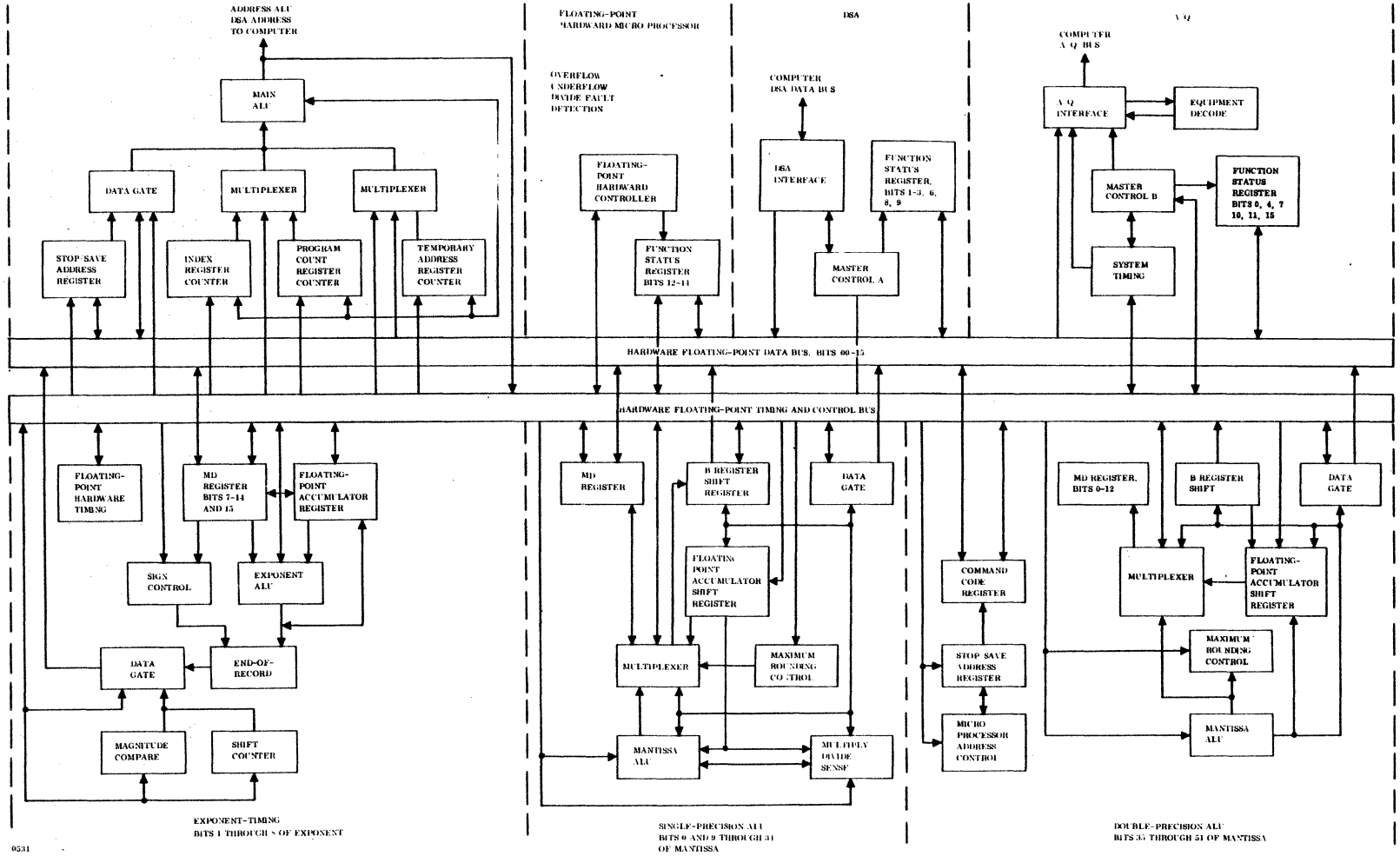


Figure 1-4. HFPU Block Diagram

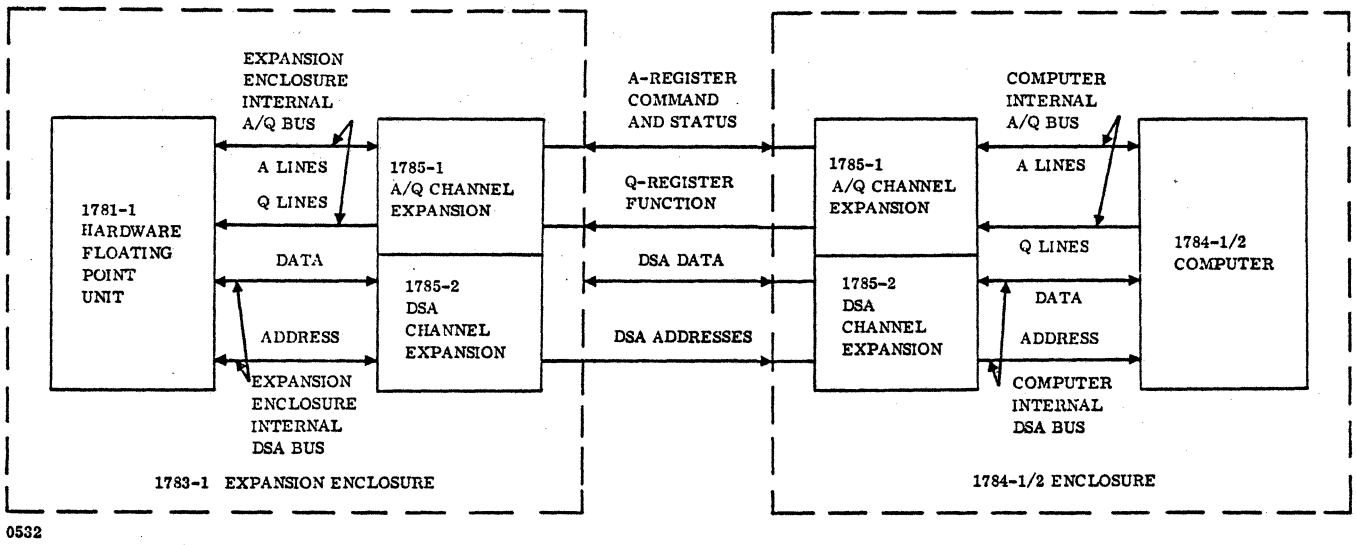


Figure 1-5. System Signal Flow

FUNCTIONAL DESCRIPTIONS

HFPU OPERATION

Under normal operation the HFPU is activated by appropriate A/Q write functions and obtains all command code instructions and data operands from System 17 memory via DSA access. It executes these command code instructions and returns the results of the operations to memory as directed. There are six registers within the HFPU that must be accessed to establish the initial condition for starting a HFPU operation via A/Q write functions. Since the HFPU is a status driven device and does not have interrupt generating capabilities, it must use these same registers to access the HFPU via specific A/Q read functions to determine the operating state of the HFPU.

FUNCTION/STATUS REGISTER

This is the main control register for the HFPU and accepts A/Q input and certain output commands at any time. If active, the HFPU accepts an A/Q channel write to function/status register only if A bit 00 (PCLR) is set. Any other A bits are ignored. Contents of this register during an A/Q read reveals the operating conditions of the HFPU.

COMMAND CODE REGISTER

This register is normally loaded via the DSA channel and contains the command code instruction word. It can be accessed by an A/Q channel read at any time but can only be loaded by an A/Q write command function 1 when the unit is not active.

INDEX REGISTER

This register contains a 16-bit digital number that is used during operand address formation for floating point calculations. It is normally loaded via the DSA channel by an INDX command. It can be read at any time by an A/Q read command when in a stable condition, but can only be loaded by a function 2 A/Q write command when the unit is not active.

PROGRAM COUNT REGISTER

This register contains a 16-bit digital number used as the base address during operand address formation. It is normally loaded via the A/Q channel by a cold start command, function 3 or 4, and incremented during floating point operations. It is also loaded via the DSA channel on a restart command.

The HFPU externally rejects any attempt to read or write the program count register while the HFPU is active and function 3 or 4 is used. The HFPU permits the program count register to be read any time it is in a stable state with a function 5 read command.

FLOATING POINT ACCUMULATOR

This register is the main arithmetic register in the HFPU. It is 32 bits wide for single precision and 48 bits wide for double precision. (See section 5 for the floating point accumulator format.) The floating point accumulator can be accessed via A/Q function 6, 7, or 8 channel writes or reads or via the DSA by any of several command codes. The HFPU externally rejects any write or read function to the floating point accumulator via the A/Q channel if the HFPU is active.

STOP SAVE ADDRESS REGISTER

This register contains a 16-bit digital number used as an absolute address for the starting memory location of where to save the content of the HFPU registers when a stop function is issued. It is addressable only by the A/Q channel. The HFPU accepts a stop save address register command at any time if that command is protected. The HFPU returns an external reject to the CPU if the stop save address register to be read with a function 9 read command at any time.

TYPES OF HFPU OPERATION

There are three methods used to initialize the HFPU.

- Cold start – Single precision mode
- Cold start – Double precision mode
- Protected restart – Single or double precision mode

Each type of cold start uses a unique function address. A restart function is used when re-entering a calling sequence after the HFPU has been interrupted by a stop function to service a higher priority routine.

COLD START SEQUENCE

A cold start sequence is initiated as follows. Function 0 is used to load the function/status register with a program master clear (PCLR) or to establish the operating condition of the HFPU before issuing a cold start function. Other bits that may be selected (not included with the program master clear bit) in the function status register are those that

select the DSA scanner access mode (bits 1 and 2), DSA protect (bit 4), double precision (bit 7), and relative addressing (bit 9). The format used for the function status register is depicted in figure 4-2 and is defined in table 4-2. If no special set-up is required, the program count register is loaded from the CPU A register by an A/Q write function 3 or 4. If the A/Q write is function 3, the unit starts in single precision mode. If the A/Q write is function 4, the unit starts in double precision mode and sets bit 7 in the function status register. Either of the cold start functions clears the index register and function status register bits 3, 6, 8, 9, 10, and 11. The address (contents of the A register when outputting function 3 or 4) is transferred to the program count register as the address of the first command code instruction word. When the HFPU accepts the starting address word, it goes into an active state (bit 15 of the function status register is set) and loads the command code register via the DSA channel. The unit remains in an active state until it either executes a FEND instruction, receives a stop function, or receives an A/Q write function 0 with A-bit 00 equal to 1 (PCLR).

STOP/RESTART SEQUENCE

A protected stop function may be issued at any time while the HFPU is in an active state. The HFPU rejects an unprotected stop function regardless of the setting of the HFPU A/Q protect bit jumper plug. A stop function is accomplished by the following sequence of events:

1. An A/Q write function 9 where the CPU A-register is transferred to the stop save address register as the stop and save address.

2. As soon as the HFPU completes its present arithmetic operation, it uses the contents of the stop save address register as the absolute address in CPU memory of where to start storing the contents of the following registers:

Stop save address register	=	The contents of the function/status register (see figure 4-2 for description/bit assignments)
Stop save address register + 1	=	The contents of the command code register†
Stop save address register + 2	=	The contents of the index register
Stop save address register + 3	=	The contents of the program count register
Stop save address register + 4	=	The contents of the floating point accumulator, bits 00 through 15
Stop save address register + 5	=	The contents of the floating point accumulator, bits 16 through 31
Stop save address register + 6	=	The contents of the floating point accumulator, bits 32 through 47 (only when operating in double precision mode)

3. When the HFPU has completed storing the last register, it goes inactive and clears bit 15 of the function/status register. A stop function issued while the HFPU is inactive causes the HFPU to go active (bit 15 of function/status register set) for the time required to store the six/seven registers into CPU memory. The HFPU returns to the inactive state (bit 15 of function/status register clear) upon completion of the retrieval operation.

After a stop function is issued, the HFPU may be restarted by a protected restart function. The HFPU rejects an unprotected restart function regardless of the setting of the HFPU A/Q protect bit jumper plug. A restart function is an A/Q write function 5, where the contents of the CPU A register is transferred to the stop save address register and the following events take place:

1. The HFPU goes to an active state and bit 15 of the function/status register is set.

2. The HFPU uses the stop save address register contents as an absolute starting address in memory of where to start the retrieval of the register contents saved on the receipt of the stop function in the following manner:

   The contents of the SSAR is used to restore the function/status register.

   The contents of the SSAR+1 is used to restore the command code register.

   The contents of the SSAR+2 is used to restore the index register.

---

† The command code register format reflects the current status of the command code word; that is, bits 15 through 12 contain the next command code to be executed. For example:

1. Comment code register is read from CPU:

Code 1	Code 2	Code 3	Code 4

2. Command code has been performed or is being performed at the time of the stop function.

3. Command code register is stored on the stop function:

Code 1	Code 2	Code 3	Code 4

The contents of the SSAR+3 is used to restore the program count register

The contents of the SSAR+4 is used to restore the floating point accumulator (bits 00 through 15).

The contents of the SSAR+5 is used to restore the floating point accumulator (bits 16 through 31).

The contents of the SSAR+6 is used to restore the floating point accumulator (bits 32 through 47).

3. When the HFPU registers are restored, the unit continues to execute command codes where it left off if the active bit in the restored function/status register (bit 15) is set. If this bit is not set, the HFPU goes to a not active or idle state.

PROTECT FEATURE

The HFPU incorporates an A/Q protect feature and a DSA protect feature. The A/Q protect feature consists of a jumper plug on the A/Q interface board. The presence of a jumper plug is defined as protected mode. The absence of a jumper plug is defined as unprotected mode.

When the HFPU is set to protect mode, it sets function/status register bit 4, accepts only protected A/Q write functions, and causes an external reject to the CPU for any unprotected A/Q write function it receives. When the HFPU is set to unprotected mode, it accepts all legitimate A/Q input/output functions. Unprotected stop functions and unprotected restart functions are defined as illegal.

The DSA protect mode feature is activated by setting bit 4 in the HFPU function/status register. This bit is set by four methods:

- A protected A/Q write function 0 (A to function/status register) with A-register bit 4 set
- A protected A/Q write function 3 or 4 (cold start single precision or double precision)
- A protected A/Q write function 9 (stop)
- The A/Q protect jumper

NOTE

The above three A/Q write functions must be protected to set function/status register bit 4 regardless of the A/Q protect jumper position.

When the DSA protect mode is active, it allows the HFPU to write data words or store register contents into protected memory locations without incurring program protect errors. Function/status register bit 4 stored in memory during the stop function reflects the DSA protect state of the HFPU before execution of the stop function. When the DSA protect mode is active (function/status register bit 4 set), all unprotected A/Q write functions are rejected.

LOOK-AHEAD BUFFER

The HFPU incorporates a look-ahead buffering feature. This feature allows the HFPU to execute succeeding command codes as soon as it begins a floating point calculation. Any command code that does not interfere with the floating point accumulator can be executed to completion. For example, a branch index command executes to completion while a branch accumulator command waits until the floating point accumulator completes the current calculation before making the decision to branch or not to branch.

Commands that load operands into the floating point accumulator load those operands into a holding register and then wait for the floating point accumulator to complete the current calculation before continuing. FLOF and FLST commands perform any necessary address modification and then wait for the floating point accumulator to complete the current calculation. FLOF or FLST do not activate DSA priority in block mode until they are able to store the operands. The FEND command does not terminate floating point accumulator activity but waits for the floating point accumulator to complete the current calculation before setting function/status register bit 6 and clearing function/status register 15.

ARITHMETIC ROUNDING RULES

The HFPU employs the following rules for rounding off the results of an arithmetic operation. These rules are implemented after normalization of the result of the arithmetic process.

NOTE

These rules are used as a guide so that hardware arithmetic results agree with software arithmetic results.

In a FADD or FSUB operation, at least four binary bits of residue of the adjusted or shifted coefficient are retained. The round-off rules are as follows:

- If the adjusted coefficient is negative and the amount of the adjusted residue is greater than -7, round off the results by adding -1.
- If the adjusted coefficient is positive and the amount of the adjusted coefficient residue is greater than +7, round off the results by adding +1.
- In all other cases, do not round off.

In a FDIV operation, the remainder at the end of the FDIV process should be tested for rounding off with the following rule:

$$FDIV = \left|\frac{A}{B}\right| = \left|C\right| + \left|R\right|$$

If $R \geq \left|\frac{B}{2}\right|$, round up by one binary digit.

If $R < \left|\frac{B}{2}\right|$, do not round the results.

In a FMPY operation, the partial product left at the end of the arithmetic process should be tested for the rounding off with the following rule:

In $\left|\text{partial product}\right| > 7$, round up by one binary digit.

In any other case, do not round the results.

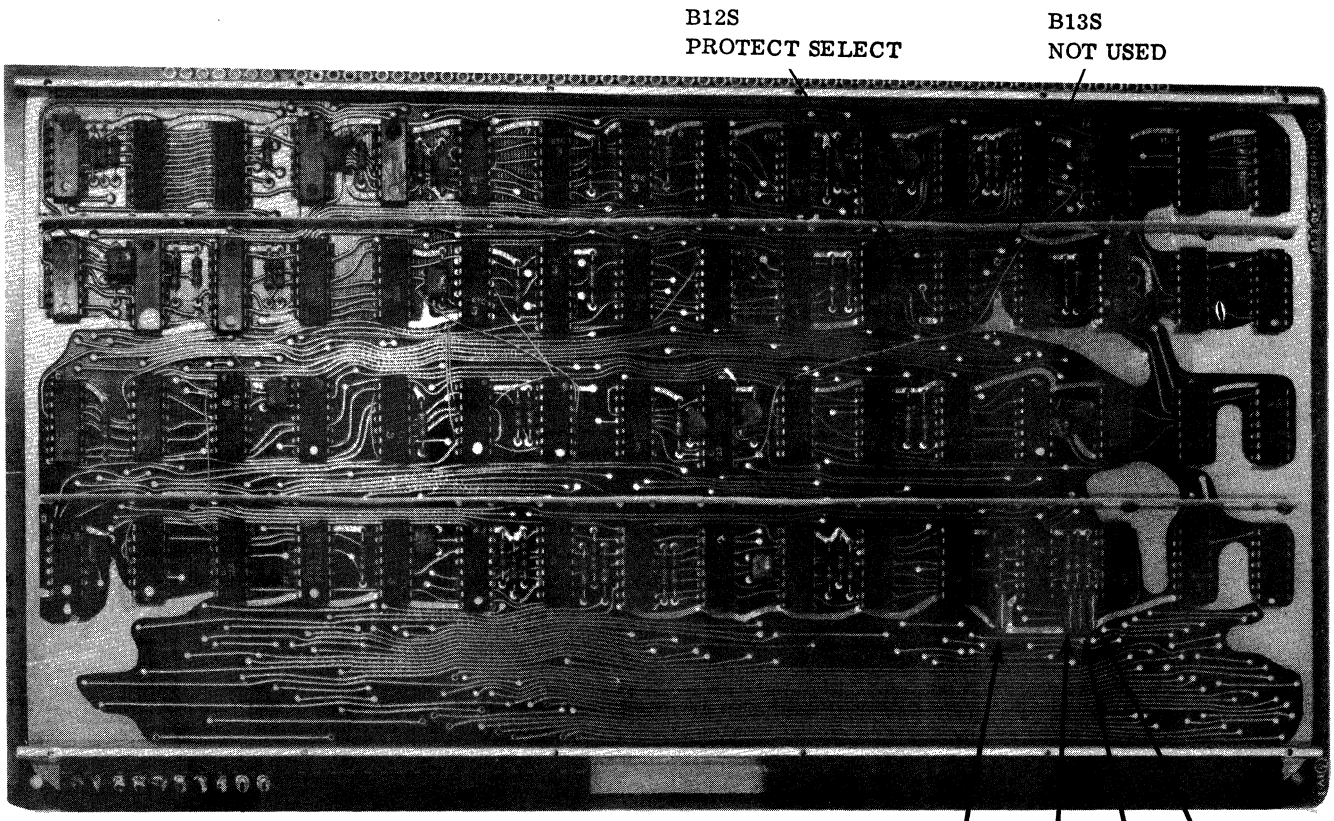
A/Q JUMPER INSTALLATION

The HFPU does not have any external operator controls, switches, or indicators. The only operator requirements that are necessary involve the proper setting of the equipment number on A/Q board position 21 when installed. Refer to figure 3-1 to locate the equipment select switches Q10 through Q07 and the HFPU protect jumper. Once it has been determined which equipment number is to be selected, use tables 3-1 and 3-2 to insert the appropriate jumpers.

DSA JUMPER INSTALLATION

The DSA card, position 22, must have the scanner select jumpers installed properly. When determining which scanner position to select (ONLY, MIDDLE, FIRST, LAST, or OUT) it is necessary to take into consideration the other equipments that are on the system DSA daisy chain. See figure 3-2 for the locations of the scanner select jumpers and table 3-3 to ensure that the correct jumper is installed.

The backplane scanner connections must also be installed correctly to ensure proper DSA operation (see table 3-4). The backplane scanner connection includes the HFPU in the total system scanner (see figure 3-3). Ensure that the HFPU cards and the backplane connector assemblies are installed correctly. The only way the HFPU can be operated is by software. Either the software operating system, HFPU diagnostic, or machine language programs must be used to exercise the HFPU.



NOTES: 1. PROTECT JUMPER IN = DEVICE IS PROTECTED.
2. JUMPER IN SELECTS Q-BIT AS FOLLOWS:
E13S JUMPER IN = Q07
E14S1 JUMPER IN = Q10
E14S2 JUMPER IN = Q09
E14S3 JUMPER IN = Q08

Figure 3-1. A/Q Board: HFPU Equipment Select and Protect Jumper Location

TABLE 3-1. EQUIPMENT NUMBER SELECT AND PROTECT SELECT

Jumper Position	Jumper In Selects Q Bit
E13S	Q7
E14-S3	Q8
E14-S2	Q9
E14-S1	Q10
B12-S	Protect†

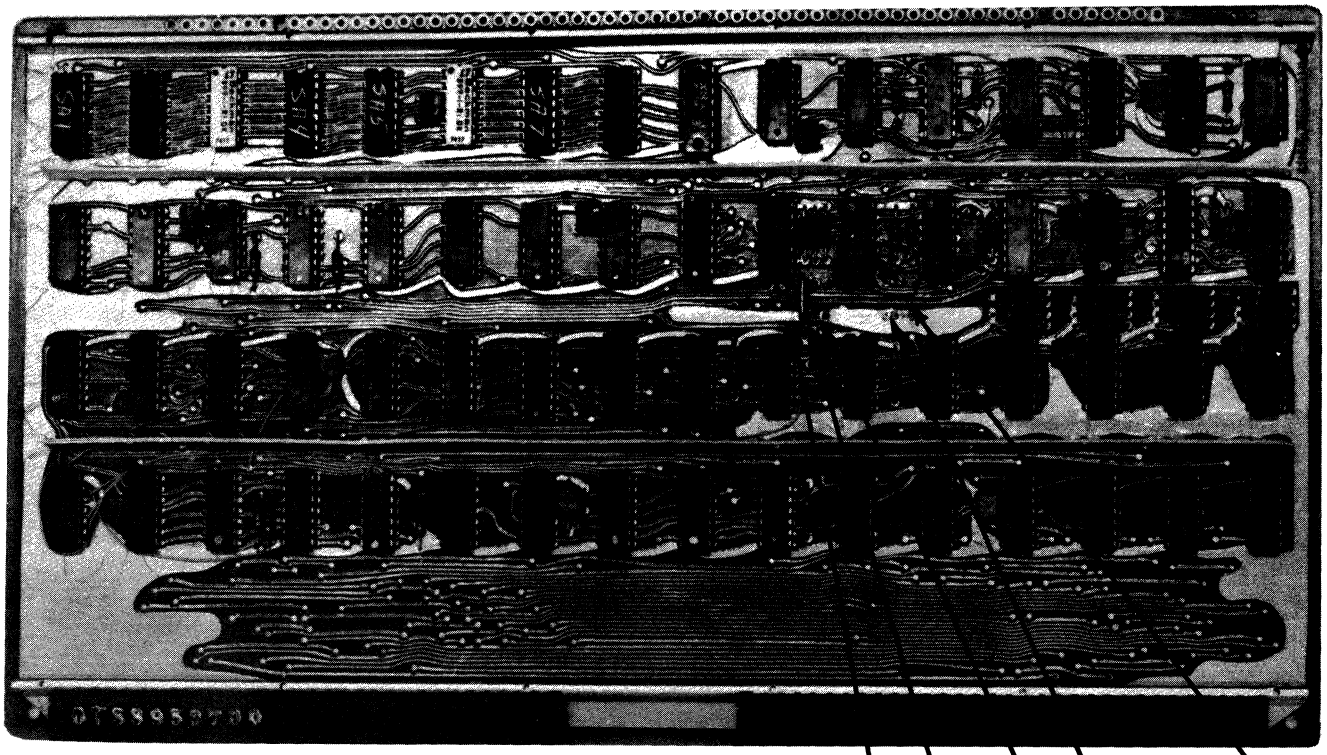
†Jumper in selects protect.

TABLE 3-2. HEXADECIMAL CODE FOR EQUIPMENT SELECTION CODE

Links		E14S1 Q10	E14S2 Q09	E14S3 Q08	E13S Q07
Hexadecimal Code (Q10 through Q7)	0	0	0	0	0
	1	0	0	0	1
	2	0	0	1	0
	3	0	0	1	1
	4	0	1	0	0
	5	0	1	0	1
	6	0	1	1	0
	7	0	1	1	1
	8	1	0	0	0
	9	1	0	0	1
	A	1	0	1	0
	B	1	0	1	1
	C	1	1	0	0
	D	1	1	0	1
	E	1	1	1	0
Normal equipment code	F	1	1	1	1

NOTES:

1. A 1 in the binary code indicates the presence of a jumper plug for the setting of the equipment code; 0 its absence.
2. The equipment code is normally F_{16}.



NOTE: JUMPER IN SELECTS SCANNER POSITION AS FOLLOWS:

C11S1 JUMPER IN = LAST
C11S2 JUMPER IN = MIDDLE
C11S3 JUMPER IN = OUT
C12S1 JUMPER IN = ONE ONLY
C12S2 JUMPER IN = FIRST

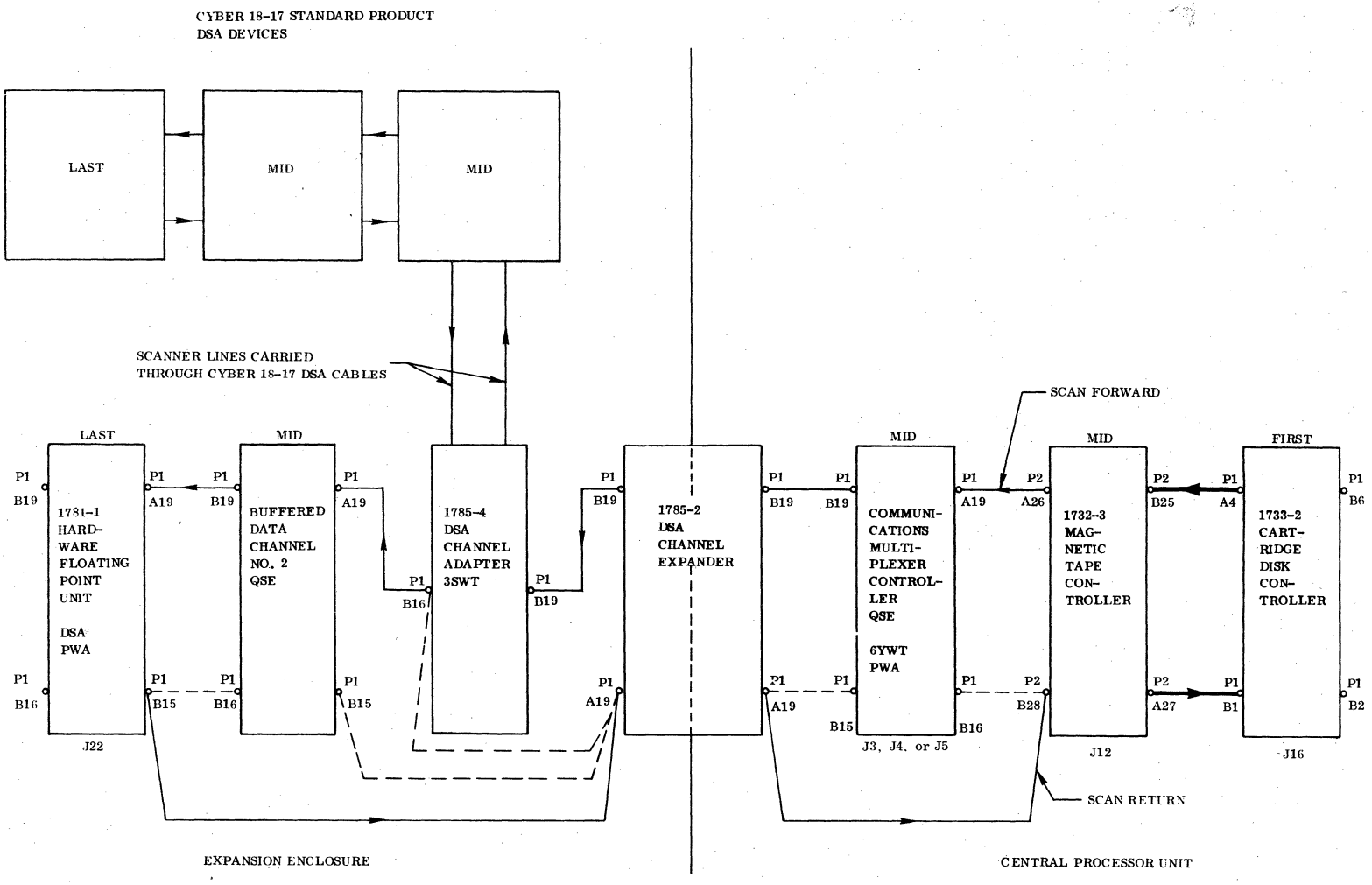
Figure 3-2. DSA Board: HFPU Scanner Jumper Location

TABLE 3-3. DSA SCANNER POSITION SELECT

Jumper Position	Scanner Position
C11-S1	LAST
C11-S2	MIDDLE
C11-S3	OUT
C12-S1	ONLY
C12-S2	FIRST
NOTE: The presence of a jumper selects the position.	

TABLE 3-4. BACKPLANE SCANNER CONNECTION

Scanner	HFPU	DSA Channel Expansion
Scan forward	P1A19	P1B19
Scan Return	P1B15	P1A19



NOTES: 1. HEAVY SOLID LINES – JUMPER WIRE ALREADY HARDWIRED ON THE CPU BACKPLANE.
2. DOTTED LINES – OPTIONAL JUMPER WIRE IN PLACE OF THE JUMPER WIRE SHOWN WITH THE SOLID LINES. ONE OR THE OTHER CONFIGURATION COULD BE USED.
3. DSA DEVICES OPERATING ON THE DSA CHANNEL ADAPTER OCCUPY ONLY MIDDLE OR LAST POSITIONS ON THE SCANNER.

0154

Figure 3-3. DSA Scanner Configuration

FUNCTION/EQUIPMENT SELECT

Function codes are transmitted via the A/Q channel. Bits set in the lower portion (0 through 3) of the Q register define the contents of the A register (figure 4-1). Bits 7 through 10 must match the equipment number setting on the HFPU A/Q board. The remaining bits of Q are ignored (should be 0). The function codes are listed in table 4-1. An A/Q channel read or write signal indicates an input or output, respectively, to the A instruction. The codes and resulting operations are described in this section.

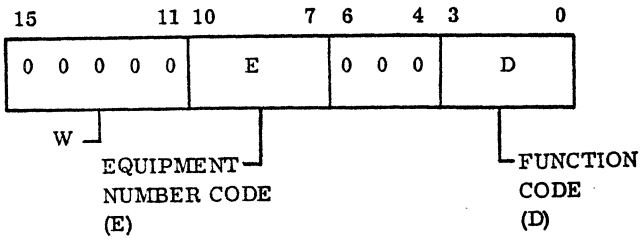


Figure 4-1. Q Register Format

FUNCTION/STATUS REGISTER

The function/status register (FSR) provides a means of presetting the desired operating conditions within the HFPU; monitoring the activity of the HFPU while it is executing a calling sequence consisting of one or more command codes, and changing operating conditions of the HFPU while executing a calling sequence. Addressing and operating mode conflicts must not be created when changing the operating conditions of the HFPU while it is executing a calling sequence. The function/status register format is shown in figure 4-2. Table 4-2 defines each bit position in the function/status register.

NOTE

The console/program master clear clears all HFPU timing, resets the HFPU to an idle state, and clears all registers with the exception of the program count register and the floating point accumulator. The console master clear enters the HFPU via a pin on the A/Q channel bus.

OPERATION COMMAND CODE

The HFPU recognizes 16 unique command codes in its command code register. Command code 0 is recognized as a special 2-byte command code; that is, the next byte is a special command to be executed. This increases the number of available command codes to 31, of which 25 are used in the HFPU (see table 4-3). These command codes are described in detail in table 4-4. After the HFPU is activated, it responds to an established calling sequence. A calling sequence may consist of one or more of these command codes. The last command code in a calling sequence should normally be a FEND.

HARDWARE EXECUTION TIMES

Table 4-5 lists the HFPU command code execution times.† These execution times assume the HFPU is in an active state and include the instruction decode time, required operand address preparation time, time required to execute the instruction to completion, and time to advance the byte counter to start the next instruction decode. The memory cycle access time is included in these execution times. The memory cycle time period is defined as:

- The access time to retrieve the pointer word (900 nanoseconds total; 600 nanoseconds memory cycle plus 300 nanoseconds to generate effective operand address)
- The access time to retrieve or store the operand
- A 257-nanosecond average time for retrieval of a command code HFPU internal request time plus time for DSA access for 600 nanoseconds memory divided by four command codes per word.

NOTE

Total hardware execution time includes basic hardware execution time plus DSA access for 600 nanoseconds memory plus command code bias time of 257 nanoseconds.

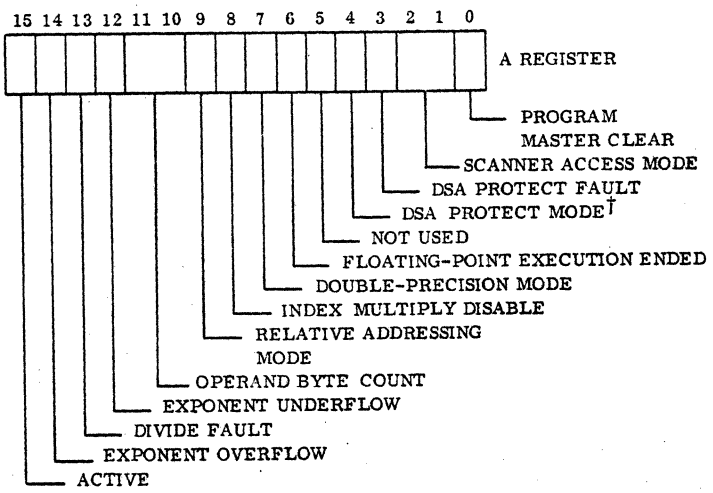
Memory access time assumes no DSA scanner delays or memory refresh cycles.

Times are nominal and assume HFPU is in hog mode.

† All times are based upon use of a 1784-2 (600 nanoseconds) CPU. If a 1784-1 (900 nanoseconds) CPU is used, is is necessary to add 300 nanoseconds to the noted execution times for each memory cycle included.

TABLE 4-1. FUNCTION CODES

Function Decode Value Set in Q (Bits 03 through 00)	Output from A	Input to A
0000	Load function/status register	Function/status register contents
0001	Load command code register†	Command code register contents
0010	Load index register†	Index register contents†
0011	Load single precision cold start†	Program counter register (if not active)†
0100	Load double precision cold start†	Program counter register (if not active)†
0101	Load restart address into stop/start address register†	Program counter register†
0110	Load floating point accumulator (00 through 15)†	Floating point accumulator contents 00 through 15†
0111	Load floating point accumulator (16 through 31)†	Floating point accumulator contents 16 through 31†
1000	Load floating point accumulator (32 through 47)†	Floating point accumulator contents 32 through 47†
1001	Stop save address register	Stop save address register status



†Bit is always set if A/Q protect jumper is installed.

Figure 4-2. Function/Status Register Format

TABLE 4-2. FUNCTION/STATUS REGISTER BIT ASSIGNMENT

Bit Position	Bit Mnemonic	Bit Definition
15	Active	Is set by the A/Q Channel write function 0 to the function/status register with A bit 15 set (the HFPU must be inactive) or by the HFPU when it is in an active state. When this bit is set, it causes the HFPU to reject all A/Q channel write functions except the A register to the function/status register and protected stop (A Register to stop save address register). The bit is cleared or reset by: Inactive HFPU status Program master clear Console master clear Inactive status does not necessarily indicate that the HFPU has completed the command code operation, as stop function 9 clears function/status register bit 15 after storing all appropriate registers. Function/status register bit 15 stored at the stop save address register during the stop function reflects the condition of the HFPU before the stop function.
14	OVFL	Exponent overflow. The bit is set by: • HFPU arithmetic operation in which the result was too large to be represented by the eight binary bits. When the bit is set as a result of an arithmetic operation, the HFPU force-sets the floating point accumulator to the largest floating-point number expressible with the correct floating point sign. • Function/status register function 0 (HFPU inactive) from CPU and A bit 14 equals 1. This action sets only this bit and does not affect the contents of the floating point accumulator. Bit is reset by: • Function/status register function 0 (HFPU inactive) from CPU and A bit 14 equal to 0. • Program master clear • Console master clear
13	DVFL	Divide fault Bit is set by: • HFPU when an attempt is made to divide by a zero or by an un-normalized operand. When set as a result of an arithmetic operation, the HFPU force-sets the floating point accumulator to the largest floating point number expressible with the correct floating point sign. • Function/status register function 0 (HFPU inactive) from CPU and A bit 13 equal to 1. This action sets only this bit and does not affect the contents of the floating point accumulator. The bit is reset by: • Function/status register (HFPU inactive) function 0 from CPU and A bit 13 equal to 0. • Program master clear • Console master clear

TABLE 4-2. FUNCTION/STATUS REGISTER BIT ASSIGNMENT (Contd)

Bit Position	Bit Mnemonic	Bit Definition
12	UNFL	Exponent underflow The bit is set by: • HFPU arithmetic operation in which the result was too small to be represented by the eight binary bits. When set as a result of an arithmetic operation, the HFPU force-sets the floating point accumulator to 0. • Function/status register function 0 (HFPU inactive) from CPU and A bit 12 equal to 1. This action sets only the bit and does not affect the contents of the floating point accumulator. The bit is reset by: • Function/status register (HFPU inactive) function 0 from CPU and A bit 12 equal to 0. • Program master clear • Console master clear
11, 10	OPBC	Operand byte count Indicates which of the four command codes in the command code register is under execution or about to be executed. It has the following bit format: Bit 11 / Bit 10 0 0 Operand byte 1. 0 1 Operand byte 2. 1 0 Operand byte 3. 1 1 Operand byte 4. These bits can be set to any initial state by a function/status register (HFPU inactive) function 0 from the CPU and A bits 11 and 10. Bits are reset by: • Function/status register (HFPU inactive) function 0 from CPU and A bits 11 and 10 set to 0 • Cold start command • Program master clear • Console master clear A/Q Write Function 1 (A Register to command code register does not affect the state of function/status register bits 11 and 10.
9	RELM	Relative addressing mode This bit, when set, selects relative addressing mode. Bit is set or reset by: • The HFPU execution of a CHMD instruction. (See table 4-2 for a detailed explanation.) • Function/status register (HFPU inactive) function 0 from the CPU and the state of A bit 09. The bit is cleared by: Cold start command Program master clear Console master clear

TABLE 4-2. FUNCTION/STATUS REGISTER BIT ASSIGNMENT (Contd)

Bit Position	Bit Mnemonic	Bit Definition
8	INXM	Index multiply disable This bit, when set, is used to inhibit the logic that multiplies the index register contents by two or three during effective address formation. • Function/status register (HFPU inactive) function 0 from the CPU and A bit 08 set to 1. The bit is reset by: • Function/status register (HFPU inactive) function 0 from CPU and A bit 8 set to 0. • Cold start command • Program master clear • Console master clear A/Q write function 2 (A register to index register does not affect the state of function/status register bit 08.
7	DBPM	Double precision mode Bit is set by a function/status register function 0 (HFPU inactive) and A bit 7 set or by a cold start function 4 in double precision. When set, all floating point calculations are performed in double precision mode (48 bits). When reset, all floating point calculations are performed in single precision mode (32 bits). The bit is reset by: • Program master clear • Console master clear • Cold start function 3 in single precision
6	FEND	Floating point execution ended The bit is set by: • HFPU execution of a FEND instruction • Function/status register (HFPU inactive) function 0 from the CPU and A bit 06 set to 1 The bit is reset by: • Function/status register (HFPU inactive) function 0 from the CPU and A bit 06 set to 0 • Cold start command • Program master clear • Console master clear
5	UNUSED	Bit is always reset.
4	PROT	Protect mode When set, it places the HFPU in a protected device mode. This mode allows the HFPU to write into protected memory locations via the DSA channel. The bit is set by a protected A register to function/status register function 0 from the CPU and A bit 04 set to 1, by a protected A/Q cold start function 3

TABLE 4-2. FUNCTION/STATUS REGISTER BIT ASSIGNMENT (Contd)

Bit Position	Bit Mnemonic	Bit Definition
4 (Contd)	PROT	or 4, by a protected A/Q stop function 9, or by the presence of the A/Q protect jumper. The bit is reset by: • An unprotected A register to function/status register function • An unprotected A/Q cold start function • Program master clear • Console master clear Function/status register bit 4 stored at the stop save address register during the stop function reflects the DSA protect mode of the HFPU before the stop function.
3	PTFT	Protect fault. When set, it indicates that the HFPU was not in protect mode and made a write data access to a protected memory location. The bit is also set or reset by a function/status register (HFPU inactive) function 0 from the CPU and the state of A bit 03. Bit is also reset by: • Cold start function • Program master clear • Console master clear
2, 1	SCAN	Scanner access mode The state of these bits selects or indicates one of three modes of HFPU DSA channel accesses. These modes are: Bit 2 Bit 1 0 0 Block The HFPU stops the scanner for up to five successive memory cycles during a command code word fetch. The HFPU does not release the scanner before determining if the first command code byte of that word requires memory. If the first command requires memory, the HFPU holds the scanner and access memory to fetch the address pointer word and one, two, or three operands. If the first command byte does not require memory or is a branch accumulator command and the floating point accumulator is active, the HFPU releases the scanner. In either case, the second, third, and fourth command bytes that require memory must wait for the scanner to return to the HFPU. These bytes can hold the scanner for up to four memory cycles. Block mode activates (first access) or maintains (second through fifth access) the DSA PRIORITY signal for all memory accesses subject to restrictions found elsewhere in this manual. 0 1 Hog Once the HFPU is started, the scanner is held until the HFPU executes a FEND instruction. The DSA priority signal is active from start to finish.

TABLE 4-2. FUNCTION/STATUS REGISTER BIT ASSIGNMENT (Contd)

Bit Position	Bit Mnemonic	Bit Definition
2, 1 (contd)	SCAN	Bit 2 Bit 1 1 0 Word The scanner is released after every DSA data word access. The DSA priority signal is not active. These bits are set by: • Function/status register (HFPU inactive) function 0 from the CPU with A bit 02 set to 1 and/or A bit 01 set to 1 These bits are reset by: • Function/status register (HFPU inactive) function 0 from the CPU with A bit 2 set to 0 and/or A bit 1 set to 0. • Function/status register (HFPU inactive) function 0 from the CPU with A bit 2 set to 1 and/or A bit 1 set to 1. • Program master clear • Console master clear
0	PCLR	Program master clear When HFPU receives A bit 00 set and function/status register function 0, it clears all timing, resets the unit to an idle state, and clears all registers with the exception of the program count register and the floating point accumulator. The HFPU ignores any other A bits that are set. Bit is not used on an A/Q read function.

TABLE 4-3. COMMAND CODES

Type of Code	Command Code Mnemonic	4-Bit Code	Description
Standard	FLOF	1	Float-to-fixed
	FIXF	2	Fixed-to-float
	STRI	3	Store index
	FEND	4	End of calling sequence
	CHMD	5	Change address mode
	NIDX	6	Clear index
	FCOM	7	Floating complement
	FSUB	8	Floating subtract
	FMPY	9	Floating multiply
	FDIV	A	Floating divide
	FLDD	B	Floating load
	ADDI	C	Add to index
	FLST	D	Floating store
	FADD	E	Floating add
	INDX	F	Index

TABLE 4-3. COMMAND CODES (Contd)

Type of Code	Command Code Mnemonic	4-Bit Code	Description
Special	SPEC	0	Special command code – must precede each of the following command codes before they are performed
	CACS	1	Continue another calling sequence
	BRAM	2	Branch if accumulator is negative
	BRAZ	3	Branch if accumulator is zero
	BRAN	4	Branch if accumulator is nonzero
	BRAD	5	Branch if accumulator is positive
	BRIM	6	Branch if index is negative
	BRIZ	7	Branch if index is zero
	BRIN	8	Branch if index is nonzero
	BRIP	9	Branch if index is positive
		A, B, C, D, E, F	Act as FENDS when preceded by command code of 0 (SPEC)

TABLE 4-4. OPERATION COMMAND CODE DEFINITION

Command Code Mnemonic	Description
FLOF	Float-to-fixed The contents of the floating point accumulator are converted to fixed point and the results stored at the effective operand address. Floating point accumulator bits 16 through 31 contain the fixed point number. If positive overflow occurs, floating point accumulator bits 16 through 31 contain 7FFF. If negative overflow occurs, floating point accumulator bits 16 through 31 contain 8000. The raw, unmultiplied, index value is used to form the effective address for FLOF.
FIXF	Fixed to float The contents of the effective operand address are converted to floating point and the result placed in the floating point accumulator. The raw, unmultiplied, index value is used to form the effective address for FIXF.
STRI	Store index Stores the contents of the index register at the effective operand address. Does not alter the contents of the index register. Indexed address information is inhibited during the execution of this instruction.

TABLE 4-4. OPERATION COMMAND CODE DEFINITION (Contd)

Command Code Mnemonic	Description
FEND	End of calling sequence This operation terminates the calling sequence and causes the HFPU to return to an idle state. Execution of this code sets bit 6 and clears bit 15 in the function/status register.
CHMD	Change mode All operand addresses following this operation code in the calling sequence are made relative if the preceding addresses were absolute and absolute if preceding addresses were relative. Does not affect the index register value. Sets bit 9 of the function/status register when relative mode address is in effect. No operand address is needed for this code.
NIDX	No index Clears the index register, which disables the indexing of operand addresses. No operand address is needed for this code.
FCOM	Floating complement Complements the contents of the floating point accumulator. No operand address is needed for this code.
FSUB	Floating subtract The contents found at the effective operand address are subtracted from the contents of the floating point accumulator and the results are then placed in the floating point accumulator.
FMPY	Floating multiply The contents found at the effective operand address are multiplied by the contents of the floating point accumulator and the results are placed in the floating point accumulator.
FDIV	Floating divide The contents of the floating point accumulator are divided by the contents found at the effective operand address and the results are placed in the floating point accumulator.
FLDD	Floating load The contents found at the effective operand address are loaded into the floating point accumulator. This is a normalized floating-point number.
ADDI	Add to index Adds the contents of the effective operand address to the contents of the index register and places the result in the index register. Indexed address formation is inhibited during the execution of the instruction.
FLST	Floating store The contents of the floating point accumulator are stored at the effective operand address. The contents of the floating point accumulator are not altered by this operation.
FADD	Floating add The contents found at the effective operand addresses are added to the contents of the floating point accumulator and the results are placed in the floating point accumulator.

TABLE 4-4. OPERATION COMMAND CODE DEFINITION (Contd)

Command Code Mnemonic	Description
INDX	Index The contents found at the effective operand address are loaded into the index register. The operand addresses of all subsequent FLOF, FLDD, FLST, FADD, FSUB, FMPY, FDIV, and FIXF operations are affected in the following manner: • If function/status register bit 8 is clear, the contents of the index register is multiplied by 2 when the unit is in single precision mode and the effective operand address is being formed. The contents of the index register are not changed. • If function/status register bit 8 is clear, the contents of the index register is mulitplied by 3 when the unit is in double precision mode and the effective operand address is being formed. The contents of the index register are not changed. • If function/status register bit 8 is set, the raw index register contents is added to the base address to form the effective address. • For the functions FLOF and FIXF, the raw index value is added to the base address.
SPEC	Special command code This code causes the HFPU to recognize the next byte as a code within the following branch (jump) command code subset. If the next byte is 0, a FEND is executed.
CACS	Continue another calling sequence† Starts a new floating point instruction sequence by loading the contents of the effective operand address into the command code register. The new code execution starts at command code byte 1. Indexed address formation is inhibited during the execution of this instruction.
BRAM	Branch accumulator negative† If the condition is satisfied (the contents of floating point accumulator is negative), execution continues by loading the contents of the effective operand address into the command code register. The new code execution starts at command code byte 1. Indexed address formation is inhibited during the execution of this instruction. If condition is not satisfied, the program count register is incremented by +1 before the next command code is executed.
BRAZ	Branch accumulator zero† If the condition is satisfied (the contents of floating point accumulator is 0), execution continues by loading the contents of the effective operand address into the command code register. The new code execution starts at command code byte 1. Indexed address formation is inhibited during the execution of this instruction. If condition is not satisfied, the program count register is incremented by +1 before the next command is executed.
BRAN	Branch accumulator non-zero† If the condition is satisfied (the contents of floating point accumulator is non-zero), execution continues by loading the contents of the effective operand address into the command count register. The new code execution starts at command code byte 1. Indexed address formation is inhibited during the execution of this instruction. If the condition is not satisfied, the program count register is incremented by +1 before the next command is executed.

† These command codes are executed only if the preceding byte is a SPEC code.

TABLE 4-4. OPERATION COMMAND CODE DEFINITION (Contd)

Command Code Mnemonic	Description
BRAP†	Branch accumulator positive† If the condition is satisfied (the contents of floating point accumulator is positive including positive 0) execution continues by loading the contents of the effective operand address into the command code register. The new code execution will start at command code byte 1. Indexed address formation is inhibited during the execution of this instruction. If the condition is not executed, the program count register is incremented by +1 before the next command is executed.
BRIM†	Branch index register negative† If the condition is satisfied (the contents of the index register is negative), execution continues by loading the contents of the effective operand address into the command code register. The new code execution starts at command code byte 1. Indexed address formation is inhibited during the execution of this instruction. If the condition is not executed, the program count register is incremented by +1 before the next command is executed.
BRIZ†	Branch index register zero† If the condition is satisfied (the contents of the index register is 0), execution continues by loading the contents of the effective operand address into the command code register. The new code execution starts at command code byte 1. Indexed address formation is inhibited during the execution of this instruction. If the condition is not executed, the program count register is incremented by +1 before the next command is executed.
BRIN†	Branch index register non-zero† If the condition is satisfied (the contents of the index register is non-zero), execution continues by loading the contents of the effective operand address into the command code register. The new code execution starts at command code byte 1. Indexed address formation is inhibited during the execution of this instruction. If the condition is not executed, the program code register is incremented by +1 before the next command is executed.
BRIP†	Branch index register positive† If the condition is satisfied, (the contents of the index register is positive), execution continues by loading the contents of the effective operand address into the command code register. The new code execution starts at command code byte 1. Indexed address formation is inhibited during the execution of this instruction. If the condition is not executed, the program count register is incremented by +1 before the next command is executed.

†These command codes are executed only if the preceding byte is a SPEC code.

TABLE 4-5. HARDWARE EXECUTION TIMES

Command Code	Maximum Total Single Precision Hardware Execution Time in Microseconds	Maximum Total Double Precision Hardware Execution Time in Microseconds	Memory Access Required Single Precision/ Double Precision
FEND	0.450	0.450	0/0
CHMD	0.450	0.450	0/0
NIDX	0.450	0.450	0/0
FCOM	0.955	0.955	0/0
FSUB	8.565	10.925	2/3
FMPY	11.425	15.545	2/3
FDIV	11.865	15.985	2/3
FLDD	3.835	4.435	2/3
FLST	2.535	3.135	2/3
FADD	8.567	10.925	2/3
INDX	1.925	1.925	1/1
STRI	1.925	1.925	1/1
ADDI	1.925	1.925	1/1
SPEC	0.450	0.450	0/0
FIXF	6.585	6.585	1/1
FLOF	4.870	4.870	2/3
CACS	1.090	1.090	1/1
BRAM	1.480	1.480	1/1
BRAZ	1.480	1.480	1/1
BRAN	1.480	1.480	1/1
BRAP	1.480	1.480	1/1
BRIM	1.480	1.480	1/1
BRIZ	1.480	1.480	1/1
BRIN	1.480	1.480	1/1
BRIP	1.480	1.480	1/1

Programming the HFPU requires a minimum of instructions. Each register may be loaded and statused by means of the following program. Only the lower four bits of the Q register need to be changed. Using this program all registers can be preloaded with data and addresses to allow restart function 5 to be executed.

$0000	E000	Load Q register
$0001	W,EQ,FCN	Equipment number and desired function †
$0002	C000	Load A register
$0003	(Data)	Desired data
$0004	0B00	No operation
$0005	03FE	Output
$0006	0B00	No operation
$0007	02FE	Input (monitor the A register, contents should equal that sent out – location 0003
$0008	0000	Selective stop

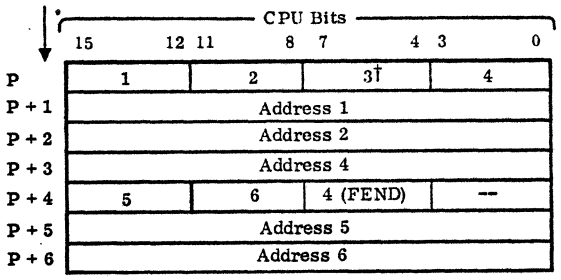
A typical program to cause the HFPU to go into execution would consist of the following:

$0000	E000	
$0001	( )	Equipment number plus function 0 (if the equipment number is F then use 0780)
$0002	C000	
$0003	0001	Function/status word – program clear
$0004	0B00	
$0005	03FE	
$0006	C000	
$0007	0084	Function/status word; double precision, word mode
$0008	0B00	
$0009	03FE	
$000A	E000	
$000B	( )	Equipment number and function 4 double precision cold start = (0784)
$000C	C000	
$000D	( )	Starting address of calling sequence
$000E	0B00	
$000F	03FE	
$00010	0000	Selective stop or include status/check loop for FEND status

CALLING SEQUENCE GENERATION

A basic calling sequence consists of an instruction word consisting of four commands, followed by the operand address (address pointers). The left-most 4-bit byte is the first operation; the operand addresses, if they are required, follow in the same order as the operation bytes, one word per byte. As many bytes may exist as desired, but the terminating byte must be a 4, the operation FEND.

CPU WORD LOCATIONS



† Does not require a memory access, thus no address 3 exists.

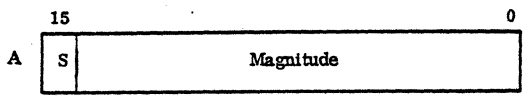
† When functions 3, 4, and 5 are executed, they should be accompanied by a legitimate calling sequence since they cause the HFPU to use the contents of the A register as a pointer in memory from which to obtain its calling sequence. If the command codes are not controlled, other areas in memory could be destroyed.

The addresses are the location in memory where operands for the respective command codes are stored. Not all operations require memory access; for example, command code 3 does not have a corresponding A3. The following command codes do not require pointers or subsequent operands: CHMD, FEND, NIDX, and SPEC. The following is an example of a calling sequence for a single precision operation. The double precision calling sequence is the same except that the pointers' values must be increased by one. That is, the data occupies three memory locations instead of two when doing double precision. Assume the starting address to be $0100.

$0100	BF7A	FLDD, FADD, FCOM, FDIV
$0001	1000	Pointer for data to be loaded into accumulator
$0002	1002	Pointer for data to be added to accumulator value
$0003	1004	Pointer for data to be divided into accumulator value
$0004	7D44	FCOM, FLST, FEND, FEND
$0005	1006	Pointer for location in which to store the results; i.e., the content of the accumulator
$1000	xxxx	Floating load data
$0001	xxxx	
$0002	yyyy	Floating add data
$0003	yyyy	
$0004	zzzz	Floating divide data
$0005	zzzz	
$0006	( )	Floating store results
$0007	( )	

FIXED/FLOAT NUMBER CONVERSIONS

The integer (fixed) number format is:



Where: S is the sign of the integer number.

0 is a positive number, and

1 is a negative number with the magnitude in ones complement form.

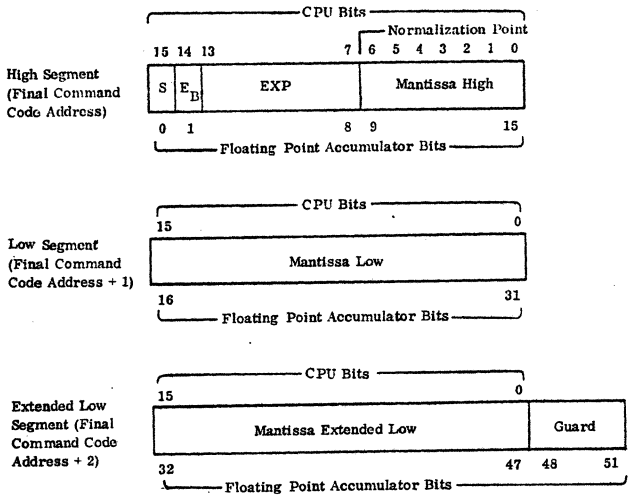
The float-to-fixed operation is performed by executing command code 1, which converts the floating point number in the floating point accumulator register to an integer and transfers the integer to the effective operand address. Floating point accumulator 31 through 16 also contains the result. If positive overflow occurs, floating point accumulator 31 through 16 will contain 7FFF and if negative overflow occurs, floating point accumulator 31 through 16 contains 8000. The fixed-to-float operation is performed by executing command code 2, which loads an integer number into the HFPU from the specified operand address, begins a conversion process, and upon completion, places the converted number into the floating point accumulator. This number may be retrieved in two ways:

- Reading status of the HFPU floating point accumulator register by successive A/Q read commands to functions 6, 7, and 8.
- Executing a FLST instruction to a specified memory location.

FLOATING POINT ACCUMULATOR FORMATTING

This section is useful in forming data into floating point accumulator format and gives examples of the different addressing methods used by the HFPU.

Floating point numbers used in the arithmetic operations have the following format:



Where: S is the sign bit of the entire floating point number. When the sign bit is 0, the floating point number is positive. When the sign bit is 1, the floating point number is negative.

E_B is the exponent sign bit, biased by an exclusive OR with 80_{16} during packing and unpacking.

EXP is the seven binary bits that represent the magnitude of the exponent ($-127 \leq$ exponent ≤ 127).

Mantissa is the normalized magnitude of the floating point number which is a fractional coefficient. A normalized positive coefficient has the form ($0.1xxx...x_{LOW}$) where S is 0. A normalized negative coefficient has the form ($0.0xxxx...x_{LOW}$) where S is 1.

A single precision number has the expressable number range:

$$-2^{127}\ (1-2^{-23})\ \times\ 2^{127}\ (1-2^{-23})$$

A double precision number has the expressable number range:

$$-2^{127}\ (1-2^{-39})\ \times\ 2^{127}\ (1-2^{-39})$$

When the floating point number is negative, the entire floating point accumulator including the exponent is in ones complement form. A floating point zero is represented as all bits set to 0. The floating point number is always normalized for any floating point arithmetic operation including FLST and FLDD.

The extended low segment of the operand is used for double precision mode.

OPERAND ADDRESSING

All operand addresses used within the HFPU conform to one of the following methods:

- Absolute (16 bits)
- Relative (16 bits with bit 15 equal to sign)
- Indexed (16 bits) – Value in index register is multiplied by 2 for single precision operations and by 3 for double precision operations if function/status register bit 8 is not set.
- Relative indexed (index handling as for indexed mode)

Table 5-1 depicts the address methods.

If function/status register bit 9 is set, relative addressing mode is in effect. If function/status bit 9 is clear, absolute addressing is in effect. Absolute addressing means that the pointer word is in an absolute address; conversely, relative addressing means that the pointer word is a 16-bit signed displacement from the current program count register.

If function/status register bit 8 is clear, the contents of the index register are multiplied by 2 or 3 and added to the argument address (pointer word) to obtain the final address. If function/status register bit 8 is set, the contents of the index register is added to the argument address to obtain the final address.

HFPU COMMAND CODE/FORTRAN CORRELATION

The following examples show the relationship of HFPU command codes to their use in setting up FORTRAN arithmetic statements. See table 5-2.

The calling sequence for FORTRAN expression A = B+CXD is as follows:

```
B9ED      (FLDD, FMPY, FADD, FLST)
 D        Address of D
<C>       Address of C
<B>       Address of B
<A>       Address of A
4000      (FEND, . . . .)
```

The calling sequence for FORTRAN expression A(I) = B (J) + C (K) * D(L) is as follows:

```
FBF9      (INDX, FLDD, INDX, FMPY)
<L>       Address of L
<D>       Address of array D
<K>       Address of K
<C>       Address of array C
FEFD      (INDX, FADD, INDX, FLST)
<J>       Address of J
<B>       Address of array B
<A>       Address of array A
4000      (FEND . . . .)
```

TABLE 5-1. ADDRESSING METHODS EXAMPLES

Types of Addressing	Location	Contents	Description
Absolute	0100_{16}	$B444_{16}$	Command Code (FLDD, FEND . . .)
	0101_{16}	0200_{16}	Pointer address (ABS)
	0200_{16}	$xxxx_{16}$	Operand
	0201_{16}	$xxxx_{16}$	Operand
	0202_{16}	$xxxx_{16}$	Operand (double precision only)
Relative	0100_{16}	$B444_{16}$	Command code (FLDD, FEND . . .)
	0101_{16}	0200_{16}	Pointer address (relative)
	0301_{16}	$xxxx_{16}$	Operand
	0302_{16}	$xxxx_{16}$	Operand
	0303_{16}	$xxxx_{16}$	Operand (double precision only)
Indexed †	0100_{16}	$B444_{16}$	Command code (FLDD, FEND . . .)
	0101_{16}	0200_{16}	Pointer address x 2 (index register)
	0400_{16}	$xxxx_{16}$	Operand
	0401_{16}	$xxxx_{16}$	Operand
Indexed † †	0100_{16}	$B444_{16}$	Command code (FLDD, FEND . . .)
	0101_{16}	0200_{16}	Pointer address x 3 (index register)
	0500_{16}	$xxxx_{16}$	Operand
	0501_{16}	$xxxx_{16}$	Operand
	0502_{16}	$xxxx_{16}$	Operand
Relative Indexed †	0100_{16}	$B444_{16}$	Command code (FLDD, FEND . . .)
	0101_{16}	0200_{16}	Pointer address relative x 2 (index register)
	0501_{16}	$xxxx_{16}$	Operand
	0502_{16}	$xxxx_{16}$	Operand
Relative Indexed † †	0100_{16}	$B444_{16}$	Command code (FLDD, FEND . . .)
	0101_{16}	0200_{16}	Pointer address relative x 3 (index register)
	0601_{16}	$xxxx_{16}$	Operand
	0602_{16}	$xxxx_{16}$	Operand
	0603_{16}	$xxxx_{16}$	Operand

† Where index register equals 100 and single-precision mode

†† Where index register equals 100 and double-precision mode

TABLE 5-2. FUNCTION OPERATING TIME CORRELATION

Function	Time in Microseconds	Latencies	Comments
Fetch Command Code	1.25	1	Total time – no overlap
FLDD	4.63	0	Total time – no overlap
FMPY	10.89	1	Total less the overlapping of FLDD
FADD	6.05	0†	Total less the overlapping of FADD
FLST	2.25	1	Irreducible component
Fetch C. C.	1.25	1	No overlap
FEND	.20	0	No overlap
	26.52	1	Latencies

†Latency overlaps the preceding function

TABLE A-1. FLOATING POINT CONVERSION TABLE (DECIMAL AND HEXADECIMAL NUMBERS TO FLOATING POINT)

Normalization Point

Decimal	Operand Sign	Exponent Sign	Exponent						Coefficient								Packed Hexadecimal Form With Bias of 80_{16}	Hexadecimal
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000 0000 4000 0000	0
1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	40C0 0000	1
2	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	4140 0000	2
3	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	4160 0000	3
4	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	41C0 0000	4
5	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	41D0 0000	5
6	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	41E0 0000	6
7	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	41F0 0000	7
8	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	4240 0000	8
9	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	4248 0000	9
10	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	4250 0000	A
11	0	0	0	0	0	0	1	0	0	1	0	1	1	0	0	0	4258 0000	B
12	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	4260 0000	C
13	0	0	0	0	0	0	1	0	0	1	1	0	1	0	0	0	4268 0000	D
14	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0	4270 0000	E
15	0	0	0	0	0	0	1	0	0	1	1	1	1	0	0	0	4278 0000	F
16	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	42C0 0000	10
17	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	42C4 0000	11
18	0	0	0	0	0	0	1	0	1	1	0	0	1	0	0	0	42C8 0000	12
19	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	42CC 0000	13
20	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	0	42D0 0000	14
21	0	0	0	0	0	0	1	0	1	1	0	1	0	1	0	0	42D4 0000	15
22	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	42D8 0000	16
23	0	0	0	0	0	0	1	0	1	1	0	1	1	1	0	0	42DC 0000	17
24	0	0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	42E0 0000	18
25	0	0	0	0	0	0	1	0	1	1	1	0	0	1	0	0	42E4 0000	19
26	0	0	0	0	0	0	1	0	1	1	1	0	1	0	0	0	42E8 0000	1A
27	0	0	0	0	0	0	1	0	1	1	1	0	1	1	0	0	42EC 0000	1B
28	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0	0	42F0 0000	1C
29	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0	0	42F4 0000	1D

TABLE A-1. FLOATING POINT CONVERSION TABLE (DECIMAL AND HEXADECIMAL NUMBERS TO FLOATING POINT) (Contd)

Decimal	Packed Hexadecimal Form With Bias of 80_{16}		Hexadecimal	Decimal	Packed Hexadecimal Form With Bias of 80_{16}		Hexadecimal
30	4 2 F 8	0 0 0 0	1E	60	4 3 7 8	0 0 0 0	3C
31	4 2 F C	0 0 0 0	1F	61	4 3 7 A	0 0 0 0	3D
32	4 3 4 0	0 0 0 0	20	62	4 3 7 C	0 0 0 0	3E
33	4 3 4 2	0 0 0 0	21	63	4 3 7 E	0 0 0 0	3F
34	4 3 4 4	0 0 0 0	22	64	4 3 C 0	0 0 0 0	40
35	4 3 4 6	0 0 0 0	23	65	4 3 C 1	0 0 0 0	41
36	4 3 4 8	0 0 0 0	24	66	4 3 C 2	0 0 0 0	42
37	4 3 4 A	0 0 0 0	25	67	4 3 C 3	0 0 0 0	43
38	4 3 4 C	0 0 0 0	26	68	4 3 C 4	0 0 0 0	44
39	4 3 4 E	0 0 0 0	27	69	4 3 C 5	0 0 0 0	45
40	4 3 5 0	0 0 0 0	28	70	4 3 C 6	0 0 0 0	46
41	4 3 5 2	0 0 0 0	29	71	4 3 C 7	0 0 0 0	47
42	4 3 5 4	0 0 0 0	2A	72	4 3 C 8	0 0 0 0	48
43	4 3 5 6	0 0 0 0	2B	73	4 3 C 9	0 0 0 0	49
44	4 3 5 8	0 0 0 0	2C	74	4 3 C A	0 0 0 0	4A
45	4 3 5 A	0 0 0 0	2D	75	4 3 C B	0 0 0 0	4B
46	4 3 5 C	0 0 0 0	2E	76	4 3 C C	0 0 0 0	4C
47	4 3 5 E	0 0 0 0	2F	77	4 3 C D	0 0 0 0	4D
48	4 3 6 0	0 0 0 0	30	78	4 3 C E	0 0 0 0	4E
49	4 3 6 2	0 0 0 0	31	79	4 3 C F	0 0 0 0	4F
50	4 3 6 4	0 0 0 0	32	80	4 3 D 0	0 0 0 0	50
51	4 3 6 6	0 0 0 0	33	81	4 3 D 1	0 0 0 0	51
52	4 3 6 8	0 0 0 0	34	82	4 3 D 2	0 0 0 0	52
53	4 3 6 A	0 0 0 0	35	83	4 3 D 3	0 0 0 0	53
54	4 3 6 C	0 0 0 0	36	84	4 3 D 4	0 0 0 0	54
55	4 3 6 E	0 0 0 0	37	85	4 3 D 5	0 0 0 0	55
56	4 3 7 0	0 0 0 0	38	86	4 3 D 6	0 0 0 0	56
57	4 3 7 2	0 0 0 0	39	87	4 3 D 7	0 0 0 0	57
58	4 3 7 4	0 0 0 0	3A	88	4 3 D 8	0 0 0 0	58
59	4 3 7 6	0 0 0 0	3B	89	4 3 D 9	0 0 0 0	59

TABLE A-1. FLOATING POINT CONVERSION TABLE (DECIMAL AND HEXADECIMAL NUMBERS TO FLOATING POINT) (Contd)

Decimal	Packed Hexadecimal Form With Bias of 80_{16}								Hexadecimal	Decimal	Packed Hexadecimal Form With Bias of 80_{16}								Hexadecimal
90	4	3	D	A	0	0	0	0	5A	97	4	3	E	1	0	0	0	0	61
91	4	3	D	B	0	0	0	0	5B	98	4	3	E	2	0	0	0	0	62
92	4	3	D	C	0	0	0	0	5C	99	4	3	E	3	0	0	0	0	63
93	4	3	D	D	0	0	0	0	5D	100	4	3	E	4	0	0	0	0	64
94	4	3	D	E	0	0	0	0	5E	101	4	3	E	5	0	0	0	0	65
95	4	3	D	F	0	0	0	0	5F	102	4	3	E	6	0	0	0	0	66
96	4	3	E	0	0	0	0	0	60	103	4	3	E	7	0	0	0	0	67

INTEGERS

POSITIVE INTEGER

Decimal to Binary Conversion

$39_{10} = 0\ 1\ 0\ 0\ 1\ 1\ 1._2$

$1 \times 2^0 = 1_{10}$

$1 \times 2^1 = 2_{10}$

$1 \times 2^2 = 4_{10}$

$0 \times 2^3 = 0_{10}$

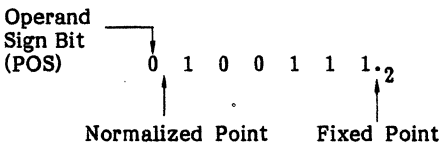
$0 \times 2^3 = 0_{10}$

$1 \times 2^5 = 32_{10}$

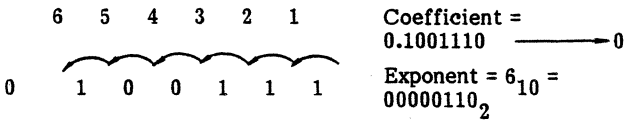
39_{10}

Binary Fixed Point to Binary Floating Point Conversion

Point Conversion



Shift the binary fixed point to the normalized point to obtain the exponent. The numbers of shifts equals the exponent.



Bias Exponent (Exclusive OR with 80_{16})

True exponent	=	0000	0100
Bias	=	1000	0000
Bias exponent	=	1000	0110

Pack Integer Coefficient and Exponent into Floating Point Word Format

See figure B-1 for packing the integer coefficient and exponent into the floating point word format.

NEGATIVE INTEGER

Express the negative number as a positive number and convert as you would for a positive integer. (Follow the positive integer procedure.) Complement the entire 32 bits and the conversion is complete; e.g., $-39_{10} = BCB1\ FFFF_{16}$.

FRACTIONS

POSITIVE FRACTION

Decimal to Binary Conversion

$3/16_{10} = 0\ .\ 0\ 0\ 1\ 1\ 0\ 0$

$1 \times 2^{-4} = 1/16_{10}$

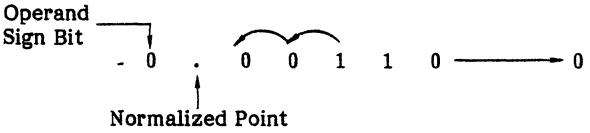
$1 \times 2^{-3} = 1/8_{10}$

$0 \times 2^{-2} = 0_{10}$

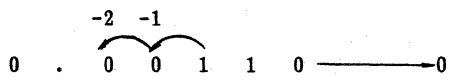
$0 \times 2^{-1} = 0_{10}$

$3/16_{10}$

Binary Fixed Point to Binary Floating Point Conversion

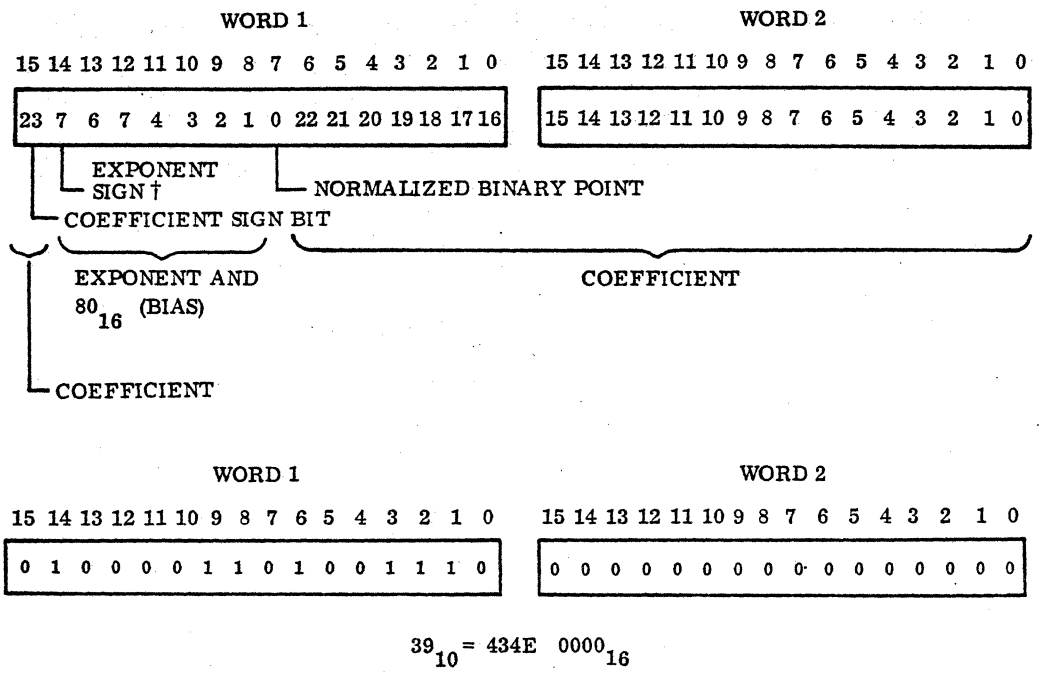


Normalize the coefficient; i.e., shift left until a 1 appears in bit position 22 of the coefficient.



The number of shifts equal the exponent negative.

Coefficient = 0.11000 → 0
Exponent = -2_{10} = 1111 1101_2



$39_{10} = 434E\ \ 0000_{16}$

†TO OBTAIN THE TRUE EXPONENT SIGN OF A PACKED FLOATING POINT NUMBER, DEBIAS THE EXPONENT (I.E., EXCLUSIVE OR WITH 80_{16}) AND, IF THE COEFFICIENT IS NEGATIVE, COMPLEMENT THE ENTIRE EXPONENT. THE TRUE EXPONENT AND SIGN WILL RESULT.

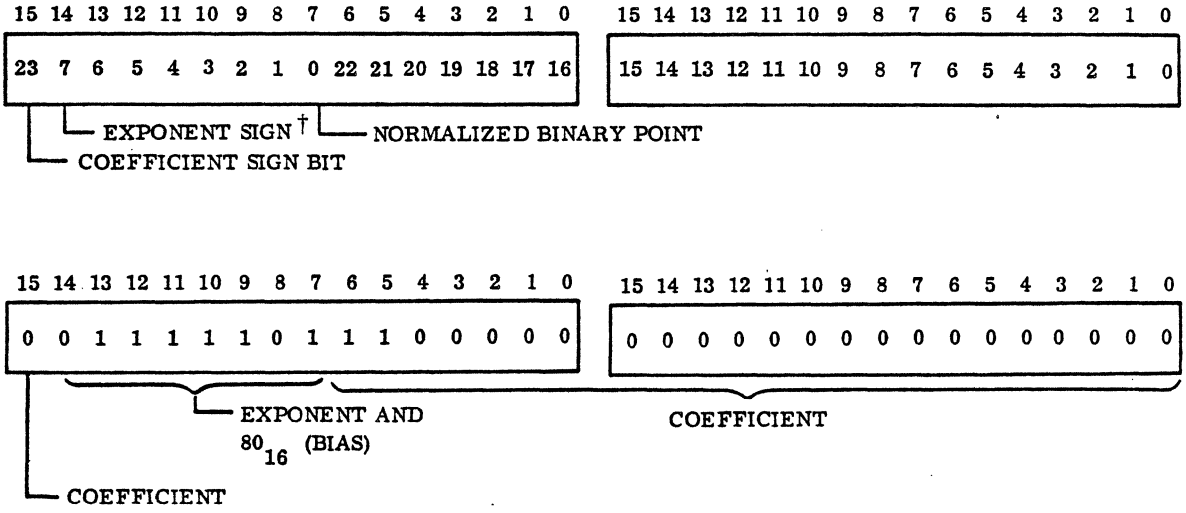
Figure B-1. Packing Integer Coefficient and Exponent into Floating Point Word Format

Bias Exponent (Exclusive OR with 80_{16})

True exponent	=	1111	1101_2
Bias	=	1000	0000_2
Biased exponent	=	0111	1101_2

Pack Fraction Coefficient and Exponent into Floating Point Word Format

See figure B-2 for packing the fraction coefficient and exponent into the floating point word format.



$3/16 = 3EE00000_{16}$

† TO OBTAIN THE TRUE EXPONENT SIGN OF A PACKED FLOATING POINT NUMBER, DEBIAS THE EXPONENT (I.E., EXCLUSIVE OR WITH 80_{16}) AND, IF THE COEFFICIENT IS NEGATIVE, COMPLEMENT THE ENTIRE EXPONENT. THE TRUE EXPONENT AND SIGN WILL RESULT.

Figure B-2. Packing Fraction Coefficient and Exponent into Floating Point Word Format

COMMENT SHEET

MANUAL TITLE CDC® Hardware Floating Point Unit Reference Manual

PUBLICATION NO. 88951100 REVISION A

FROM NAME:

BUSINESS
ADDRESS:

COMMENTS: This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number.

CUT ALONG LINE

STAPLE

STAPLE

FOLD

FIRST CLASS
PERMIT NO. 333

LA JOLLA CA.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
PUBLICATIONS AND GRAPHICS DIVISION
4455 EASTGATE MALL
LA JOLLA, CALIFORNIA 92037

CUT ALONG LINE

FOLD

STAPLE

STAPLE