Gould CONCEPT/32

Floating-Point Accelerator

Model 3611

Technical Manual

March 1983

Publication Order Number: 303-003020-000

GOULD
Electronics

This manual is supplied without representation or warranty of any kind. Gould Inc., S.E.L. Computer Systems Division therefore assumes no responsibility and shall have no liability of any kind arising from the supply or use of this publication or any material contained herein.

HISTORY

The Gould CONCEPT/32 Floating-Point Accelerator (Model 3611) Technical Manual, Publication Order Number **303-003020-000**, was printed March, 1983.

This manual contains the following pages:

CONTENTS

TABLES

ILLUSTRATIONS

CHAPTER 1

GENERAL DESCRIPTION

1.1 Introduction

The optional floating-point accelerator (FPA) is used by the central processing unit (CPU) to increase the speed of arithmetic operations.

In this manual, unless otherwise specified, the following conventions will apply:

1. The term CPU refers to the CONCEPT 32/67 Central Processing Unit.

2. The terms add and subtract unit, A/S unit and FPA A board refer to the 160-103556 Add and Subtract Unit.

3. The terms multiply and divide unit, M/D unit and FPA B board refer to the 160-103557 Multiply and Divide Unit.

1.2 Purpose

The FPA performs floating-point addition, subtraction, multiplication, and division on single-precision (32-bit) and double-precision (64-bit) operands. The FPA also performs fixed-point multiplication.

The following floating-point arithmetic operations are supported by the FPA:

1. Add floating-point word
2. Add floating-point doubleword
3. Add floating-point word register to register
4. Add floating-point doubleword register to register
5. Subtract floating-point word
6. Subtract floating-point doubleword
7. Subtract floating-point word register to register
8. Subtract floating-point doubleword register to register
9. Multiply floating-point word
10. Multiply floating-point doubleword
11. Multiply floating-point word register to register
12. Multiply floating-point doubleword register to register
13. Divide floating-point word
14. Divide floating-point doubleword
15. Divide floating-point word register to register
16. Divide floating-point doubleword register to register

The following fixed-point arithmetic operations are supported by the FPA:

1. Multiply by memory byte
2. Multiply by memory halfword
3. Multiply by memory word
4. Multiply register by register
5. Multiply immediate

1.3 Prerequisites

The FPA requires two SelBUS card slots immediately above, and adjacent to the CPU on a CONCEPT/32 chassis. A typical FPA installation is shown in Figure 1-1.

1.4 Equipment Description

The FPA consists of two, 15 inch by 18.5 inch, logic boards. The add and subtract (A/S) unit performs all floating-point addition and subtraction operations. This unit also contains 16 file registers which are shared by both the multiply and divide unit and the add and subtract unit.

The file registers are used to store operands for the arithmetic operations and to store the results of the operations. The contents of the file registers are a copy of the contents of the general purpose registers in the CPU.

The multiply and divide (M/D) unit performs all floating-point multiplication and division operations and fixed-point multiplication operations. The results of multiply/divide operations are put away in the file registers in the A/S unit.

The two logic boards connect directly to the backplane SelBUS at connector P1B. The SelBUS provides the FPA with power, ground and clock connections only; the FPA does not communicate directly over the SelBUS.

The two-connector, 50 pin cable on the pin side of the backplane at P1C forms the file address (FADD) bus between the two floating-point units. This bus is used to address the file register for storing the arithmetic operation results.

The three-connector 50-pin cable at J1 forms the external Y (EY) bus between the two floating-point units and the CPU. This bus is used to carry 32-bit operands from the CPU, and arithmetic results from the M/D unit, to the file registers in the A/S unit.

The two-connector, 50 pin cable at J2 forms the operand (OPR) bus between the two floating-point units. This bus carries 32-bit operands from the file registers in the A/S unit to the M/D unit.

The three-connector, 50 pin cable at J3 forms the external A (EA) bus between the A/S unit and the CPU, and the external B (EB) bus between the two floating-point units and the CPU. The EA bus carries A-port addresses from the CPU to the file registers in the A/S unit. The EB bus carries B-port addresses from the CPU to the file registers in the A/S unit, and the RB address registers in both floating-point units. The cable also carries control and test signals between the CPU and the FPA.

The three-connector, 50 pin cable at J4 forms the external buffered data (EDB) bus between the two floating-point units and the CPU. The bus carries the 32-bit memory operand to both floating-point units.

See Appendix B for detailed signal pin assignments for each connector.

Table 1-1 lists the physical, electrical, environmental, and operational specifications of the floating-point accelerator.

1.5 Functional Description

The following paragraphs describe the operations performed by the FPA and the different data formats used with the FPA. Figure 1-2 is a simplified functional block diagram of the FPA and its interaction with the CPU.

1.5.1 FPA File Registers

FPA file registers 0 through 7 are copies of the CPU (software) general purpose registers and are updated each time the CPU performs a write to the GPRs. All floating-point results are put away in the FPA file registers and the CPU will use the FPA copy of the GPRs for store in

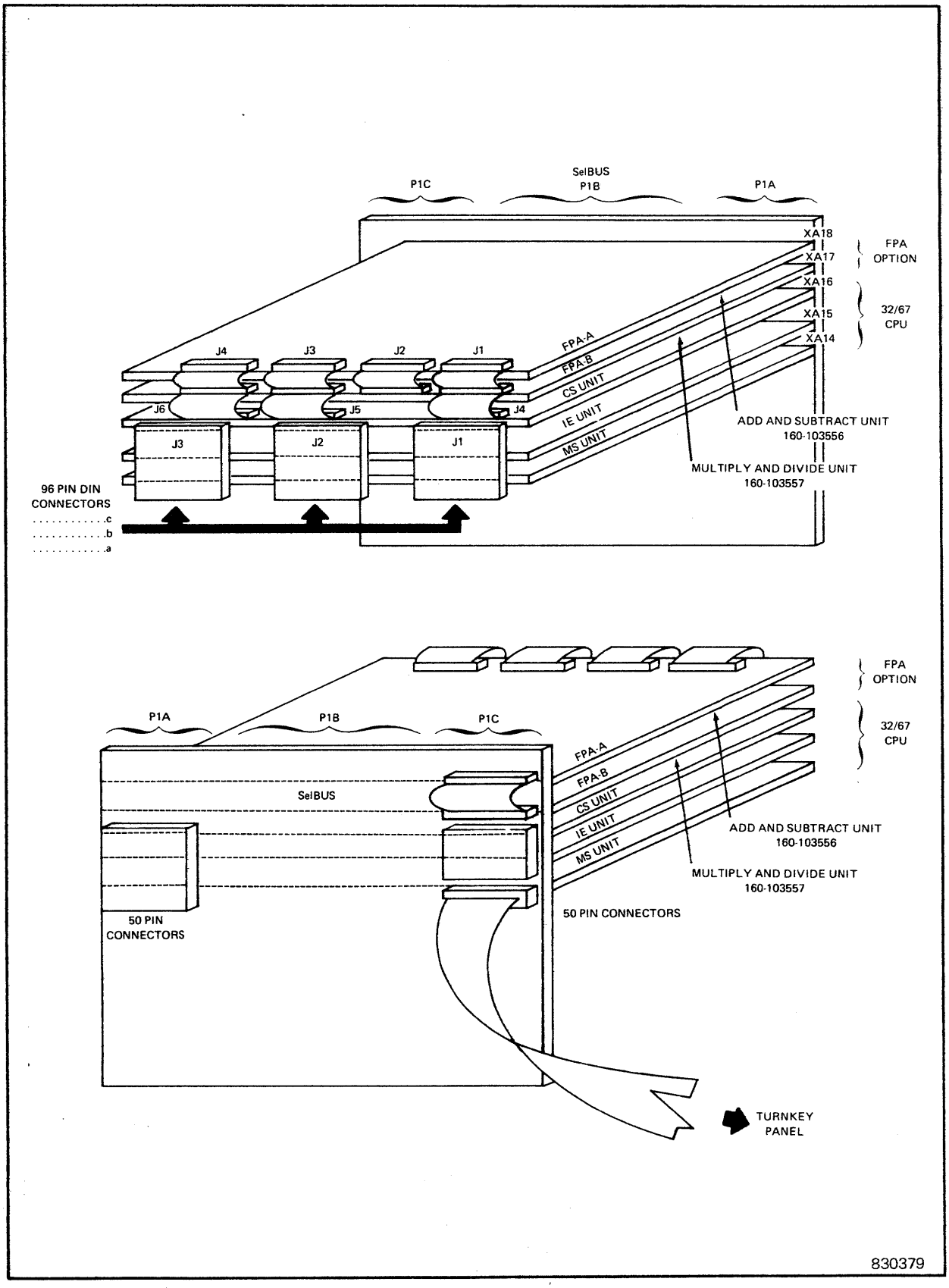

Figure 1-1. Floating-point Accelerator Typical Installation

Table 1-1 (Sheet 1 of 2)
Floating-point Accelerator Specifications

Characteristic	Specification
PHYSICAL	
Number of PWB	Two
Length (each PWB)	18.5 inches (46.99 cm)
Width (each PWB)	15.0 inches (38.10 cm)
Weigh (total for boards and cables)	8 pounds
ELECTRICAL	
Voltage	5 vdc $\pm$ 5%
Current	
Unit A	18 amps
Unit B	24 amps
Power	
Unit A	90 watts
Unit B	120 watts
ENVIRONMENTAL	
Operating	
Temperature	+50 degrees to +100 degrees Fahrenheit (+10 degrees to +40 degrees Celsius)
Relative humidity	5% to 95%
Storage	
Temperature	-40 degrees to +140 degrees Fahrenheit (-40 degrees to +60 degrees Celsius)
Relative humidity	2% to 95%

Table 1-1 (Sheet 2 of 2)
Floating-point Accelerator Specifications

Characteristic	Specification
OPERATIONAL	
Instruction execution time (with 600-nanosecond memory)	
Floating-point instructions	
Add word (ADFW)	0.90 microseconds
Add doubleword (ADFD)	1.50 microseconds
Add word register to register (ADRFW)	1.05 microseconds
Add doubleword register to register (ADRFD)	1.80 microseconds
Subtract word (SUFW)	0.90 microseconds
Subtract doubleword (SUFD)	1.50 microseconds
Subtract word register to register (SURFW)	1.05 microseconds
Subtract doubleword register to register (SURFD)	1.80 microseconds
Multiply word (MPFW)	1.35 microseconds
Multiply doubleword (MPFD)	2.55 microseconds
Multiply word register to register (MPRFW)	1.50 microseconds
Multiply doubleword register to register (MPRFD)	2.85 microseconds
Divide word (DVFW)	4.95 microseconds
Divide doubleword (DVFD)	8.10 microseconds
Divide word register to register (DVRFW)	5.10 microseconds
Divide doubleword register to register (DVRFD)	8.40 microseconds
Fixed-point instructions	
Multiply memory byte (MPMB)	1.95 microseconds
Multiply memory halfword (MPMH)	1.95 microseconds
Multiply memory word (MPMW)	1.80 microseconds
Multiply register by register (MPR)	1.95 microseconds
Multiply immediate (MPI)	1.95 microseconds

Note: Fixed-point multiply and floating-point add, subtract, and multiply operations will yield identical results when performed by either the FPA or the CPU firmware. However, the result of a floating-point divide operation performed by the FPA will be one bit more accurate than the result of the same operation performed by the CPU firmware.

memory instructions. For CPU register modification instructions, the CPU will use the FPA copy of a register, or register pair in double precision, if a floating-point operation has been executed by the same register(s).

The remaining eight FPA file registers (8 through 15) may be used as work registers in writable control store (WCS) applications on a per process basis. The eight upper registers cannot be saved during CPU interrupt/trap context switching; however, they can be accessed by CPU microcode that is not a part of the standard microcode set.

1.5.1.1 Register Conflict Resolution

The CPU and FPA hardware resolves register conflicts between instructions such that, if a floating-point destination register is specified as a source register to a subsequent instruction, and the floating-point instruction is not complete, the CPU will pause and wait for the floating-point instruction to complete. The result of the floating-point operation is then used as the source operand to the subsequent instruction. The subsequent instruction may be a floating-point, fixed point, or logical instruction.

The register conflict hardware cannot resolve conflicts between pre- or post-indexing registers and floating-point target registers. If a floating-point register is specified as an indexing register to a subsequent instruction, and the floating-point instruction is not complete, the results will be indeterminate.

1.5.2 Instruction Formats

An arithmetic instruction supported by the FPA will conform to one of three different instruction formats: memory reference, interregister, and immediate. For detailed descriptions of these instruction formats, see the CPU Reference Manual.

1.5.2.1 Memory Reference Instruction Format

The memory reference instruction format causes the CPU to fetch one operand from memory and the other operand from a general purpose register. The results of the arithmetic operations are returned to the same general purpose register, overwriting the previously stored contents.

The memory reference format is used by the multiply by memory fixed-point instructions and all floating-point instructions other than the register to register type.

1.5.2.2 Interregister Instruction Format

The interregister instruction format causes the CPU to fetch one operand from a source general purpose register and the other operand from a destination general purpose register. The results of the arithmetic operations are returned to the destination general purpose register, overwriting the previously stored contents.

The interregister format is used by the multiply register by register fixed-point instruction and register to register floating-point instructions.

1.5.2.3 Immediate Instruction Format

The immediate instruction format causes the CPU to fetch one operand from a general purpose register. The 16-bit operand portion of the instruction (bits 16 through 31) is sign-extended left, to form the other

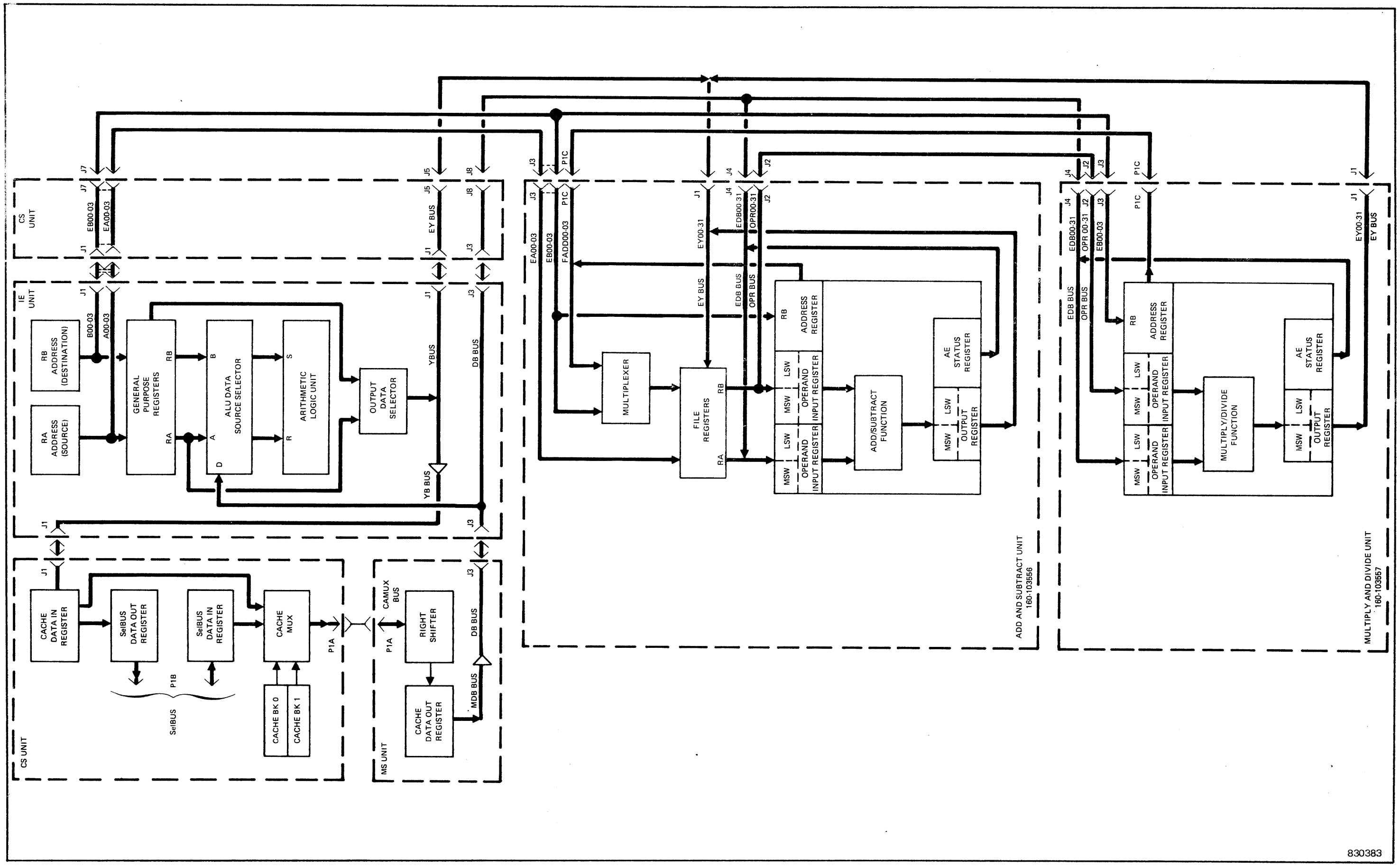

Figure 1-2. FPA/CPU Simplified Block Diagram

32-bit operand, and sent to the FPA. The results of the arithmetic operation are returned to the same general purpose register, overwriting the previously stored contents.

The immediate format is used only by the multiply immediate fixed-point instruction.

1.5.3 Operand Formats

The operands of the arithmetic operation must be stored in memory and/or the general purpose register(s) in the proper format. In addition, floating-point operands must also be normalized. The following paragraphs detail the various operand formats.

1.5.3.1 Fixed-point Operand Formats

Figure 1-3 illustrates the formats for fixed-point operands received by the FPA.

The byte and halfword integers are right justified in a 32-bit word format; the most-significant portion of the byte and halfword operands are zero-filled and sign-extended, respectively. The alignment, zero-fill, and sign-extend functions are performed in the CPU after the operands are fetched from memory.

For every fixed-point multiply instruction executed, the FPA receives two 32-bit operands and returns a 64-bit (doubleword) answer.

1.5.3.2 Floating-point Operand Formats

Figures 1-4 and 1-5 show the proper formats for the single- and double-precision floating-point operands. Both formats consist of a sign (S), an exponent, and a fraction. The sign bit (bit 0) represents the positive (0) or negative (1) value of the fraction. The exponent is a seven-bit (bits 1 through 7) binary number and represents the power to the base 16 of the fraction. The exponent can either be a positive or negative value. The hexadecimal value of 40 represents an exponent of 0 to the base 16. Any hexadecimal value above 40 represents a positive exponent (i.e., hexadecimal of 41 represents the exponent value of positive 1 to the base 16) and any hexadecimal value below 40 represents a negative exponent (i.e., hexadecimal value of 3F represents the exponent value of negative 1 to the base 16).

The fraction is represented by 24 bits (bits 8 through 31) in a single-precision operand and by 56 bits (bits 8 through 63) in a double-precision operand. The radix point of the normalized, binary fraction is assumed to be to the immediate left of its most-significant bit.

When the sign bit represents a negative number, the biased magnitude of the exponent is stored in the one's complement form and the fraction in the two's complement form. Positive numbers are stored in the absolute value.

1.5.3.3 Normalized Floating-point Operands

The floating-point operand is considered to be normalized when the absolute value of the most-significant hexadecimal digit of the fraction contains a value less than one and greater than or equal to one-sixteenth. Hexadecimal fractions with magnitudes less than one-sixteenth must be normalized by left shifting the fraction and decrementing the exponent by one accordingly.

1.5.4 Operand Entry

The floating-point accelerator receives single- and double-precision

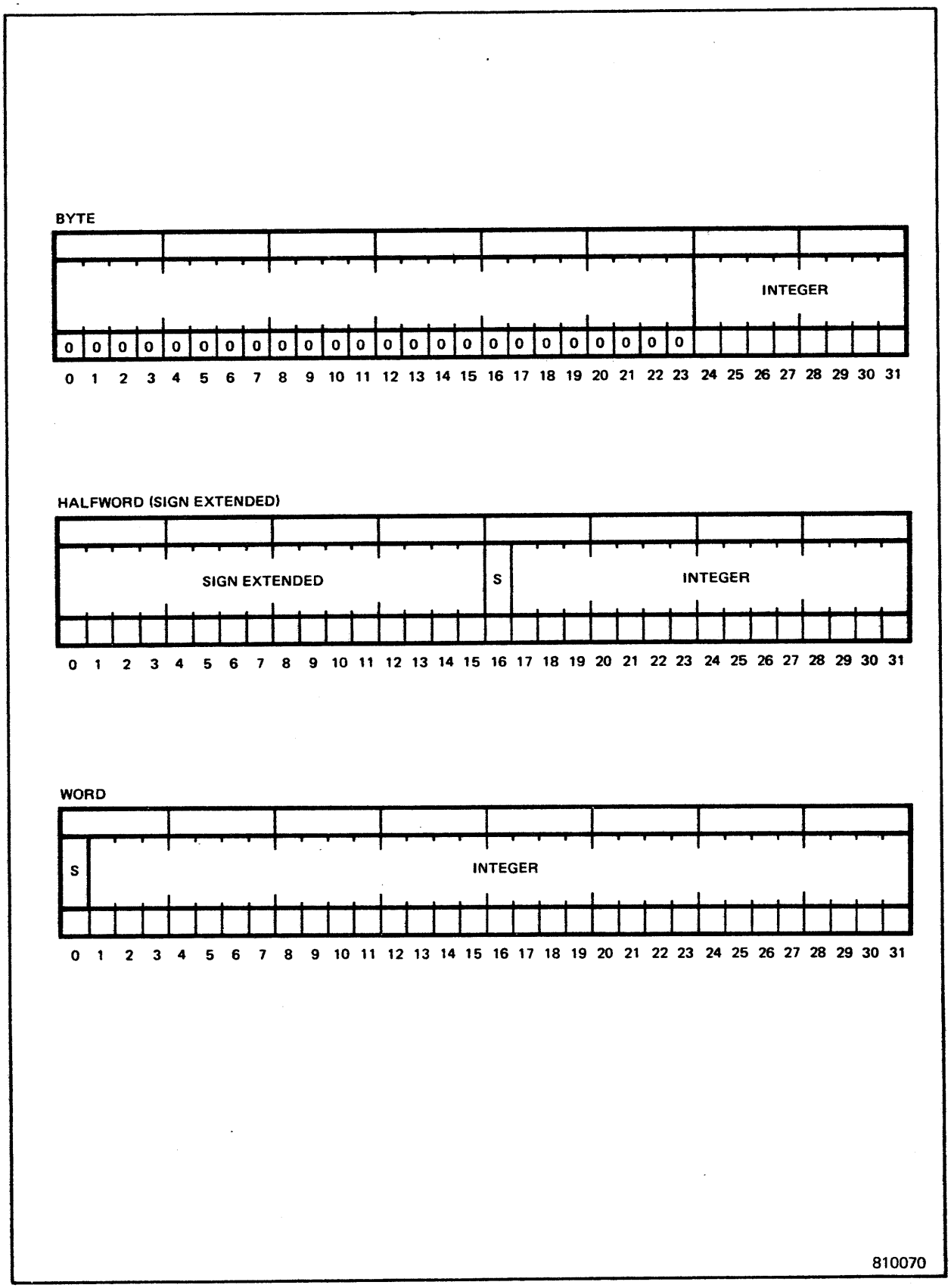

Figure 1-3. Fixed-point Operand Formats

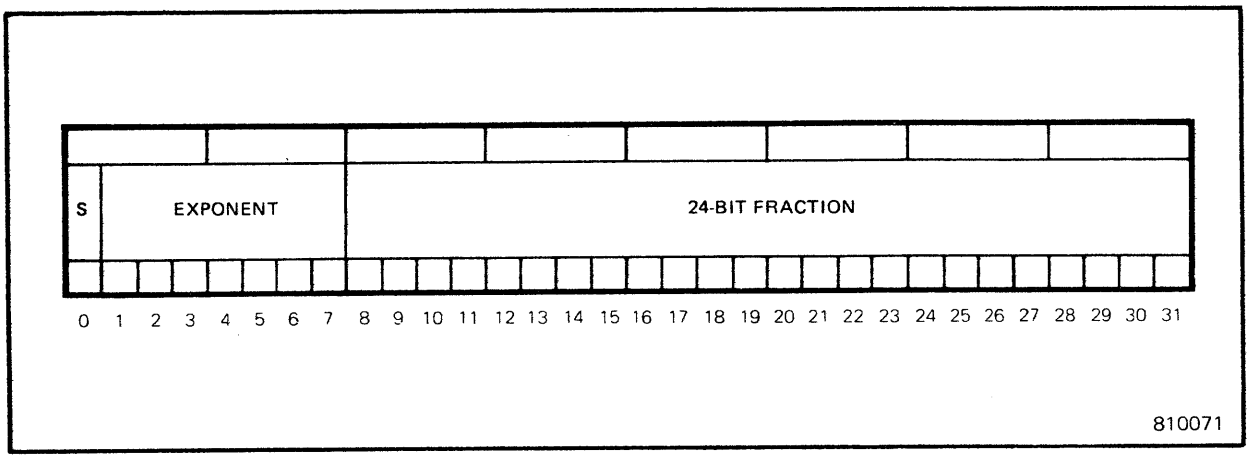

Figure 1-4. Single-precision Floating-point Operand Format

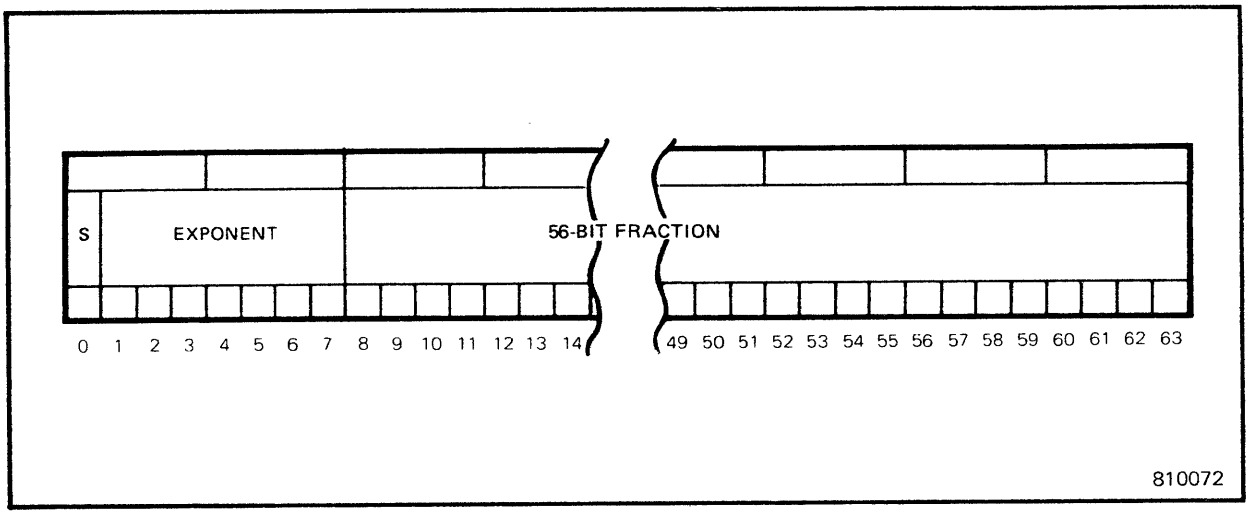

Figure 1-5. Double-precision Floating-point Operand Format

operands from memory and from a general purpose register. The addend of an addition operation, the subtrahend of a subtraction operation, the multiplicand of a multiplication operation, and the divisor of a division operation are stored in a memory location. The augend, minuend, multiplier and dividend are stored in a general purpose register.

1.5.4.1 Memory Operands

Operands fetched from memory enter the data-in register of the CPU and are enabled onto the DB bus (bits DB00 through DB31). These bits are applied to both floating-point units, via the external DB (EDB) bus, through a cable at J4.

1.5.4.2 File Register Operands

The CPU microword controls the loading of an operand from a general purpose register in the CPU, to a file register in the floating-point accelerator. The CPU microword sets up the arithmetic logic unit of the CPU to place the operand on the internal Y bus (bits Y00 through Y31) and enables the external Y bus (bits EY00 through EY31) at J1. The EY bits are applied to the floating-point unit A by the cable at J1. At the same time, the CPU microword activates the register B (RB) address (bits B00 through B03) in which the operand is to be stored. The register B address is the location of the general purpose register which is designated as the destination

register. The register B address (bits EB00 through EB03) is applied to both floating-point units by a cable at J3. The operand is simultaneously stored in the general purpose register and the file register which has been addressed by the RB address.

Double-precision operands are loaded into two consecutive file register and general purpose register addresses.

1.5.5 Input Register Loading

The CPU microword establishes which floating-point unit is enabled based on the arithmetic operation being performed. The A/S unit is enabled for the addition and subtraction operations and the M/D unit is enabled for the multiplication and division operations.

The RB address is stored in the RB address register of the enabled floating-point unit. The stored RB address is used for storing the results of the arithmetic operation.

1.5.5.1 Single Precision

The input registers of the appropriate floating-point unit are loaded during the CPU memory fetch cycle. At the time the operand from memory is enabled onto the DB bus, the operand from the file registers is enabled onto the OPR bus (bits OPR00 through OPR31). The CPU enables the input registers of the unit which is to perform the arithmetic computation, and both operands are loaded simultaneously into the most-significant word (MSW) position of the operand input registers.

The floating-point unit calculations are performed on 64-bit operands. During the loading of single-precision operands, the least-significant word position is automatically zero extended.

1.5.5.2 Double Precision

Double-precision operands are contained in two consecutive file register locations and memory locations. The most-significant word of the doubleword resides in the even address and the least-significant word resides in the odd address. The least-significant words (LSW) of both the memory and register operands are loaded into the least-significant word positions. The CPU then fetches the most-significant operand from memory and simultaneously loads both words into the most-significant word positions of the operand input registers.

1.5.6 Arithmetic Operations

The operands are disassembled into their component parts (the sign, the exponent, and the fraction) by the floating-point accelerator prior to the arithmetic operation. Each of the component parts is manipulated and calculated independently. The results are reassembled prior to being stored in the floating-point accelerator file registers.

1.5.6.1 Addition/Subtraction

The addition/subtraction operations are performed by the add and subtract unit. The addition/subtraction instructions cause the CPU microword to enable the input registers of the A/S unit, and the operands are loaded into the input registers as previously described. After the input registers are loaded, the addition or subtraction operation begins.

The exponents of the fractions are compared to each other to determine the smaller of the two. The fraction of the operand with the smaller exponent is shifted right and the exponent is incremented by one for each hexadecimal digit shifted until the two exponents are equalized. If the difference between the two exponents is greater than the maximum allowable difference (6 for single-precision operands and 13 for double-precision operands), the smaller exponent and its fraction are considered insignificant compared to the larger exponent, and the fraction of the operand with the smaller exponent is forced to zero.

The fractions are algebraically added or subtracted and the resulting fraction is normalized or rounded.

The resulting sign, exponent and fraction are assembled and transferred to the output register of the add and subtract unit.

Since all fractional answers are automatically normalized by the A/S Unit, it is possible to use a floating-point add instruction (ADFW or ADFD) simply to normalize an operand. This can be done by adding the unnormalized operand to another operand with a zero value.

1.5.6.2 Multiplication/Division

The multiplication/division operations are performed by the multiply and divide unit. The multiplication/division instructions cause the CPU microword to enable the input registers of the M/D unit and the operands are loaded into the input registers as previously described. After the input registers are loaded, the multiplication/division operation begins.

1.5.6.2.1 Multiplication

The floating-point accelerator executes both fixed-point and floating-point multiply instructions.

The multiply operation on the floating-point fraction (or fixed-point integer) is performed using a multiple-clock operation where partial products are calculated and then added to the sum of all previous partial products. For single-precision floating-point multiply operations, the summation of partial products may be normalized (if needed) or rounded. For double-precision floating-point multiply operations, only normalization may be performed (if needed), never rounding. In fixed-point operations, no normalization or rounding is performed.

The biased exponents of the floating-point fractions are algebraically added, and then 40 (hex) is subtracted from the sum. The result is the biased exponent of the answer.

Upon completion of the floating-point or fixed-point multiply, the result is transferred to the file register located on the A/S unit.

1.5.6.2.2 Division

Division of the fraction is accomplished by first performing an algorithm using iterative multiplications to obtain the approximate reciprocal of the divisor and then multiplying the reciprocal by the dividend.

The exponent of the answer is derived by first calculating the exponent of the reciprocal and then adding this value to the exponent of the dividend.

The fraction answer is both normalized (if needed) and rounded. The corrected fraction answer, sign bit, and exponent are assembled and

transferred to the file register located on the A/S unit.

1.5.7 Result Normalizing and Rounding

The following paragraphs describe the normalizing and rounding operations.

1.5.7.1 Guard Digit

The guard digit is an additional hexadecimal digit of precision which is used in the normalizing and rounding processes of single-precision arithmetic operations and double-precision division operations. The guard digit of these operations allows the results of the operations to be calculated with a precision of seven hexadecimal digits for single precision and 15 for double precision. The guard digit becomes part of the fraction if normalization is required or is used to determine if the least-significant digit is incremented by one during the rounding process.

1.5.7.2 Normalizing

Left shift normalization of the fractional result of any arithmetic operation is required when its absolute value is less than one-sixteenth. The floating-point unit achieves normalization by shifting the hexadecimal fraction to the left until its magnitude is greater than, or equal to, one-sixteenth. The exponent is decremented by one for each hexadecimal shift. The guard digit, of the single-precision arithmetic operation and the double-precision division operation, is shifted into the least-significant digit position on the first left shift. Subsequent shifts place zeros in the least-significant position of the hexadecimal fraction.

Since the double-precision addition, subtraction and multiplication operations do not employ the guard digit, zeros are always shifted into the least-significant digit position during the normalizing process.

When the fraction contains all zeros, the exponent is forced to a value of zero.

1.5.7.3 Rounding

The rounding process examines the most-significant bit of the guard digit for a one or a zero. If the most-significant bit of the guard digit is a one (represents a hexadecimal value of 8 or greater), a one is added to the hexadecimal fraction. If the most-significant bit of the guard digit is a zero (represents a hexadecimal value of 7 or less), the fraction is not incremented. Thus, both positive and negative fractions are rounded upwards (i.e., more positive).

1.5.8 Result Storage

The results of the arithmetic operation are transferred from the output register of the floating-point units, via the external Y bus (bits EY00 through EY31), to the file registers. The B address register places the B address (bits FADD00 through FADD03) on the external address bus. The results, which are stored in the file register at the location specified by the B address, overwrite the previously stored contents.

The CPU is informed that the results of the arithmetic operation are available for transfer to the destination general purpose register. The CPU microword addresses the destination file register (B address) via the register A address lines (bits A00 through A03). This action places the contents of the destination file register, which was designated by the

B address, on the external DB bus (bits EDB00 through EDB31). The result is stored in the general purpose register designated by the RB address. The result overwrites the previously stored contents of the destination general purpose register.

1.5.9 Arithmetic Exception

An arithmetic exception (AE) will occur if the answer to an arithmetic operation results in an exponent underflow or overflow. The AE will cause a trap, within the CPU, if the AE trap is enabled.

If an AE occurs, and the AE trap is not enabled, the CPU firmware will return a full-scale positive result (@ 7FFFFFFF) with a positive overflow, a full-scale negative result (@ 80000001) with a negative overflow, and (@ 00000000) with an underflow.

At the time the CPU accepts the results of the arithmetic operation from the file register, the CPU tests the floating-point unit for an AE. If an AE has occurred, the CPU firmware enables the arithmetic exception status word onto the external DB bus. The status word informs the CPU of the sign of the resultant which caused the arithmetic exception and whether it resulted in an underflow or an overflow.

An arithmetic exception will prevent the answer from being stored into the file register. Therefore, the file register and the destination general purpose register will retain the original operand.

1.5.10 General Purpose Register Busy

During the arithmetic operation, the destination general purpose register is assigned a busy status.

The CPU requests the transfer of the results from the file register to the destination general purpose register by addressing the file register with the destination address on the RA lines. Addressing the destination register causes the CPU to pause until the results of the operation have been stored into the file register. After the results have been stored, the destination general purpose register is released from its busy status and the CPU is allowed to read the results from the file register and update its general purpose register.

1.5.11 FPA Enable/Disable

Operation of the floating-point accelerator can be enabled or disabled by means of the switch (S1) located on the front edge of the A/S unit circuit board. With the circuit card properly installed, S1 should be switched to the right to enable the FPA.

Operation of the FPA can also be enabled or disabled by the CPU firmware. At power on reset time, an unknown status in the FPA file status register may cause the CPU to stop or perform some error operation. At this time, the CPU firmware will issue a level order (RESET.FPA) to disable the FPA, and a sequence of microcode will clear all the FPA file registers and set the file status register to a known state. At the completion of this reset sequence, the firmware will issue another level order (SET.FPA) to enable the FPA, and the FPA will be ready to resume normal operation.

The SI hardware switch and the firmware set/reset FPA function can override each other. Enabling switch S1 will place the FPA under firmware control.

CHAPTER 2

THEORY OF OPERATION

2.1 Add and Subtract Unit

The add and subtract (A/S) unit contains the logic required to perform floating-point addition and subtraction arithmetic operations. In addition, it contains the floating-point file which provides the interface between the CPU and both floating-point units.

Figure 2-1 is a functional block diagram of the 160-103556 Add and Subtract Unit. The block diagram details the major functional blocks and major signals of the A/S unit. The signal mnemonics shown on the block diagram are similar to those used on the logic diagram. For the add and subtract unit, sheet numbers shown in parentheses in the text, and in the lower right hand corner of each functional block on block diagrams, are the sheet numbers on the A/S unit logic diagram, 130-103556-000, where the circuit details can be found.

2.1.1 Floating-point File

The floating-point file is a major part of the interface between the CPU and the floating-point units. The floating-point arithmetic arguments are passed from the floating-point file registers (FPR) to the operand input registers. The floating-point unit returns the results of the arithmetic operation to the original destination FPR and, subsequently, to the CPU.

The operands from the CPU can be either single-precision or double-precision operands. Both types of operands contain a sign bit (sign of the fraction), a seven-bit exponent value, and either a 24-bit fraction for single-precision operands or a 56-bit fraction for double-precision operands. The radix point of the fraction is assumed to be immediately before the first bit of the fraction.

The exponents of negative numbers are applied to the floating-point file as one's complement numbers and the fractions as two's complement numbers. The exponents and fractions of positive numbers are applied to the file register as absolute values.

2.1.1.1 File Register Logic

The floating-point file register (sheets 28 through 30) stores operands and results of the arithmetic operations input on the external Y (EY) bus. There are sixteen 32-bit file registers, eight of which correspond directly to the CPU general purpose registers (GPR), addresses 0 through 7. The remaining eight, addresses 8 through F, may be used as work registers in writable control store (WCS) applications.

The output of the file register transfers operands to the operand (OPR) bus for loading the operand input register.

The file register can be loaded with an operand from the CPU or with a result of an arithmetic operation by either floating-point unit.

During the execution of a floating-point instruction, the operand in the

general purpose register of the CPU will be loaded into the floating-point file register of the same address (example: GPR0 ⟶ FPR0). This is accomplished by the CPU microword which addresses the (file) register (B-port) via the B address lines (EB00 through EB03). Simultaneously, the external destination field (CREG36 through CREG39) of the microword will specify the floating-point file as the external destination. At the end of the same CPU cycle, the contents of the GPR are loaded into the addressed FPR via the external Y bus lines (EY00 through EY31).

The decoding of the external destination field causes the generation of a load file (LDFILE) signal which clocks the operand, from the EY bus, into the file register designated by the GPR address (EB00 through EB03).

The logic to decode the CPU microword external destination field is contained in the file contention and decode logic.

The results of arithmetic operations are placed on the EY bus from the floating-point unit output register under the control of the microengine. The register B-port address, the same address used for storing the operand, is placed on the file address lines (FADD00 through FADD03) by the unit which performed the arithmetic operation. The microengine issues a unit store (UNITSTORE) signal and the results on the EY bus are stored into the file register designated by the original GPR address.

If, during the arithmetic operation, an arithmetic exception occurs, the arithmetic exception data (AEDATA) signal is generated. This signal inhibits the UNITSTORE, preventing the results from being loaded into the file register, and the file register retains the original operand.

The CPU accesses the contents of the file register by issuing a file enable signal (FILEEN) and applying the file register address, as designated by the destination GPR, to the register A-port (RA) external address lines (EA00 through EA03). The FILEEN signal generates the file output enable (FILEOE) signal which enables the contents of the addressed file register onto the external data bus (EDB00 through EDB31). The CPU GPR is updated with the contents of the file register at the end of the CPU cycle.

2.1.1.2 File Contention Logic

The file contention logic (sheet 27) contains the circuitry for decoding the external destination (ED) field (CREG36 through CREG39) of the CPU microword. The ED field is decoded by the decoder as shown in table 2-1.

The external B lines (EB00 through EB03) are multiplexed with the file address lines (FADD00 through FADD03). When the UNITSTORE signal is not present, the external B lines are selected and generate the file B-port address (FB00 through FB03). When the UNITSTORE signal is present, the file address lines are selected and generate the file B-port address.

2.1.2 Input Registers

The following paragraphs describe the operation of the single- and double-precision input registers.

2.1.2.1 Single Precision

The 24-bit fractions of both operands are loaded simultaneously into the single-precision input register (sheet 4) by the load single (LDSINGLE01) signal from the microengine. One operand fraction is obtained from the file register's B-port via the operand bus (OPR08 through OPR31). The other operand

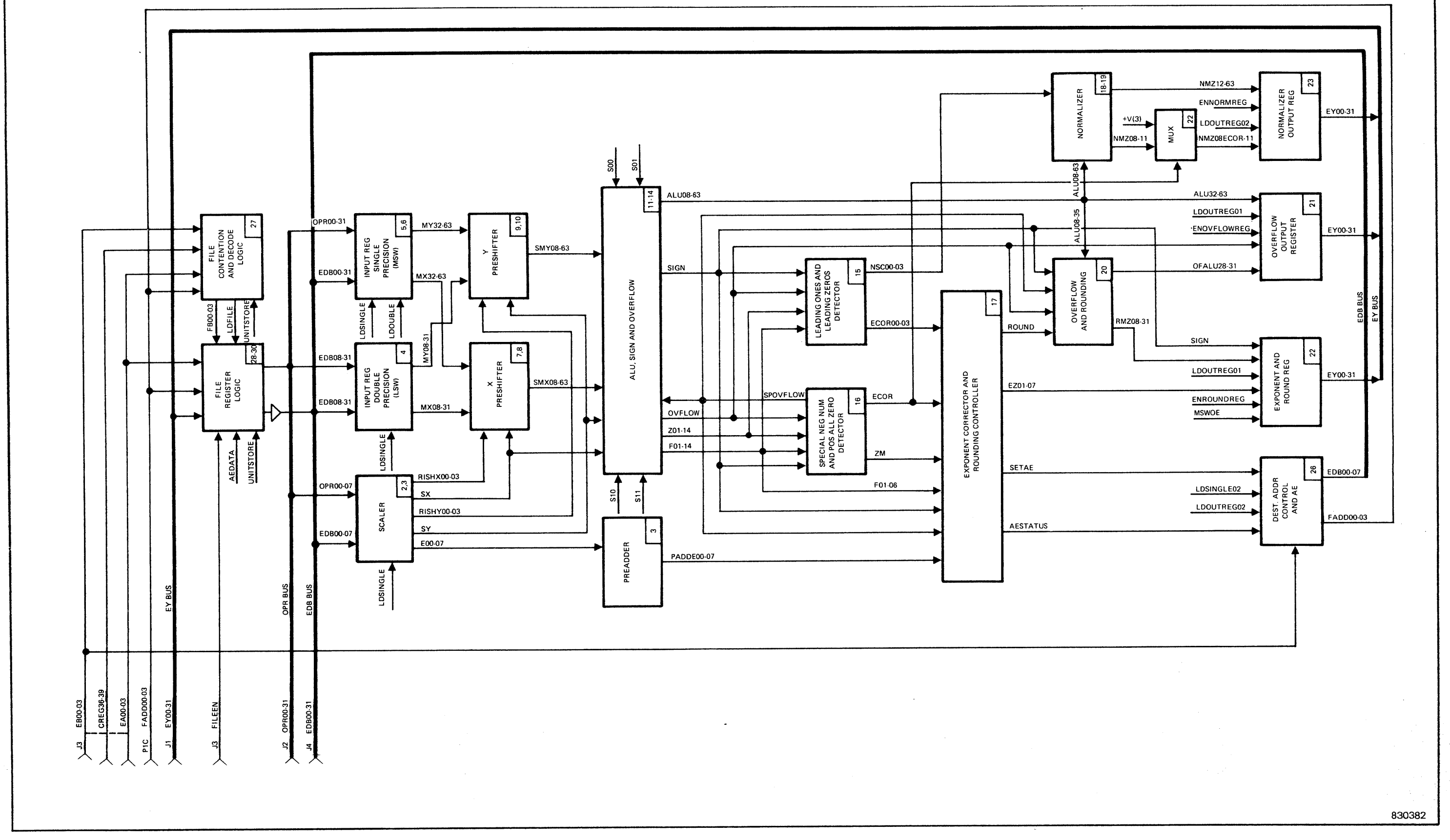

Figure 2-1. Add and Subtract Unit - Block Diagram

Table 2-1
CREG Bits 36-39

Syntax	CREG Bit 36	37	38	39	Function
FP.FILE	1	0	1	1	Floating-point file. This syntax generates the load file (LDFILE) signal. This signal is used to load operands into the file register.
A/S.MSW	1	1	0	0	Add/subtract most-significant word. This syntax generates the single-precision add (SPADD) signal. The signal controls the loading of a word from a selected file register and a word from the EDB bus into the two single-precision (MSW) input registers. This signal is generated for both single- and double-precision operands.
A/S.LSW	1	1	0	1	Add/subtract least-significant word. This syntax generates the double-precision add (DPADD) signal. The signal controls the loading of a word from a selected file register and a word from the EDB bus into the two double-precision (LSW) input registers. This signal is generated for double-precision operands.

is prefetched from memory. The CPU microword enables the memory data onto the external data (EDB) bus (EDB00 through EDB31) after the valid memory data is returned to the CPU, via cache multiplexer and right shifter, and clocked into the cache data out register. At the end of the current CPU cycle, both operands are loaded into the most-significant word (MSW) portion (for single-precision operands) of the FPU input register. Simultaneously, the least-significant word (LSW) portion (double-precision) of the input register is cleared.

The single operand (SINGLEOPR01) signal clears the double-precision input register when a single-precision operand is loaded into the single-precision input register.

The latched outputs of the input register are applied to the X and Y preshifters.

2.1.2.2 Double Precision

The least-significant 32 bits (LSW) of the 56-bit fractions of both operands are loaded simultaneously into the double-precision input register (sheets 5 and 6) by the load double (LDDOUBLE01) signal from the microengine. One operand fraction (LSW) is obtained from the file register B-port via the operand bus (OPR00 through OPR31). The other operand fraction (LSW) is obtained from the cache data out register via the external data bus (EDB00 through EDB31).

After the least-significant word has been loaded and latched into the double-precision input register, the most-significant word is loaded into the single-precision input register as described in paragraph 2.1.2.1; however, the LSW is not cleared.

The latched outputs of the input register are applied to the X and Y preshifters.

2.1.3 Scaler

The scaler (sheets 2 and 3) contains the logic to determine the right-shift count required and which fraction (the X or the Y) requires right shifting in order to make the two exponents equal.

The load single (LDINGLE01) signal from the microengine loads the scaler with the sign bits (bit 0) and the exponents (bit 1 through bit 7) of both operands. The external data bus, bits EDB00 through EDB07, carries the sign and exponent of the X operand. The operand bus, bits OPR00 through OPR07, carries the sign and exponent of the Y operand.

If the sign bit of one or both of the fractions is negative (bit 0 = 1), that fraction's exponent is one's complemented. If the sign bit is positive (bit 0 = 0), the exponent is not altered. The resulting exponent values are simultaneously subtracted from each other using the two's complement method.

The scaler simultaneously subtracts the Y exponent value from the X exponent value and the X exponent value from the Y exponent value and determines which exponent value, the X or the Y, is the greater.

If the X exponent value is the greater, the Y right shift (RISHY00 through RISHY03) count is equal to the difference between the Y exponent value and the X exponent value. The Y right shift count is applied to the Y preshifter.

If the Y exponent value is the greater, the X right shift (RISHX00 through RISHX03) count is equal to the difference between the X exponent value and the Y exponent value. The X right shift count is applied to the X preshifter.

If the difference between the exponent values is greater than six, for single-precision operands, or 13, for double-precision operands, the smaller fraction is considered insignificant and the maximum shift count of 15 is established. This causes the smaller operand to be replaced with all zeros by the preshifter.

The exponent (X or Y) which has the greater absolute value is applied to the preadder as bits E01 through E07. E00 is always a zero.

2.1.4 Preadder

The preadder (sheet 3) predicts the possibility of an arithmetic exception's occurring during the addition or subtraction of two fractions.

In the scaler, the resulting exponent is applied to the exponent lines (E01 through E07). E00 is always a logical zero. A hexadecimal value (see table 2-2) is added to the exponent. The results of the addition indicate the possibility of an arithmetic exception when PADDE00 = 1.

The preadded exponent (PADDE00 through PADDE07) is applied to the exponent corrector logic.

The exponents that will not have an AE in the preadder are shown in Table 2-3.

2.1.5 Preshifters

The preshifters (sheets 7 through 10) shift the X or Y mantissa to compensate for the equalizing of the exponents.

The X mantissa (MX08 through MX31 for single precision and MX32 through

Table 2-2
Preadder Values

Decimal Exponent Sign	Precision	Hexadecimal Value Added
Positive	Single	01
Negative	Single	F8
Positive	Double	01
Negative	Double	F2

Table 2-3
AE Prediction

Decimal Exponent Sign	Precision	Decimal Exponent Value	Hexadecimal Value With 40 Bias
Positive	Single	+ 62	7E
Negative	Single	-56	08
Positive	Single	+62	7E
Negative	Single	-50	0E

Table 2-4
ALU Control Lines

Function	S11	S10	S01	S00
OFF	0	0	0	0
ADD SP	1	1	0	0
ADD DP	1	1	1	1
SUB SP	0	1	0	0
SUB DP	0	1	0	1

MX63 for double precision) is applied to the inputs of the X preshifter (sheets 7 and 8). The Y mantissa (MY08 through MY31 for single precision and MY32 through MY63 for double precision) is applied to the inputs of the Y preshifter (sheets 9 and 10). Either the X or the Y mantissa is right shifted and sign bit extended as determined by the shift count (RISHX00 through RISHX03 or RISHY00 through RISHY03).

2.1.6 ALU Sign and Overflow

Since the arithmetic calculations are performed by the two's complement method, the sign of the mantissa is part of the calculation. Both sign bits (SX and SY) are applied to the sign portion of the arithmetic logic unit (ALU) (sheets 11 through 14), and both mantissas (SMX08 through SMX63 and SMY08 through SMY63) are applied to the mantissa portion of the ALU.

The control signals S10, S11, S00 and S01 from the microengine are applied to the ALU. These signals control the functions of the ALU for both single- and double-precision operation. Table 2-4 shows the states of the control lines for the

ALU functions. The S10 and S11 signals control the ALUs associated with the sign bit and the seven most-significant hexadecimal digits and the S00 and S01 signals control the remaining ALUs associated with the seven least-significant hexadecimal digits.

The ALU is turned off during the loading of the input registers and during the preshift operation. The ALU is turned on after the shifted mantissa signals are stable, approximately 110 nanoseconds into the machine cycle.

The seventh hexadecimal digit (least-significant) is the guard digit. The guard digit provides an extra hexadecimal digit of precision for single-precision operations when either of the X and Y mantissas have been right shifted by the preshifter.

The resulting sign is a function of the addition or subtraction operation of the sign bits and the overflow from the most-significant hexadecimal digit.

An overflow condition can be generated by overflow from the most-significant hexadecimal fraction digit or by a special overflow case which is caused by the addition or subtraction of some special numbers.

Figure 2-2 is a simplified version of the sign and overflow functions. The sign bits (SX and SY) and the most-significant hexadecimal fraction digits (SMX08 through 11 and SMY08 through 11) are applied to the arithmetic logic units. The control lines (S11 and S10) establish the add or subtract function. The carry-out (COUT2) of the most-significant hexadecimal digit ALU is applied to the carry-in (CIN1) of the sign bit ALU (sheet 14) and to the exclusive OR gate B. The carry-out (COUT1) of the sign bit ALU is also applied to the exclusive OR gate B. The sign bit (the result of the addition or subtraction operation) and the overflow (OVFLOW) are applied to the exclusive OR gate C, the output of which is the SIGN+ signal. The OVFLOW signal is also applied to OR gate A, along with the special overflow signal (SPOVFLOW). The outputs of the A gate are the signals OVFLOW01 through OVFLOW 04.

The exclusive OR gate B implements the rules for an overflow condition. Table 2-5 is a state diagram for the overflow condition.

The rule for the overflow condition is: an OVFLOW signal will be generated when COUT1 is present and CIN1 is not present or CIN1 is present and COUT1 is not present; an OVFLOW signal will not be generated when COUT1 and CIN1 are both present or neither are present. The OVFLOW01 through OFVLOW04 signals are generated when either an OVFLOW signal or an SPOVFLOW signal or both are input to OR gate A.

The special overflow is generated under special conditions; for example: when two negative eights are added or a positive eight is subtracted from a negative eight. These two operations will satisfy the rule CIN1 and COUT1 so as to produce no overflow. In this case, these two operations are detected by the special negative number logic and a special overflow signal (SPOVFLOW) is issued. This causes the overflow signals OVFLOW01 through OVFLOW04 to be generated.

The ALU functinal block also contains logic which detects a guard digit value less than, or equal to, seven (hexadecimal) with all other digits equal to zero, for single-precision negative fractions. In this special case, the guard digit is cleared, an F (hexadecimal) is inserted into the most-signficant hexadecimal digit, and the exponent is decremented by one.

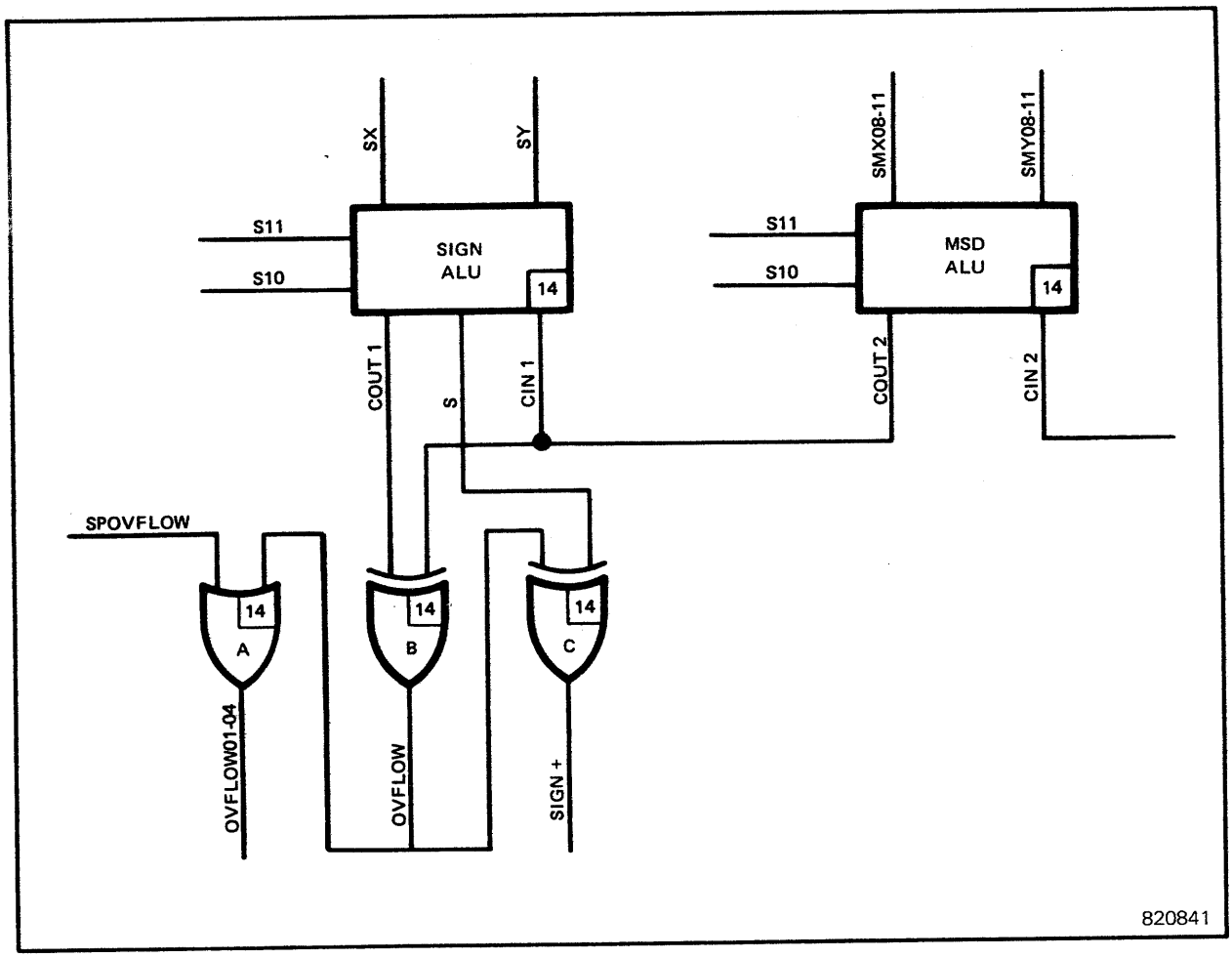

Figure 2-2. Simplified Sign and Overflow Function

Table 2-5
Overflow State Diagram

CIN1	COUT1	SPOVFLOW	OVFLOW01-04
0	0	0	0
0	1	0	1
1	0	0	1
1	1	0	0
1	1	1	1
X	X	1	1

X = Don't Care

2.1.7 Special Negative Number and Positive All Zero Detector

The special negative number and positive all-zero detector logic (sheet 16) detects an all-zeroes result in single- and double-precision hexadecimal fractions of an arithmetic operation. It also detects a single-precision negative number in which the hexadecimal mantissa is all zeros and the guard digit is equal to, or less than, a hexadecimal 7.

Both of these conditions produce the error correct signal (ECOR) to the exponent correction logic. (Some special single-precision, negative number cases also cause the generation of the special overflow signal (SPOVFLOW) which is used to correct the results.)

2.1.8 Leading Ones and Leading Zeros Detector

In the leading ones and leading zeros detector circuitry (sheet 15), leading Fs (hexadecimal) of single- and double-precision negative mantissas (presented in the twos complement form) and leading zeros of single- and double-precision positive mantissas are detected if no overflow condition exists. The number of leading Fs or zeros detected establishes a count on the exponent correct (ECOR00 through ECOR03) lines which is used to decrement the exponent.

The normalized shift count (NSC) is used to left shift the mantissa so that a hexadecimal digit other than a zero for a positive mantissa and an F (hexadecimal) for a negative mantissa is to the immediate right of the radix point.

2.1.9 Exponent Corrector

The exponent corrector (sheet 17) adjusts the exponent to compensate for the bit that was added in the preadder, for normalizing of the mantissa, and for special case numbers.

The error correction bits (ECOR00 through ECOR03) are added to the exponent (PADDE00 through PADDE07) from the preadder. The exponent is adjusted based on the conditions that exist.

When there is no overflow, no request for normalization, and the mantissa is either positive or negative, a value of FX (hexadecimal) is added to the preadded exponent. The value of X (hexadecimal) is the one's complement of the mantissa shift count, which in this case is F. This causes the preadded exponent to be decremented by 1 to compensate for the preadded bit.

When there is no overflow, but there is a request for normalizing, and the mantissa is either positive or negative, a value of FX (hexadecimal) is added to the preadded exponent. The value of X (hexadecimal) is the one's complement of the mantissa shift count.

When there is a normal overflow or special overflow (negative mantissa only), no request for normalizing, and the normal overflow mantissa is either a positive or negative, a value of 00 (hexadecimal) is added to the preadded exponent.

The special negative number case where the mantissa contains one or more leading Fs, followed by all zeros, there is no overflow, and a request for normalizing, a value of FX+1 (hexadecimal) is added to the preadded exponent. The value of X (hexadecimal) is the one's complement of the mantissa shift count. The 1 (hexadecimal) is added because the error correct (ECOR) signal is true for the special negative numbers.

When the sign of the mantissa is negative, the corrected exponent is one's complemented before it is transferred to the exponent output register. The corrected exponents of positive mantissas are in their absolute values and are not complemented before being transferred to the exponent output register.

The exponent corrector logic also contains the logic for determining if an arithmetic exception has occurred. The exponent is seven bits wide; however, the output of the preadder is eight bits wide. This allows the most-significant bit (bit 00) of the exponent corrector to be used to indicate the occurrence of an arithmetic exception (SETAE). The set arithmetic exception signal is added with the sign positive (SIGN+) signals to indicate a positive or negative arithmetic exception. The arithmetic exception signals are transferred to the arithmetic exception status register. The arithmetic exception status is transferred to the CPU by firmware request.

2.1.10 Normalizer

The normalizer (sheets 18 and 19) receives the mantissa from the arithmetic logic units and the normalized shift count from the leading ones and leading zeros detector. The mantissa is shifted left by the amount specified by the shift count. Both single- and double-precision mantissas can be normalized.

2.1.11 Overflow and Rounding

In the overflow and rounding circuitry (sheet 20), single-precision mantissas are rounded to the next higher hexadecimal digit if the guard digit (ALU32 through ALU35) is equal to, or greater than, eight (hexadecimal). This is accomplished by adding a hexadecimal one (the ROUND signal) to the least-significant mantissa digit. There is an exception to the rounding of the single-precision mantissa. A mantissa which contains all Fs (hexadecimal) is not rounded because it would cause an overflow condition.

There are several overflow conditions which require special handling of the most-significant digit (ALU08 through ALU11). In each case, the digit is placed into the most-significant digit and the mantissa is shifted right by the overflow right shifter (2:1 multiplexers).

When a positive overflow condition exists, a one (hexadecimal) is placed into the most-significant digit.

When a negative overflow condition exists, an E (hexadecimal) is placed into the most-significant digit.

When a special overflow condition exists, an F (hexadecimal) is placed into the most-significant digit.

2.1.12 Normalizer Error Corrector

The normalizer error corrector (sheet 22) is a 2:1 multiplexer which normally selects the normalized bits (NMZ08 through NMZ11); however, in some special cases it will select an F (hexadecimal).

These special cases exist when the result of the ALU operation has a negative sign, the mantissa must be normalized one or more positions, and the normalized mantissa contains all zeros.

An example of this special case is as follows:

> Assume the result of the ALU operation is a negative FFFF00. The mantissa requests normalizing because of the four leading Fs. The result of normalizing this mantissa is a negative 000000, which is an invalid negative number. The special negative

number detector detects the special case of negative mantissas with leading Fs followed by all zeros and generates the error correction (ECOR) signal. The ECOR signal causes the multiplexer to select an F instead of the normalized bits (NMZ08 through NMZ11) and applies it, via the normalized error corrected lines (MNZ08ECOR through NMZ11ECOR), to the normalized output register.

2.1.13 Output Registers

The add and subtract unit has two sets of mantissa output registers (sheets 21 through 23). One set contains the sign, the exponent, the results of an ALU overflow, and/or the results of a single-precision rounding request. The other set of output registers contains the sign, the exponent, and the normalized results if an overflow condition does not exist.

Both sets of registers are 64 bits wide (the exponent portion of the exponent and rounded register is common to both sets) at their inputs and both sets send their contents to the 32-bit wide EY bus (EY00 through EY31). The selection of one or the other set of output registers and the sequencing of their outputs to the EY bus are controlled by the microengine.

2.1.13.1 Exponent and Rounded Register

The exponent and rounded register (sheet 22) contains the sign (SIGN+), exponent (EZ01 through EZ07), and the rounded mantissa (RMZ08 through RMZ31) of a single-precision result, or the most-significant word of a double-precision result. The least-significant word of the double-overflow output register.

The exponent portion of this register also contains the sign and exponent for the normalized output register.

2.1.13.2 Overflow Output Register

The overflow output register (sheet 21) works in conjunction with the exponent and rounded register. It contains the least-significant word (ALU32 through ALU63) of a double-precision result.

2.1.13.3 Normalized Output Register

The normalized output register (sheet 23) contains the normalized results of single- and double-precision operations (NMZ08ECOR through NMZ11ECOR, NMZ12 through NMZ31, and NMZ32 through NMZ63). The sign bit and exponent of the normalized results are contained in the exponent portion of the exponent and rounded register.

2.1.14 Destination Address Control

The destination address control (sheet 26) stores the address of the original file register in which the resulting operand is to be stored. This address is placed on the file address lines (FADD00 through FADD03) during the resulting operand storing process.

2.1.15 Arithmetic Exception Register

The arithmetic exception register (sheet 26) contains the status of the arithmetic exception. The arithmetic exception output enable (AEOE) from the CPU is common to both floating-point accelerator units. The status of both the add and subtract unit and the multiply/divide unit are

reported. The CPU is able to examine the individual bytes of the status word. The status word format is shown in Figure 2-3.

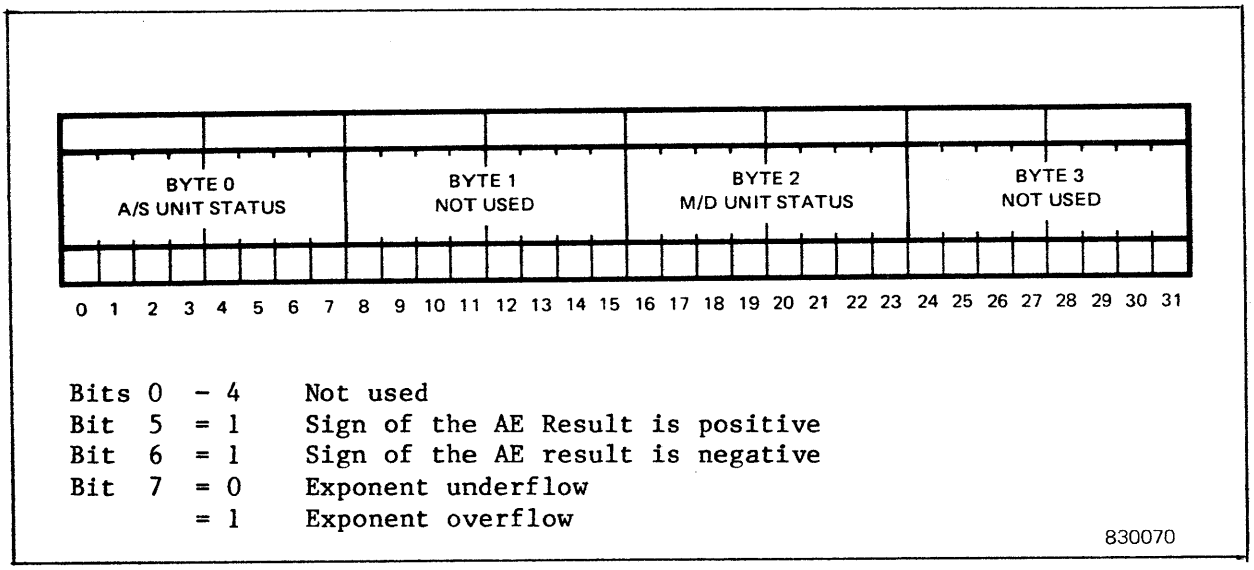

Figure 2-3. Status Word Format

The bits of status byte 0 correspond to bits EDB00 through EDB07 on the external DB bus.

The CPU enables the detection of the arithmetic exception by issuing the arithmetic exception enable (AEXCPEN) signal. When an arithmetic exception occurs, the add and subtract unit will respond with an arithmetic exception signal (AEPEND) to the CPU. The CPU will request the arithmetic exception status word by enabling the status register output.

An arithmetic exception prevents the results from being stored back into the original destination file register.

2.1.16 Clock and Microengine

The clock and microengine circuitry (sheets 24 and 25) contains the logic for the timing, control, and sequencing of the add and subtract unit.

2.2 Multiply and Divide Unit

The multiply and divide (M/D) unit contains the logic required to perform the floating-point multiplication and division and fixed-point multiplication operations.

For reference, a block diagram of the 160-103557 M/D unit is provided in Figure 2-4. In the text that follows, additional block diagrams are provided that illustrate that portion of the M/D unit applicable to the particular arithmetic operation being described.

For the multiply and divide unit, sheet numbers shown in parentheses in the text, and in the lower right hand corner of each functional block on block diagrams, are the sheet numbers on the M/D unit logic diagram, 130-103557-000, where the circuit details can be found.

Execution of the floating-point multiply, floating-point divide, and fixed-point multiply operations are described separately. Where similar functions exist among the instructions, they are explained in the description of the floating-point multiply and then referred to, but

not duplicated, in the text for the other instructions. The reader should understand the description of the floating-point multiply before proceeding to the floating-point divide or fixed-point multiply.

2.2.1 Floating-Point Multiply

The main steps implemented for execution of the multiply floating-point word and doubleword instructions by the M/D unit are as follows:

Operands are input from the CPU and the FPA register file.

Instruction execution is initiated by the CPU.

The incoming floating-point multiplicand and multiplier operands are disassembled and loaded into the fraction and exponent logic of the FPA B Board.

For single-precision operations, the fraction is a 24-bit hexadecimal number positioned in bits 08 through 31 of the incoming word operand. For double-precision operations, the fraction is a 56-bit hexadecimal number positioned in bits 08 through 63 of the incoming doubleword operand.

If either, one or both, of the fractions has a negative sign (bit 0), the exponent (bits 01 through 07) of the negative fraction is one's complemented. The biased exponents are algebraically added, and then 40 (hex) is subtracted from the sum. The result is the biased exponent of the answer.

The two fractions are multiplied together and then normalized or rounded as needed to produce the final fraction answer with an appropriate sign. The sign of the final fraction will be negative if either of the original operands was negative.

If the fraction answer is negative, then the biased exponent of the answer will be one's complemented.

The final fraction, fraction sign, and exponent are assembled into the correct format for a single- or double-precision floating-point operand and stored in the FPA register file.

2.2.1.1 Simplified Sequence of Events

1. Load operands
   a. Input the fraction sign bits and the exponents of both operands from the OPR and EDB buses into the exponent input logic (sheet 27).

   b. If either one (or both) of the fraction sign bits is negative, one's complement the exponent associated with the negative fraction (sheet 27).

   c. Load the exponents into the exponent input registers (sheet 27).

   d. Load the fractions of the multiplier (X) and the multiplicand (Y) from the OPR and EDB buses respectively into the fraction logic (sheets 2 and 10).

      Note

      One machine cycle is required to load single-precision operands; two machine cycles are required to load double-precision operands.

   e. Input the destination file address from the DR bus into the RR address logic sheets 26 and 29).

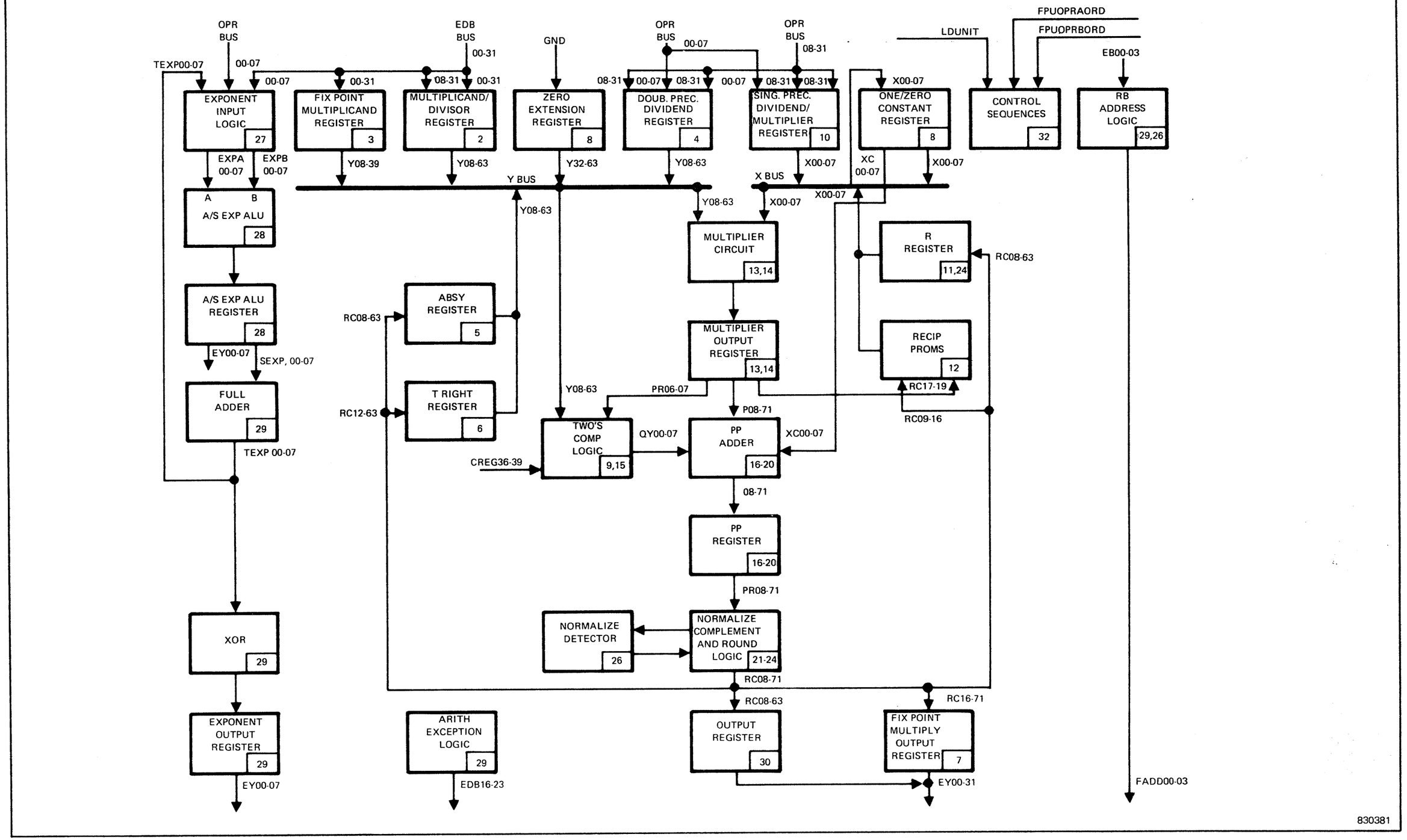

Figure 2-4. Multiply and Divide Unit-Block Diagram

2. In the A/S exponent ALU (sheet 28), add the biased exponents and subtract 40 (hex) from the sum. The result is the biased exponent of the answer.

3. a. Multiply the two fractions together.

   b. Load the biased exponent sum obtained in step two into the exponent ALU register (sheet 28).

4. Correct the fraction product from step 3:

   a. Normalize the fraction answer if needed.

      Exception: Do not normalize negative answer if fraction portion = F0000000

   b. Enable rounding of fraction answer only when normalization is not needed.

   c. If the fraction product required normalization, decrement the exponent.

5. a. If either one (but not both) of the original operands had a negative fraction, one's complement the biased exponent produced in step 2 and/or step 4-b (sheet 29).

   b. If either one (but not both) of the original operands had a negative fraction, set the sign of the fraction answer to a logical one (sheet 29).

   c. Load the sign bit of the fraction answer and the biased exponent into the exponent output register (sheet 29).

      Exception: If the fractional answer from step 3 is all zeros, of if the exponent of the original operands had a negative fraction, one's complement the biased exponent produced in step 2 and/or step 4-b (sheet 29).

   d. Load the corrected fraction answer from step 4 into the floating-point output register (sheet 30).

6. Gate the contents of the exponent output and floating-point output registers into their respective bit positions on the external Y (EY) bus. Register contents are shown in Table 2-6.

7. Output the destination file address from the RB address logic onto the file address bus (FADD00 through 03). This is the address of the file register(s) to which the answer is destined.

2.2.1.2 Sequence Control

Internal operation of the M/D unit is largely controlled by a sequencer (sheet 32). The sequencer includes:

1. Two 512-location by 8-bit control programmable read-only-memories (PROMs)
2. Two synchronous binary counters
3. Five Decode/Demultiplexers (DCD/DMUX)

The PROMs contain encoded control bits which are used to step the M/D unit through the sequence required to execute a given arithmetic

Table 2-6
Output Register Contents

Single-precision answers:		
Exp. Out. Reg. Flt. Pt. Out.Reg.	EY00-07 EY08-31	Cycle n
Double-precision answers:		
Exp. Out. Reg. Flt. Pt. Out. Reg. 08-31	EY00-07 EY08-31	Cycle n
Flt. Pt. Out. Reg. 32-63	EY00-31	Cycle n+1

operation. A memory map of both PROMs is provided in Table 2-7.

Some of the PROM bits drive discrete control lines; the others are arranged in four-bit codes which must be decoded by the appropriate decode/demultiplexer circuits.

The PROMs are addressed sequentially. The high order portion of the PROM address is determined by decoding the A/S and M/D unit orders (LFPUOPRAORD and LFPUOPRBORD) and CREG bits 36 through 39, all of which originate in the CPU. The two floating-point unit orders (sheet 32) specify the type of arithmetic operation (see Table 2-8). CREG bits 36 through 39 are decoded on sheet 15 and indicate whether the operation is single precision (HLDSPMULT) or double precision (HLDDPMULT). The decoding of these inputs from the CPU determines what arithmetic operation will be performed and provides the appropriate address range for the sequencer PROMs (MB84 and MB81). Within that address range is stored the control code for the particular arithmetic operation decoded.

The two binary counters supply the low order portion of the PROM address and sequentially step the address count through the appropriate range. Both counters are reset at the end of each sequence. Therefore, the low-order portion of the PROM address will be at zero when the CPU orders for the next arithmetic operation are received. The reset is a function of HDONE which is coded in the last PROM location of each sequence.

Both PROMs are accessed simultaneously; their parallel outputs provide 16-bits of usable control code which are divided into four fields. Tables 2-9 and 2-10 define the control codes and active sequence control signals applicable to the multiply floating-point word instruction. Tables 2-11 and 2-12 define the control codes and active sequence control signals applicable to the multiply floating-point doubleword instruction. Signal definitions of all the control codes in the sequencer PROMs are provided in Appendix A.

2.2.1.3 Operand Loading

Two operands are loaded simultaneously into the M/D unit logic at the beginning of each instruction execution sequence. One operand originates in memory; it is transferred to the M/D unit board on the 32-bit EDB bus from the CPU's cache data out register. The other operand is fetched from one of the A/S unit's file registers; it is transferred to the FPA B board on the 32-bit OPR bus.

Table 2-7
Sequence Control PROM Map - MB84 and MB81

Address	Contents
0000-0004	Single Precision Flt. Pt. Multiply Sequence Control Code
005 - 03F	All code = 20 (hex).....MB84 only All code = zeros.....MB81 only
040 - 05C	Single Precision Flt. Pt. Divide Sequence Control Code
05D - 07F	All code = 20 (hex).....MB84 only All code = zeros.....MB81 only
080 - 085	Fixed Pt. Multiply Sequence Control Code
086 - 0FF	All code = 20 (hex).....MB84 only All code = zeros.....MB81 only
100 - 108	Double Precision Flt. Pt. Multiply Sequence Control Code
109 - 13F	All code = 20 (hex).....MB84 only All code = zeros.....MB81 only
140 - 16D	Double Precision Flt. Pt. Divide Sequence Control Code
16E - 1FF	All code = 20 (hex).....MB84 only All code = zeros.....MB81 only

Single-precision operands require only one bus transfer, and are always loaded into the most-significant word position of the operand input registers. Double-precision operands require two bus transfers; the least-signficant word (bits 32 through 63) is always transferred first.

The operand input registers are loaded as a result of decoding (sheet 15) CREG bits 36 through 39. These bits originate in the CPU and represent the contents of the external destination field of the CPU microword; they specify whether the M/D unit should load a single- or double-precision operand. For single-precision operations, one CPU microword with "M/D.MSW" in the external destination field will initiate the operand load. For double-precision operations, "M/D.LSW" will be specified first, followed by another microword with "M/D.MSW".

Disassembly of the operands into their component parts is accomplished by steering the fraction, fraction sign bit, and exponent into the appropriate input registers of the exponent and fraction logic.

Table 2-8
Operation Decode

CREG 36-39 HEX VALUE	HLDDP MULT SIGNAL	FPA.A	FPA.B	Operation
E	0	0	0	Single-precision Flt.-Pt. Multiply
E	0	0	1	Single-precision Flt.-Pt. Divide
E	0	1	0	Fixed-Pt. Multiply
E	0	1	1	No Operation
F	1	0	0	Double-precision Flt.-Pt. Multiply
F	1	0	1	Double-precision Flt.-Pt. Divide
F	1	1	0	No Operation
F	1	1	1	No Operation

Table 2-9
Sequence Control PROM Coding - Multiply Floating-point Word

Address	PROM MB84		PROM MB81	
	Field A	Field B	Field C	Field D
	8 7 6 5	4 3 2 1	8 7 6 5	4 3 2 1
000	0 1 1 0	0 0 0 0	0 0 0 0	0 0 0 1
001	0 0 0 1	1 0 0 0	0 0 0 0	0 0 0 0
002	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
003	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 0
004	1 0 1 0	1 1 0 1	1 1 0 0	1 1 1 1

Table 2-10
Active Sequence Control Signals - Multiply Floating-point Word

Step	PROM MB84		PROM MB81	
	Field A	Field B	Field C	Field D
1	HSTOPCLKEN		LSPZEN	LX1STRT
2	LTADDEN HMULTCLR	LTPPA1STRT	LSPZEN	
3	LTADDEN			
4	LTADDEN			LLDEXP
5	HDONE	LSETAE	LSELMULT	LLDOUT

Table 2-11
Sequence Control PROM Coding - Multiply Floating-point Doubleword

Address	PROM MB84		PROM MB81	
	Field A	Field B	Field C	Field D
	8 7 6 5	4 3 2 1	8 7 6 5	4 3 2 1
100	0 1 1 0	0 0 0 0	0 0 0 1	1 0 1 0
101	0 0 0 1	1 0 1 1	0 0 0 1	0 0 0 0
102	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0
103	0 0 0 0	1 0 0 1	0 0 0 1	0 0 1 0
104	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1
105	0 0 0 0	1 0 0 0	0 0 0 1	0 0 0 0
106	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0
107	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 0
108	1 0 1 0	1 1 0 1	1 1 0 0	1 1 1 1

Table 2-12
Active Sequence Control Signals - Multiply Floating-point Doubleword

Step	PROM MB84		PROM MB81	
	Field A	Field B	Field C	Field D
1	HSTOPCLKEN		LDPYEN	LX3STRT
2	LTADDEN HMULTCLR	LTPPA3STRT	LDPYEN	NOP
3	LTADDEN		LDPYEN	NOP
4	LTADDEN	LTPPA2STRT	LDPYEN	LX2STRT
5	LTADDEN	LDPYEN	LX1STRT	
6	LTADDEN	LTPPA1STRT	LDPYEN	NOP
7	LTADDEN		LDPYEN	NOP
8	LTADDEN			LLDEXP
9	HDONE	LSETAE	LSELMULT	LLDOUT

Bits 08 through 31 of the memory operand (multiplicand) are input from the EDB bus into bit positions 08 through 31 of the multiplicand/divisor (M/D) register. Bits 00 through 07 (fraction sign bit and exponent) are input into the exponent input register.

The register operand (multiplier) is similarly loaded into the dividend/multiplier (D/M) register and the exponent input register from the OPR bus.

For single-precision operations, the least-significant portions of the M/D and D/M registers (bits 32 through 63) are not loaded. They retain old data. This portion of these two registers is never gated out to the fraction logic during single-precision multiply.

However, the multiply logic always requires a 56-bit multiplicand. Thus, zeros (in bit positions 32 through 63) are appended to the 24-bit (08 through 31) single-precision fraction output from the M/D register. The zeros are output by the zero extension register and are asserted onto the Y bus in bit positions 32 through 63. The Y bus connects the multiplicand input register (M/D register) and the zero extension register to the fraction multiply logic.

The D/M register output (multiplier) is gated from the D/M register to the multiply logic on the X bus. It is not necessary to append zeros to the multiplier, since only an eight-bit subset of the multiplier is used each multiply iteration. The D/M register's output is sequentially stepped, beginning with bits 24 through 31 the first iteration and followed by bits 15 through 23 and 08 through 15, the second and third multiply iterations.

2.2.1.4 Destination File Address

The M/D unit receives a destination file address from the A/S unit at the same time that the initial operands are received. The destination file address specifies the file register into which the answer should be stored when the arithmetic operation is completed.

The M/D unit simply retains the destination file address and then returns it to the A/S unit, along with the answer, upon completion of the arithmetic operation.

The destination file address is input from the EB bus (bits 00 through 03) and loaded into the file address input register (sheet 29) simultaneous with loading the initial single-precision operand (or, the MSW of the initial double-precision operand). The contents of the file address input register are output to the RB address logic (sheet 26) where they are retained. Upon completion of the arithmetic operation, the destination file address will be sent, via the file address bus (FADD00 through 03), to the A/S unit. This address transfer is controlled by the output control logic (sheet 31).

2.2.1.5 Exponent Handling

The basic sequence for handling exponents is outlined as follows:

1. One's complement either exponent if its associated fraction is negative.
2. Add exponents without removing their 40 (hex) bias.
3. Subtract 40 (hex) from the exponent sum.
4. Load the result from step 3 into the exponent ALU register.
5. Decrement the exponent sum if the fraction answer needed to be normalized.
6. One's complement the final exponent if the sign of fraction answer is negative.
7. Load the exponent output register.
8. Gate the exponent onto the EY bus (EY01 through 07).

There are two exponent input registers. The exponent of the multiplier (exponent A) is always loaded into the exponent A input register (sheet 27); the exponent of the multiplicand is always loaded into the exponent B input register (sheet 27).

When the exponents are input from their respective data buses, each exponent passes through a separate exclusive OR network located in the data path between the data bus input and the A and B exponent input registers (see Figure 2-5). Each exclusive OR network is fed by one of the exponents (A or B) and the sign bit of the fraction associated with that particular exponent. If the sign bit is a logical one (indicating a negative fraction), the exclusive OR network will one's complement the exponent.

The A and B exponents are added together in the exponent ALU (sheet 28) without first removing the bias of either exponent. The addition of the two biased exponents produces a sum with a bias of 80 (hex). To correct this, 40 (hex) is subtracted from the sum of the biased exponents using the full adder which is located at the exponent ALU output. The subtraction is performed by adding the two's complement of 40 (hex) to the output of the exponent ALU.

The result of the exponent add/subtract operation is loaded into the exponent ALU register (sheet 28) when the sequence control logic issues the load exponent signal, LLDEXPL. If the fraction product is zero, the exponent ALU register will be reset, forcing the value of the exponent to zero (LFRCZEXP).

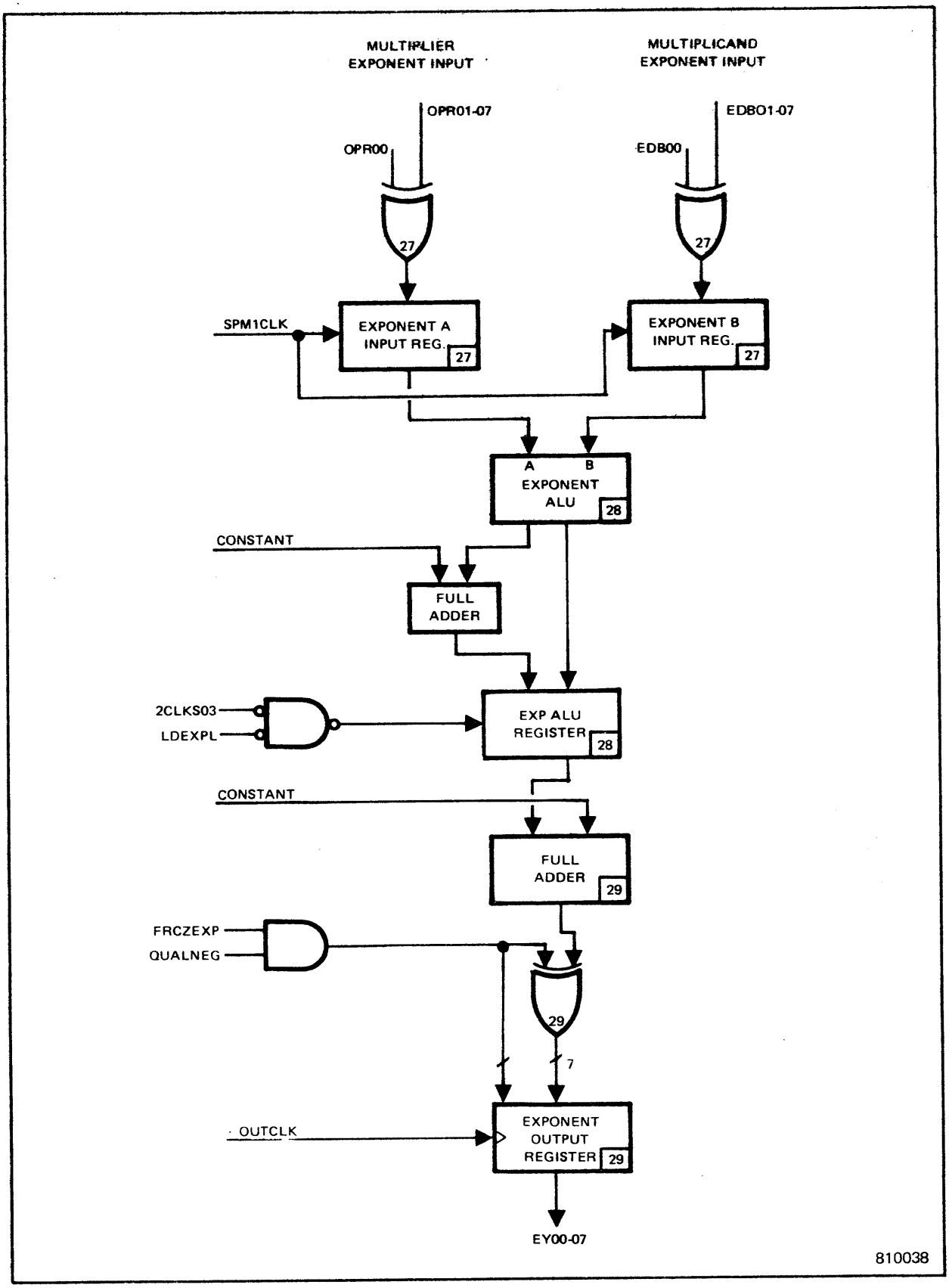

Figure 2-5. Exponent Logic Block Diagram (Floating-point Multiply)

The output of the exponent ALU register propagates to the exponent full adder (sheet 29). The exponent full adder decrements the exponent by one if the fraction normalize detector (sheet 26) has determined that the product of the fraction multiply needs to be normalized. The exponent will never have to be decremented by a value greater than one because the fraction product will never need to be shifted by more than one hex digit to achieve normalization.

If the sign of the fraction product is negative, the exponent will be one's complemented prior to being loaded into the exponent output register (sheet 29). One's complementing the exponent will be inhibited if the exponent was forced to zero in the exponent ALU register (see above).

The exponent output register is loaded with the sign of the fraction and the seven-bit biased exponent when the sequence control logic issues the load output registers signal (LLDOUT). Loading the exponent output register occurs simultaneous with the loading of the fraction into the floating-point output register.

2.2.1.6 Sign Bit Manipulation

The sign bits of both operands enter the FPA B board along with two exponents and fractions from the EDB and OPR buses. The two sign bits are examined independent of their respective fractions and are used to:

1. Generate the sign of the fraction product.
2. Condition the fraction normalize logic.
3. Control the exponent one's complement logic.
4. Condition the fraction two's complement logic (i.e., the two's complement logic associated with the partial product adder).

The sign of the fraction product is determined by an exclusive OR of the signs of the original fraction multiplier and multiplicand (sheets 29 and 24). If either one (but not both) of the original fractions is negative, the sign of the fraction product will be negative. If both are negative or positive, the sign of the fraction product will be positive.

The sign of the fraction product is loaded into the exponent output register (bit 0 position) (sheet 29) at the same time as the final exponent.

The exclusive OR function described above also provides an indication (POSITIVE or NEGATIVE) to the fraction normalize logic (sheet 26) as to whether the fraction product will be negative or positive.

1. If the sign is negative, the normalize logic is conditioned to test for leading Fs in the fraction product. Leading Fs in conjunction with a negative sign bit mean normalization is required.

2. If the sign is positive, the normalize logic is conditioned to test for leading zeros. Leading zeros in conjunction with a positive sign bit mean normalization is required.

There are three sets of exponent one's complement logic. Two sets are for the original exponents (sheet 27) and one set is for the exponent of the answer (sheet 29).

The one's complement network for each of the incoming exponents is controlled by the sign of the fraction associated with the particular (A or B) exponent. If the sign of the fraction is negative, the exponent of that fraction is one's complemented as it is input from the data bus.

The one's complement network for the final exponent (sheet 29) is controlled by the sign of the fraction product. If the fraction product is negative, the final exponent is one's complemented prior to being loaded into the exponent output register.

The sign bits of the original fractions (multiplier and multiplicand) are used to condition the fraction two's complement logic. To understand this application, refer to the section entitled two's complement handling that is provided later in this chapter (paragraph 2.2.1.7.2).

2.2.1.7 Fraction Multiply

A simplified diagram of the fraction multiply logic is provided in Figure 2-6.

The multiplication of fractions begins on the cycle after operand loading when the sequence control logic enables the output of the multiplier (X) and multiplicand (Y) onto the X and Y buses, respectively. The entire 56-bit multiplicand is output onto the Y bus. One byte of the multiplier is output onto the X bus. For single-precision operations only, the zero-extension register will output zeros into bit positions 32 through 63 of the Y bus.

The multiplication of fractions requires a total of four machine cycles (eight for double precision) to execute. Each of the first three cycles involves multiplying a 56-bit multiplicand by eight bits of the multiplier and adding (accumulation) the result (partial product) from the previous cycle. A different multiplier byte is used each cycle, starting with the least-significant byte (bits 24 through 31 for single precision; bits 56 through 63 for double precision). No multiplication occurs in the fourth cycle; instead, one final accumulate is performed. At the end of the fourth cycle, the fraction product resides in the partial product (PP) register.

2.2.1.7.1 Basic Steps

The text that follows provides a more detailed description of the basic steps for the multiplication of fractions. The two's complement function, which is an integral part of the fraction multiply, is described separately. For reference, a detailed block diagram of the fraction multiply logic is provided in Figure 2-7.

The basic steps for the first iteration of the fraction multiply are outlined as follows:

1. Multiply the least-significant byte of the multiplier times the entire multiplicand (bits 08 through 63). This produces a 112-bit result which propagates from the multiplier circuit to the multiplier output register.

   Note

   Bits 24 through 31 of X are used for the first iteration of a single-precision multiply. For double precision, X bits 56 through 63 are used instead.

2. The 112 bits are overlapped and added together in ALU1 of the PP adder logic to produce a 56-bit result.

3. The previous partial product currently in the PP output register (which is always zero for the first iteration) is shifted right eight bit positions and then added to the result from ALU1. This add is performed in ALU2. The

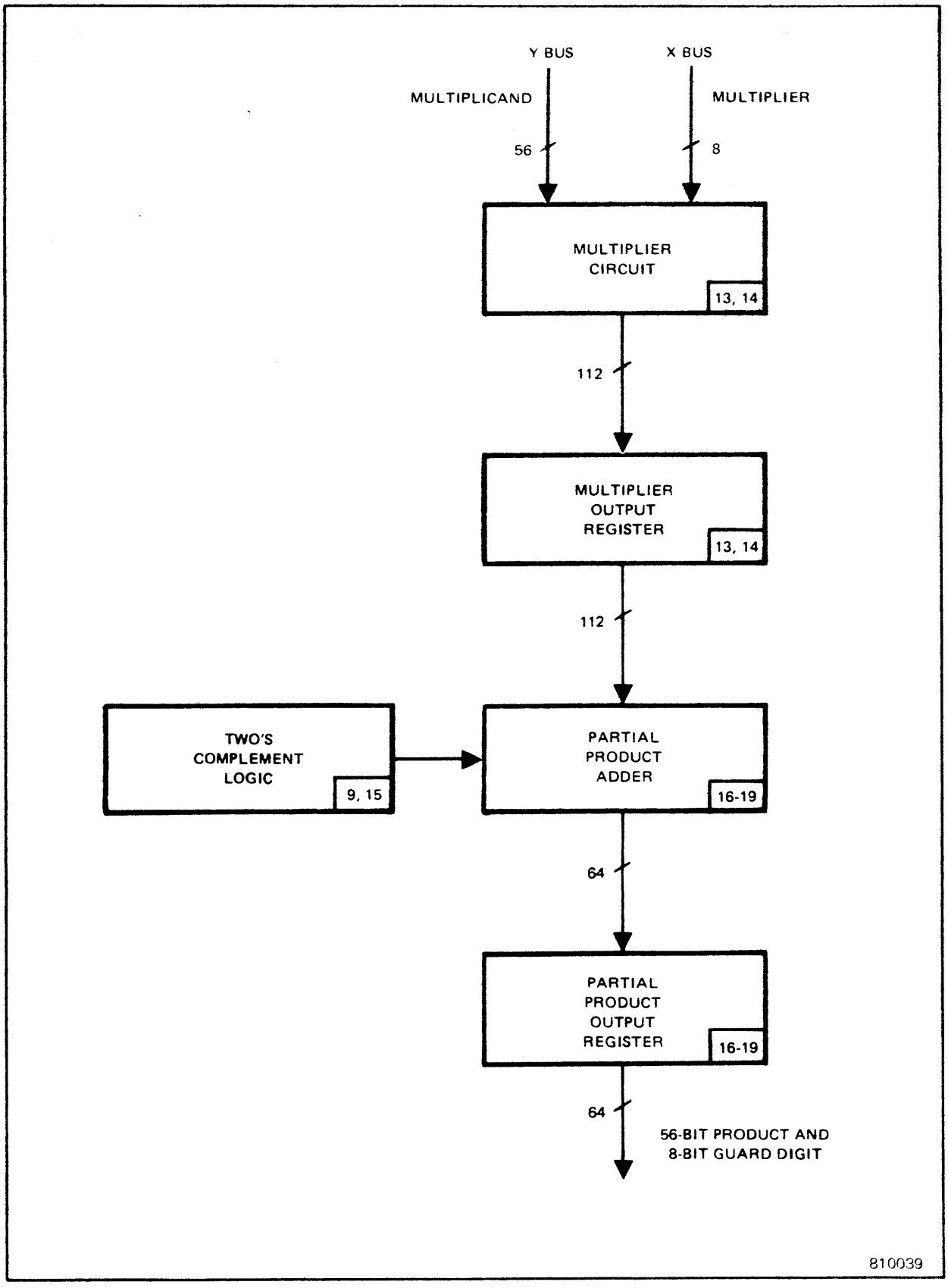

Figure 2-6. Multiply Logic - Simplified Block Diagram

previous partial-product value is zero only for the first iteration of the fraction multiply.

4. The result of the ALU2 addition propagates to the PP output register. One fraction multiply iteration is completed. The value that propagates into the PP output register at the end of the first iteration represents the first partial product.

Subsequent iterations follow the same basic steps outlined above. The differences are noted in the paragraphs that follow.

In each subsequent fraction multiply iteration, a progressively more significant byte of the multiplier (X) is multiplied times the entire 56-bit multiplicand (Y). This is performed in step 1 above. Only the bits of X differ each iteration.

Step 3 above is performed in the same manner during each subsequent iteration. However, the value in the PP output register for the second iteration (step 3) will be the partial product that was produced the first iteration. Thereafter, the value in the PP output register for each iteration (step 3) will be the result of the multiply from the previous iteration plus the sum of all previous partial products. Each iteration, the new partial product is added to the sum of all previous partial products. The process is cumulative. At the end of the last iteration, the value in the PP output register will be a 56-bit product plus an 8-bit guard digit.

Due to the 8-bit shift right that occurs each iteration (step 3), the most-significant byte of the first partial product will have been shifted into the least-significant byte position of the final product by the time the last iteration is completed.

2.2.1.7.2 Two's Complement Handling

Since there is an established convention that all negative floating-point fractions input to or output from the FPA will be in their two's complement form, this must be taken into account during the execution of the fraction multiply. The method used is broadly defined as follows:

1. When the fraction multiply occurs in the multiplier circuit, only the absolute values of the two fractions are used; that is, they are consideration of the sign of either fraction.

2. If either one or both of the original fractions was negative, then a two's complement value will be added to the result from step one. This is a function of the two's complement logic and the partial product adder.

The two's complement value that is added to the product of X times Y (if either one or both fractions are negative) is derived by the following method:

1. If Y is negative, one's complement X and add it to the product of X times Y.
2. If X is negative, one's complement Y and add it to the product of X times Y.
3. Add a binary constant to the product of X times Y:

   - if neither X nor Y are negative, add the binary constant of 00 to each partial product.

   - if either X or Y (but not both) are negative, add the binary constant of 01 to each partial product.

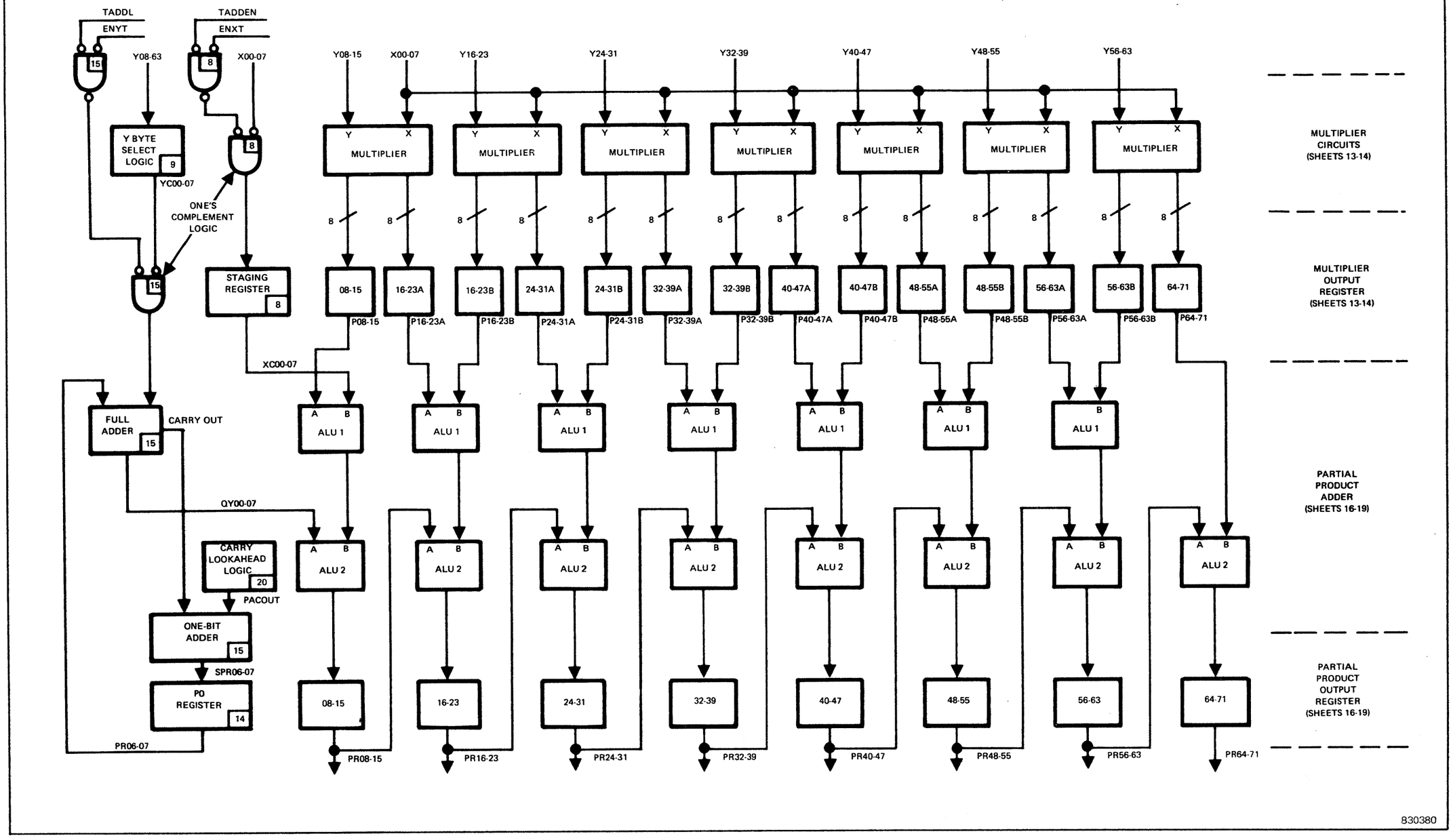

Figure 2-7. Multiply Logic - Detailed Block Diagram

- if both X and Y are negative, add the binary constant of 11 to each partial product.

As previously described, the manner in which partial products are shifted to the right by 8 bits each multiply iteration means that only the most-significant byte of each partial product will become part of the final answer. This fact makes it possible to add the two's complement value incrementally, by adding it to the most-significant byte of each new partial product.

The two's complement logic is included in Figure 2-7. The X and Y bits enter the two's complement logic from the X and Y bus, respectively, as shown at the top left portion of the diagram. Only the sign bits and one byte of each fraction are selected, per multiply iteration, starting with the least-significant byte of X and Y the first iteration. Each iteration thereafter, a progressively more-significant byte of X and Y is selected.

The operand input register for the multiplier (D/M register) has a byte select circuit that gates the appropriate multiplier (X) byte onto the X bus each iteration of the fraction multiply operation; this same multiplier (X) byte is also input to the two's complement logic. A special byte select circuit is required for Y. This circuit, which is illustrated in Figure 2-7, is necessary because the entire multiplicand (instead of just one byte) is used by the fraction multiply logic each iteration; therefore, the entire multiplicand will be present on the Y bus each iteration.

The selected X and Y bytes are input to a set of gates (logic sheets 8 and 15 respectively) which one's complement the X and/or Y bytes if the sign of their opposite fraction is negative. Whenever the sign is positive, the gates are disabled and their output is all zeros.

The output of the one's complement gates for the selected X byte is staged in a register latch (sheet 8) and then added to the first byte of the 56-bit product of X times Y. This add is performed in ALU1. ALU2 is used in a similar manner to add the output of the Y one's complement logic to the first byte output from ALU1. In addition, this Y value contains one of the binary constants: 00, 01, or 10. This constant is stored in the multiplier output register (sheet 14) and added to the Y value (which could be either zero or the one's complement of Y) in the full adder (sheet 15) illustrated in Figure 2-7.

Initially, the multiplier output register (sheet 14) is preset with the appropriate constant which is determined in accordance with rule three listed above. Thereafter, a one-bit adder (sheet 15) (see Figure 2-7) is used to track the carry-out from the first byte of ALU1 and the full adder. This provides for carry propagation into the first byte of each subsequent multiply iteration. Since only the first byte of each multiply iteration can become part of the final fraction answer, this carry-out (when there is a carry) is, in effect, propagating into the next more-significant byte of the final fraction answer with each multiply iteration.

2.2.1.8 Fraction Correction

Fraction correction for floating-point multiplication involves either normalizing or rounding the fraction product. Both single- and double-precision fractions are normalized when needed; however, the rounding function is applicable only to single-precision fractions.

One machine cycle is required to correct the fraction product and is a function of the normalize/complement/round logic (sheets 21 through 24) shown in Figure 2-4. For multiply, this logic provides the data structure for normalization and rounding. It also provides a two's complement function; however, this function is used only during divide.

2.2.1.8.1 Normalization

Normalization is performed by a normalize detector and a set of 2:1 multiplexers. The normalize detector (sheet 26) samples the output of the PP register (sheets 16 through 19) and determines when the normalization of the fraction product is needed. If normalization is needed, the detector provides the appropriate select signals to the 2:1 multiplexers (sheets 21 through 24) which steer the incoming fraction product to the left four bits (one hex digit). This effectively shifts bits 12 through 63 of the fraction product into bit positions 08 through 59, and zeros into 60 through 63.

A fractional product will never require a shift left of more than four bit positions to achieve normalization.

Normalization is influenced by the sign of the fraction. If the sign is negative, the normalize detector tests for leading Fs in the fraction product. All ones in the most-significant hex digit of the fraction product (when the sign is negative) will enable normalization. If the sign is positive, the normalize detector tests for leading zeros in the fraction product. All zeros in the most-significant hex digit of the fraction product (when the sign is positive) will enable normalization.

There is one exception to the above. Normalization is not enabled, when the sign is negative, if the fraction product equals F0000000.

2.2.1.8.2 Rounding

In floating-point multiplication, the rounding function is only available for single-precision operations. Even then, it is enabled only when normalization is not needed.

When enabled, rounding is performed by adding a logical one to the most-significant bit of the guard digit (bit 32). If bit 32 of the fraction product was already a one, then a carry will propagate into bit 31 and the fraction product will be rounded upward by a value of one.

The round function is performed in the complement/round ALU (sheets 21 through 24). The primary purpose of this ALU is for use during division. Rounding is its only function during floating-point multiply. When rounding is not enabled, all zeros are added to the fraction product.

2.2.1.9 Answer Output

The fraction product, fraction sign bit, and exponent are simultaneously loaded into their respective output registers under control of the sequence control logic (sheet 32):

1. Fraction product (HRC08 through 63) is loaded into the floating-point output register (sheet 30).

2. Fraction sign bit is loaded into bit 00 of the exponent output register (sheet 29).

3. Exponent is loaded into bits 01 through 07 of the exponent output register (sheet 29).

The output registers are loaded as a result of the LLDOUT signal from the sequence control logic. This signal is staged in a control flip-flop (sheet 14). The output of the control flip-flop (LLDOUTREG)

actually enables the loading of the above listed registers.

Simultaneous with loading the output registers, the output control logic (sheet 31) is enabled. This logic controls access to the EY bus by the M/D unit. The EY bus is shared by the CPU and the A/S and M/D units. If the EY bus is currently available, the contents of the output registers will be gated onto the bus and sent to the A/S unit's FPA file register on the next machine cycle after the loading of the output registers.

Access to the EY bus is prioritized. The list that follows defines the priority in descending order.

1. CPU (HCPUOUT)
2. A/S unit (LADDOUT)
3. M/D unit

Whenever the M/D unit board loads its output registers, the output control logic will test for the presence of HCPUOUT and LADDOUT. If either one is true, the EY bus is not currently available to the M/D unit. As soon as the bus is available, the output control logic will output the fraction, fraction sign bit, and the exponent onto the EY bus. Along with the data, the output control logic also transfers the address of the file register for which the data is intended. This four-bit address is sent on the file address bus (FADD00 through 03) from the RB address register (sheet 26) to the A/S unit. In addition to the data and the address, the output control logic sends the signal, LUNITSTORE, on a discrete line to the file register. This signal conditions the file register for a file write operation.

2.2.1.10 Arithmetic Exception

An arithmetic exception (AE) is an error indication. For floating-point multiplication, only two conditions will generate an AE: exponent underflow and exponent overflow.

If the M/D unit detects either one of the above conditions (sheet 29), the output control logic (sheet 31) sends an error indication (LAEDATA) to the FPA file register, along with the data, and notifies the CPU that an AE is pending (LAEPEND).

The error signal (LAEDATA), sent to the FPA file register, inhibits the file write operation. Thus, the original register operand is retained in the file.

The CPU tests for the presence of an AE when it reads the contents of the file register. The AE pending signal informs the CPU that the data in the file register is the original register operand and not the answer from the arithmetic operation.

After the CPU tests for, and detects, the presence of an AE, it can determine the type of AE by reading the arithmetic exception register which is located on the M/D unit. The contents of this register provide the CPU with the information required to properly set up the software-testable condition codes.

The AE register (sheet 29) is an eight-bit register whose contents are output to the ED bus (EDB16 through 23) upon request of the CPU (LAEEN; sheet 28). The contents of the AE register are shown in Table 2-13.

2.2.2 Floating-point Divide

The M/D unit executes both divide floating-point word (DVFW) and divide floating-point doubleword (DVFD) instructions. The paragraphs that follow provide a brief overview of the steps involved.

Operands are input from the CPU and the FPA file register.

Table 2-13
AE Register Contents

Bit	Definition
16-20	Not used.
21	Set if the sign of AE result is positive.
22	Set if the sign of AE result is negative.
23	Set if exponent underflow occurred. Reset (logical 0) if exponent overflow occurred.

Note

AE register bits 16 through 23 correspond to ED bus bits 16 through 23, respectively.

Instruction execution is initiated by the CPU.

The incoming floating-point divisor and dividend operands are disassembled and loaded into the fraction and exponent logic of the M/D unit. For single-precision operations, the fraction is a 24-bit hexadecimal number positioned in bits 08 through 31 of the incoming word operand. For double-precision operations, the fraction is a 56-bit hexadecimal number positioned in bits 08 through 63 of the incoming doubleword operand. If either one or both of the incoming fractions has a negative sign (bit 0), the exponent (bits 01 through 07) of the negative fraction is one's complemented.

After the incoming operands are disassembled and loaded into the fraction and exponent logic, the division of fractions is initiated. Simultaneous with the division of fractions, exponent calculations are performed in the exponent logic.

Fraction division is performed by first calculating the reciprocal of the divisor and then multiplying that reciprocal value by the dividend to obtain the quotient.

The quotient is corrected by enabling both normalization (if needed) and rounding.

If the fraction answer (quotient) is negative, then the biased exponent of the answer will be one's complemented.

The final fraction, fraction sign, and exponent are assembled into the correct format for a single- or double-precision floating-point operand, and stored in the register file.

2.2.2.1 Simplified Sequence of Events

1. Load operands

   a. Input the fraction sign bits and the exponents of both operands from the OPR and EDB buses into the exponent input logic (sheet 27).

   b. If either one (or both) of the fraction sign bits is negative, one's complement the exponent associated with the negative fraction (sheet 27).

c. Load the exponents into the exponent input registers (sheet 27).

d. Load the fractions of the dividend and divisor from the OPR and EDB buses repectively into the fraction logic (sheets 2, 4, and 10).

Note

One machine cycle is required to load single-precision operands; two machine cycles are required to load double-precision operands.

e. Input the destination file address from the EB bus into the RB address logic.

2. a. Calculate the reciprocal of the divisor.

b. Calculate the exponent of the reciprocal.

3. Multiply the dividend and the reciprocal of the divisor together to obtain the fraction quotient.

4. Add the dividend exponent and reciprocal exponent. The result is the biased exponent of the answer.

5. Correct the fraction answer obtained in step three:

Normalize the fraction answer if needed.
Exception: Do not normalize negative answer if fraction portion = F0000000

Enable rounding of the fraction answer regardless of whether normalization was performed.

6. a. If either one (but not both) of the original operands had a negative fraction, one's complement (sheet 29) the biased exponent produced in step 4.

b. If either one (but not both) of the original operands had a negative fraction, set the sign of the fraction answer to a logical one (sheet 29).

c. Load the sign bit of the fraction answer and the biased exponent into the exponent output register (sheet 29).

Exception: If the fraction answer from step three is all zeros, force the exponent of the answer to zero and inhibit the exponent one's complement function.

d. Load the corrected fraction answer from step five into the floating-point output register (sheet 30).

7. Gate the contents of the exponent output and floating-point output registers into their respective bit position on the external Y (EY) bus. Register contents are shown in Table 2-14.

Table 2-14
Output Register Contents

Single-precision answers:		
Exp. Out. Reg. Flt. Out. Reg.	EY00-07 EY08-31	Cycle n
Double-precision answers:		
Exp. Out. Reg. Flt. Out. Reg. 08-31	EY00-07 EY08-31	Cycle n
Flt. Out. Reg. 32-63	EY00-31	Cycle n+1

8. Output the destination file address from the RB address logic onto the file address bus (FADD00 through 03). This is the address of the file register(s) to which the answer is destined.

2.2.2.2 Operand Loading

Operand loading for floating-point division is basically the same as for floating-point multiply. The reader should refer to the description of operand loading for floating-point multiply. The text that follows primarily focuses on the differences.

The memory operand (divisor) and the register operand (dividend) are input to the FPA B board on the EDB and OPR buses, respectively. For single-precision operations, bits 08 through 31 of the memory operand (divisor) are input from the EDB bus into bit positions 08 through 31 of the multiplicand/divisor (M/D) register (sheet 2). Bits 00 through 07 (fraction sign bit and exponent) are input into the exponent B input register (sheet 27).

For double-precision operations, two machine cycles are required to complete operand loading. Bits 32 through 63 of the fraction are loaded first (into the M/D register) on the first bus transfer. The MSW (which contains the sign bit, exponent and the most significant 24 bits of the fraction) is loaded into the exponent B input register, and the M/D register, on the second bus transfer.

The register operand (dividend) is similarly loaded into the dividend/multiplier (D/M) register (sheet 10) and the exponent A input register (sheet 27). In addition, a copy of the fraction portion of the dividend is also loaded into the double-precision dividend (sheet 4) register. The contents of this register are used only during double-precison divide operations.

2.2.2.3 Destination File Address

The destination file address is handled in the same manner for the floating-point divide instructions as that previously described for the floating-point multiply (paragraph 2.2.1.4).

2.2.2.4 Fraction Division

The divide algorithm uses multiplication in many instances. To understand this portion of the algorithm, the reader should refer to the description of fraction multiply provided earlier in this chapter

(paragraph 2.2.1.7). That information is not repeated in the text that follows.

As previously stated, division is performed by multiplying the dividend by the reciprocal of the divisor. Most of the divide algorithm involves calculating the correct reciprocal. The reciprocal is calculated by starting with an approximate reciprocal value and incrementally correcting it until an acceptable degree of accuracy is obtained.

The initial reciprocal approximation resides in a programmable read-only-memory (PROM) and is based on the most-significant twelve bits of the divisor. For single-precision operations, the initial reciprocal approximation will have to be corrected twice to obtain the final reciprocal. For double-precision operations, three corrections are required to obtain the final reciprocal.

Since it is known that the product of any number (n) and its reciprocal (r) should equal one, the degree of accuracy of the reciprocal approximation can be calculated by multiplying it by the divisor. The product of this multiplication is called the error factor of the reciprocal. The error factor will be closer to the value of 1.0 each time a more accurate reciprocal is used in the calculation.

Each new reciprocal approximation is calculated by multiplying the preceding reciprocal approximation and its error factor together.

The following is a simplified outline of the steps used to obtain the final reciprocal:

1. Look up the first reciprocal approximation in the reciprocal PROM.
2. Calculate the first error factor by multiplying the first reciprocal and absolute value of the divisor together.
3. Find the second reciprocal by multiplying the first error factor and the first reciprocal together.
4. Calculate the second error factor by multiplying the second reciprocal and the absolute value of the divisor together.
5. Find the third reciprocal by multiplying the second error factor and the absolute value of the divisor together.

For single-precision operations, the third reciprocal approximation will be used as the final reciprocal. Double-precision operations require an additional iteration.

A more detailed outline of the steps required for fraction division, including the hardware data paths, is provided as follows:

1. Find the first reciprocal approximation of the divisor:
   a. Obtain the absolute value of the divisor:
      Multiply the divisor by a constant of one. This multiply is simply to get the divisor through the multiply pipeline to the complement logic.

      If the divisor is negative, take the two's complement of the value produced in the preceding step. This is performed in the complement/round ALU. The result is the absolute value of the divisor.

b. Load the absolute value of the divisor into the ABSY register (sheet 5).

c. Use the absolute value of the divisor to look up the first reciprocal approximation in the reciprocal PROM (sheet 12).

d. Load the first reciprocal approximation into the reciprocal PROM output register (sheet 12).

2. Calculate the error factor of the first reciprocal:

a. Multiply the absolute value of the divisor by the first reciprocal approximation.

b. Two's complement the product from step 2a. This is performed in the complement/round ALU (sheets 21 through 23).

c. Shift the two's complemented value to the right one hexadecimal digit, and load it into the T right register.

3. Find the second reciprocal approximation:

a. Multiply the error factor resident in the T right register by the first reciprocal approximation (which is located in the reciprocal PROM output register).

b. Load the product from step 3a into the R register (sheet 11). This is the second reciprocal approximation.

4. Calculate the error factor of the second reciprocal:

a. Multiply the absolute value of the divisor by the second reciprocal approximation.

b. Two's complement the product from step 4a. This is performed in the complement/round ALU.

c. Shift the two's complemented value to the right one hexadecimal digit, and load it into the T right register.

5. Find the third reciprocal approximation:

a. Multiply the error factor resident in the T right register by the second reciprocal approximation which is in the R register.

b. For single-precision operations the product from step 5a is the final reciprocal. If the original divisor was negative, take the two's complement of the final reciprocal using the complement/round ALU. Load the final reciprocal (or, its two's complement) into the ABSY register.

For double-precision operations the product from step 5a is the third reciprocal and is loaded into the R register.

Single Precision:

6. Calculate the quotient: multiply the reciprocal of the divisor (ABSY register) by the dividend (D/M register).

7. Correct the quotient by normalizing and rounding.

8. Load the fraction quotient into the floating-point output register (sheet 30).

Double Precision:

6. Calculate the error factor of the third reciprocal approximation:

   a. Multiply the absolute value of the divisor by the third reciprocal approximation.

   b. Two's complement the product from step 6a. This is performed in the complement/round ALU.

   c. Shift the two's complemented value to the right one hexadecimal digit, and load it into the T right register.

7. Find the fourth reciprocal approximation:

   a. Multiply the error factor in the T right register by the third reciprocal approximation.

   b. If the original divisor was negative, take the two's complement of the value produced in step 7a. This is performed in the complement/round ALU.

   c. Load the product from step 7a (or 7b if applicable) into the R register. This is the final reciprocal of the divisor.

8. Calculate the quotient: Multiply the dividend (double-precision dividend register) by the reciprocal of the divisor (R register).

9. Correct the quotient by normalizing and rounding.

10. Load the double-precision fraction quotient into the floating-point output register.

2.2.2.5 Exponent Handling

Exponent handling for divide includes the following basic steps:

1. When exponents are input from the data buses, one's complement either exponent if its associated fraction is negative.

2. Calculate the exponent of the divisor's reciprocal approximation.

3. Adjust the exponent calculated in step two each time a new reciprocal is derived.

4. Add the exponents of the final reciprocal and the dividend.

5. a. Decrement the exponent obtained in step four if the fraction answer (quotient) had to be normalized.

   b. Increment the exponent obtained in step four if the fraction had to be shifted right (one hexadecimal digit) to correct for fraction overflow.

Note

Fraction overflow could have occurred as the result of rounding.

6. One's complement the final exponent if the sign of the fraction answer is negative.

7. Load the exponent output register.

8. Gate the exponent onto the EY bus (EY00 through 07).

The exponent logic used during divide operations is illustrated in Figure 2-8.

When the exponents are input from their respective data buses, each exponent passes through a separate exclusive OR network located in the data path between the data bus input and the A and B exponent input registers (see Figure 2-8). Each exclusive OR network is fed by one of the exponents (A or B) and the sign bit of the fraction associated with that particular exponent. If the sign bit is a logical one (indicating a negative fraction), the exclusive OR network will one's complement the exponent.

Calculation of the initial reciprocal exponent is performed by subtracting the exponent of the divisor from the constant 82. This subtraction is performed in the exponent ALU. The result is loaded into the exponent ALU register when the sequence control logic issues LLDEXPL. The output of the exponent ALU register propagates through the full adder (which free-runs with an add zero function) and is asserted on the TEXP00 through 07 lines (see Figure 2-8).

One cycle after the exponent ALU register is loaded, the initial reciprocal exponent is clocked into both TEXP registers.

While the fraction logic is calculating the second reciprocal approximation, the initial exponent reciprocal will be adjusted by adding to it the constant of 41. This is performed in the exponent ALU after the contents of the TEXP A register and the constant 41 register are gated to the A and B inputs, respectively, of the exponent ALU. The result is loaded into the exponent ALU register when LLDEXPL is issued; one cycle later it will be loaded into the TEXP registers.

The reciprocal exponent adjustment described above occurs twice during single-precision operations (i.e., coincident with calculation of the second and third reciprocal approximation in the fraction logic) and three times during double-precision operations (i.e., coincident with calculation of the second, third, and fourth reciprocal approximations).

After the fraction logic has calculated the final reciprocal of the divisor, it will find the quotient by multiplying the dividend and the divisor reciprocal together. When this occurs, the exponent logic will calculate the exponent of the quotient by adding together the exponent of the dividend and the exponent of the reciprocal. These two exponents reside (at this time) in the exponent A register and the TEXP B register, respectively. They are added together in the exponent ALU, and the result is loaded into the exponent ALU register when the sequence control logic issues the load exponent signal, LLDEXPL. If the fraction quotient is zero, the exponent ALU register will be reset to force the value of the exponent to zero (LFRCZEXP).

The output of the exponent ALU register propagates to the exponent full adder (sheet 29). The exponent full adder will decrement the quotient exponent by one if the quotient had to be normalized. The exponent will never have to be decremented by a value greater than one because the fraction quotient will never need to be shifted by more than one hexadecimal digit to achieve normalization.

The exponent full adder will increment the quotient exponent by one if the quotient had to be shifted right (one hexadecimal digit) to correct for fraction overflow. This would occur, for example, if rounding caused fraction overflow.

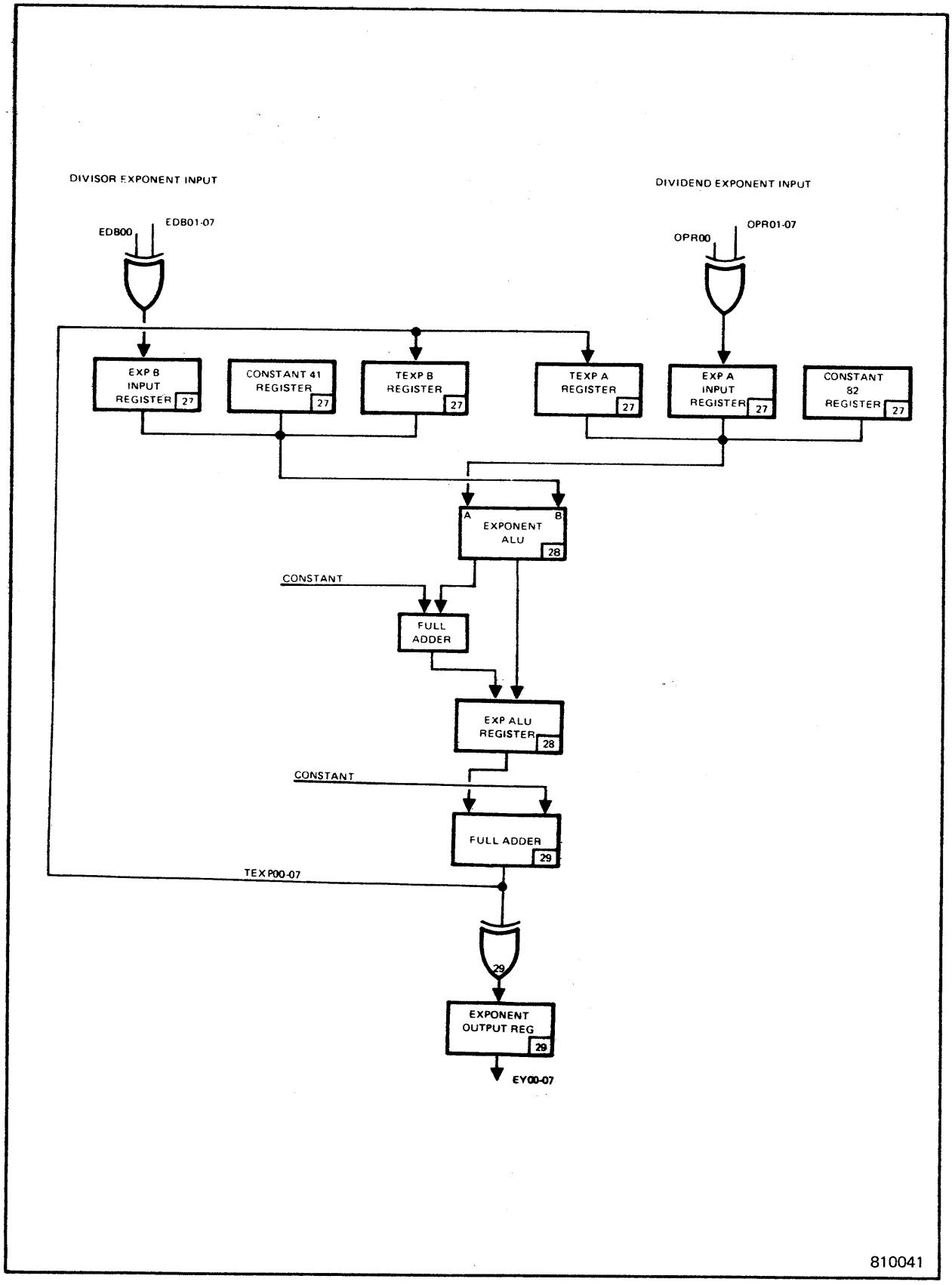

Figure 2-8. Exponent Logic Block Diagram (Floating-point Divide)

If the sign of the fraction quotient is negative, the exponent will be one's complemented prior to being loaded into the exponent output register (sheet 29). One's complementing the exponent will be inhibited if the exponent was forced to zero in the exponent ALU register.

The exponent output register is loaded with the sign of the fraction and the seven-bit biased exponent when the sequence control logic (sheet 32) issues the load output registers signal (LLDOUT). Loading the exponent output register occurs simultaneous with the loading of the fraction into the floating-point output register.

2.2.2.6 Sign Bit Manipulation

The sign bits of both operands enter the M/D unit board along with two exponents and fractions from the EDB and OPR buses. The two sign bits are examined independent of their respective fractions and are used to:

1. Generate the sign of the fraction answer.
2. Condition the fraction normalize logic.
3. Control the exponent one's complement logic.
4. Condition the fraction two's complement logic (i.e., the two's complement logic associated with the partial product adder).
5. Condition the two's complement function of the complement/round ALU logic.

With two exceptions, the above listed functions are performed in the same manner as previously described for the floating-point multiply instruction (paragraph 2.2.1.6). Sign bit manipulation differs for floating-point division only in regard to items two and five above. These are explained in the text that follows.

2.2.2.6.1 Conditioning the Normalize Logic

When handling the fraction answer, the normalize logic is conditioned by an exclusive OR of the two original sign bits (sheet 24). This function is the same for both floating-point divide and floating-point multiply.

Conditioning of the normalize logic in floating-point divide is handled differently only during calculation of the reciprocal. The reciprocal calculation includes a series of steps that involve multiplying. It is necessary that the normalize logic treat each product as a positive value. That is, normalization will be invoked only when the product has leading zeros (i.e., all zeros in the most-significant hexadecimal digit or beyond). To accomplish this, the sequence control logic issues a force positive signal (LFRCPOS) which overrides the output of the sign bit exclusive OR circuit (sheet 24). The resulting signal (HPOSITIVE) conditions the normalize detection logic (sheet 26) to treat the data (product) as a positive value.

During floating-point division, the force positive signal (LFRCPOS) is issued by the sequence control logic, after each multiply, through calculation of the final reciprocal value.

2.2.2.6.2 Two's Complement Conditioning (Complement/Round ALU)

The sign of the original divisor fraction is used in two instances during the reciprocal calculations to

condition the two's complement enable signals (LCOMPL1EN and LCOMPL2EN; sheet 25) for the complement/round ALU.

If the sign of the original divisor was negative, the two's complement function of the complement/round ALU will be enabled:

1. To obtain the absolute value of the divisor. This occurs just prior to look-up of the initial reciprocal approximation and is initiated by the LQUALCOMPEN signal from the sequence control logic.

2. To take the two's complement of the final reciprocal. This occurs just prior to loading the final reciprocal into the ABSY register (or the R register for double-precision operations) and is initiated by the LQUALCOMPEN signal from the sequence control logic.

2.2.2.7 Fraction Correction

Fraction correction for floating-point division includes the capability of both normalizing and rounding the fraction answer. This applies to both single- and double-precision fraction answers and requires only one machine cycle to execute. Prior to normalization and rounding, the uncorrected fraction answer at the output of the PP register has two guard digits (see Figures 2-9 and 2-10). This provides an additional degree of precision and makes it possible to both normalize and round.

2.2.2.7.1 Normalization

Normalization is performed by a normalize detector (sheet 26) and a set of 2:1 multiplexers (sheets 21 through 24). The normalize detector samples the contents of the PP register (sheets 16 through 19) and determines when the normalization of the fraction answer is needed. If normalization is needed, the detector provides the appropriate select signals to the 2:1 multiplexers which steer the incoming fraction answer to the left four bits (one hexadecimal digit).

When normalization is performed, the entire fraction answer will be shifted left one hexadecimal digit causing the data in bit positions 12 through 71 to be shifted into bit positions 08 through 67. Bit positions 68 through 71 are zero-filled (refer to Figures 2-9 and 2-10). A fraction answer will never require a shift left of more than four bit positions to achieve normalization. If normalization of the fraction occurs, the exponent will be decremented by a value of one.

Normalization is influenced by the sign of the fraction. If the sign is negative, the normalize detector tests for leading ones in the fraction answer. All ones in the most-significant hex digit of the fraction answer (when the sign is negative) will enable normalization. If the sign is positive, the normalize detector tests for leading zeros in the fraction answer. All zeros in the most-significant hex digit of the fraction answer (when the sign is positive) will enable normalization.

There is not exception to the above. Normalization is not enabled when the sign is negative if the fraction answer equals F0000000.

2.2.2.7.2 Rounding

In floating-point division, rounding is enabled for both single- and double-precision operations regardless of whether normalization occurs. The round function is

STEP	FRACTION						GUARD	GUARD								
1	0	5	F	2	3	A	7	7	0	0	0	0	0	0	0	0
2	5	F	2	3	A	7	7	0	0	0	0	0	0	0	0	0
3	5	F	2	3	A	7	7	0	0	0	0	0	0	0	0	0
BITS	8					31	32-35	36-39	40							71

ASSUMES SIGN BIT = LOGICAL ZERO (POSITIVE)

NOTES

1 STEP 1 REPRESENTS THE OUTPUT OF THE PP REGISTER.

2 STEP 2 REPRESENTS THE OUTPUT OF THE NORMALIZATION MULTIPLEXERS. SINCE THIS IS A POSITIVE FRACTION AND THERE WERE LEADING ZEROS IN THE DATA REPRESENTED IN STEP 1, NORMALIZATION WAS PERFORMED.

3 STEP 3 REPRESENTS THE DATA OUTPUT TO THE FLOATING-POINT OUTPUT REGISTER. ALTHOUGH ROUNDING WAS ENABLED, IT DID NOT ALTER THE SINGLE-PRECISION FRACTION SINCE THE MOST - SIGNIFICANT GUARD DIGIT CONTAINED A VALUE LESS THAN EIGHT (HEX).

810045

Figure 2-9. Single-precision Normalization and Rounding Example

STEP	FRACTION														GUARD	GUARD
1	0	5	F	2	3	A	7	7	9	F	F	B	B	B	7	9
2	5	F	2	3	A	7	7	9	F	F	B	B	B	7	9	0
3	5	F	2	3	A	7	7	9	F	F	B	B	B	8		
BITS	8													63	64-67	68-71

ASSUMES BIT = LOGICAL ZERO (POSITIVE)

NOTES

1 STEP 1 REPRESENTS THE OUTPUT OF THE PP REGISTER.

2 STEP 2 REPRESENTS THE OUTPUT OF THE NORMALIZATION MULTIPLEXERS. SINCE THIS IS A POSITIVE FRACTION AND THERE WERE LEADING ZEROS IN THE DATA REPRESENTED IN STEP 1, NORMALIZATION WAS PERFORMED.

3 STEP 3 REPRESENTS THE DATA AFTER THE NORMALIZED FRACTION HAS BEEN ROUNDED AND OUTPUT TO THE FLOATING-POINT OUTPUT REGISTER. NOTE THAT ONLY BITS 8-63 ARE OUTPUT.

810046

Figure 2-10. Double-precision Normalization and Rounding Example

performed in the complement/round ALU (sheets 21 through 24).

For single-precision fractions, rounding is performed by adding a logical one to the most-significant bit to the guard digit (bit 32). If bit 32 was already a one, then a carry will propagate into bit 31 and the fraction answer will be rounded upward by a value of one.

For double-precison fraction, rounding is performed by adding a logical one to the most-significant bit of the guard digit (bit 64). If bit 64 was already a one, then a carry will propagate into bit 63 and the fraction answer will be rounded upward by a value of one.

Rounding is possible after normalization due to the fact that there are two guard digits, and because it is never necessary to shift left by more than one hexadecimal digit to achieve a normalized answer. After normalization, only the least-significant guard digit is zero filled; the most-significant guard digit (bits 64 through 67 or 32 through 35) will contain valid data. The guarantee that there will always be valid data in the most-significant guard digit makes enabling rounding worthwhile. Although rounding is enabled, the value of the fraction will not be rounded upward unless the most-significant guard digit is equal to or greater than eight (i.e., the most-significant bit of the guard digit has to be a logical one).

If rounding causes fraction overflow, this is corrected by effectively shifting the fraction to the right one hexadecimal digit and incrementing the exponent by one. For example, rounding will cause fraction overflow if the fraction quotient is maximum positive (i.e., positive sign bit and all ones in the fraction). The right shift is implemented by forcing a one into bit position 11 of the fraction. This is performed in the floating-point output register (sheet 30) when the HSETMSDONE signal is generated by the round control logic (sheet 25). Forcing bit 11 to a one is effectively the same as shifting an overflow bit to the right one hexadecimal position.

2.2.2.8 Answer Output

Answer output for floating-point division is the same as previously described for the floating-point multiply instruction (paragraph 2.2.1.9).

2.2.2.9 Arithmetic Exception

The arithmetic exception (AE) logic has one function that is applicable only during floating-point division. Otherwise, the detection and reporting of an arithmetic exception is the same as previously described for floating-point multiplication (paragraph 2.2.1.10).

In floating-point division, if the incoming divisor fraction is equal to zero, an arithmetic exception will always result.

A test for the presence of this condition is made early in the divide algorithm. If the divisor equals zero, a flag will be set (sheet 24) for later use. The divide operation then continues normally. At the end of the divide operation, the presence of the flag will cause an AE indication to be generated (sheet 29).

The test for the divisor equal-to-zero condition is a function of the zero detection logic (sheets 21, 22, 23 and 26). The test is performed just prior to the look-up of the initial reciprocal approximation (after the divisor propagates through

the multiply pipeline). If the divisor equals zero, the LFRCZEXP signal (sheet 26) will be generated. This signal, in conjunction with the LSETAE and LDIV signals from the sequence control logic, will set the divisor equal-to-zero flag.

2.2.2.10 Sequence Control

An explanation of the sequence control logic is included in the text for the floating-point multiply instruction (paragraph 2.2.1.2).

Tables 2-15, 2-16, 2-17, and 2-18 provide the PROM control codes and sequence control signals applicable to the floating-point divide instructions.

2.2.3 Fixed-point Multiplication

The FPA B board executes the following fixed-point instructions:

1. Multiply by Memory Byte (MPMB)
2. Multiply by Memory Halfword (MPMH)
3. Multiply by Memory Word (MPMW)
4. Multiply Register by Register (MPR)
5. Multiply Immediate (MPI)

All five of the above instructions are executed in the same manner by the M/D unit and follow the same sequence of steps.

For each fixed-point instruction executed, the M/D unit receives two 32-bit operands. This is true for both the byte and halfword operands as well as the word operands. After the byte operand for an MPMB instruction is fetched from memory by the CPU, the CPU creates a 32-bit format prior to loading the operand into the M/D unit. This format, which is illustrated in Chapter 1, has the byte operand in the least-significant byte position (bits 24 through 31). The three most-significant bytes are zero-filled by the CPU. Similarly, the halfword operand (for the MPMH instruction) is fetched by the CPU and aligned in the least-significant halfword position of a 32-bit format. The most-significant halfword is sign-extended by the CPU. The alignment and zero-fill or sign-extend functions are performed prior to loading the operands into the FPA.

The multiply algorithm is the same for multiplication of the fixed-point operands as that previously described for multiplying floating-point fractions (paragraph 2.2.1.7).

There are no exponents involved in fixed-point operations.

No normalization or rounding is performed, and no arithmetic exceptions are generated for fixed-point operations.

Since fixed-point multiplication involves two 32-bit operands, provision is made for the storing of a doubleword answer.

2.2.3.1 Simplified Sequence of Events

1. Load two 32-bit operands and the destination file address.
2. Multiply operands.
3. Load doubleword answer into the fixed-point output register.
4. Transfer the destination file address and the doubleword answer to the FPA file register (A/S unit). This requires two EY bus transfers.

2.2.3.2 Operand Loading

Two 32-bit operands are loaded simultaneously into the M/D unit

Table 2-15
Sequence Control PROM Coding - Divide Floating-point Word

Address	PROM MB84		PROM MB81	
	Field A	Field B	Field C	Field D
	8 7 6 5	4 3 2 1	8 7 6 5	4 3 2 1
040	0 1 1 0	0 0 0 0	0 0 0 0	0 1 1 1
041	0 0 1 1	0 0 0 0	0 1 1 1	1 1 1 0
042	0 0 1 0	0 0 0 0	1 0 1 0	1 0 0 1
043	0 0 1 0	1 1 0 1	0 0 0 0	0 0 0 0
044	0 0 1 0	0 0 0 0	0 1 0 0	0 1 1 0
045	0 0 1 1	0 0 0 0	0 1 0 0	0 0 0 0
046	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 0
047	0 0 1 0	0 0 0 0	1 0 1 1	1 0 0 1
048	0 0 1 0	0 0 0 0	0 0 1 1	0 1 1 0
049	0 0 1 1	0 0 0 0	0 0 1 1	0 0 0 0
04A	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 0
04B	0 0 1 0	0 0 0 0	0 0 0 0	1 0 0 1
04C	0 0 1 0	1 1 0 0	0 1 0 0	0 0 1 1
04D	0 0 1 1	0 0 0 0	0 1 0 0	0 0 0 0
04E	0 0 1 0	0 0 0 0	0 1 0 0	0 0 0 0
04F	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 0
050	0 0 1 0	0 0 0 0	1 0 1 1	1 0 0 1
051	0 0 1 0	0 0 0 0	0 0 1 1	0 0 1 1
052	0 0 1 1	0 0 0 0	0 0 1 1	0 0 0 0
053	0 0 1 0	0 0 0 0	0 0 1 1	0 0 0 0
054	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 0
055	0 0 1 0	0 0 0 0	1 0 1 0	1 0 0 1
056	0 0 1 0	0 0 0 0	0 1 0 0	1 1 0 0
057	0 0 0 1	1 0 0 1	0 1 0 0	1 1 0 0
058	0 0 0 0	0 0 0 0	0 1 0 0	0 0 0 1
059	0 0 0 0	1 0 0 0	0 1 0 0	0 0 0 0
05A	0 0 0 0	0 0 0 0	0 1 0 0	0 0 0 0
05B	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 0
05C	1 0 1 0	0 0 0 0	1 0 0 1	1 1 1 1

Table 2-16
Active Sequence Control Signals - Divide Floating-point Word

Step	PROM MB84		PROM MB81	
	Field A	Field B	Field C	Field D
1	HSTOPCLKEN		LSPZEN	L1CNSTEN
2	HMULTCLR		LSUBTEXPEN	LLDEXP
3			LQUALCOMPEN	LFRCPOS
4		LSETAE	LSPZEN	NOP
5			LABSYEN	LPROMSTRT
6	HMULTCLR		LABSYEN	NOP
7			LSPZEN	LLDEXP
8			L2-XYEN	LFRCPOS
9			LTEN	LPROMSTRT
10	HMULTCLR		LTEN	NOP
11			LSPZEN	LLDEXP
12			LSPZEN	LFRCPOS
13		LREN	LABSYEN	LR1STRT
14	HMULTCLR		LABSYEN	NOP
15			LABSYEN	NOP
16			LSPZEN	LLDEXP
17			L2-XYEN	LFRCPOS
18			LTEN	LR1STRT
19	LMULTCLR		LTEN	NOP
20			LTEN	NOP
21			LSPZEN	LLDEXP
22			LQUALCOMPEN	LFRCPOS
23			LABSYEN	L0CNSTEN
24	HMULTCLR	LTPPA2STRT	LABSYEN	L0CNSTEN
25			LABSYEN	LX1STRT
26		LTPPA1STRT	LABSYEN	NOP
27			LABSYEN	NOP
28			LSPZEN	LLDEXP
29	HDONE		LRNDEN	LLDOUT

Table 2-17
Sequence Control PROM Coding - Divide Floating-point Doubleword

Address	PROM MB84		PROM MB81	
	Field A	Field B	Field C	Field D
	8 7 6 5	4 3 2 1	8 7 6 5	4 3 2 1
140	0 1 1 0	0 0 0 0	0 0 0 1	0 1 1 1
141	0 0 1 1	0 0 0 0	0 1 1 1	1 1 1 0
142	0 0 1 0	1 1 1 0	1 0 1 0	1 0 0 1
143	0 0 1 0	1 1 0 1	0 0 0 0	0 0 0 0
144	0 0 1 0	0 0 0 0	0 1 0 0	0 1 1 0
145	0 0 1 1	0 0 0 0	0 1 0 0	0 0 0 0
146	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 0
147	0 0 1 0	0 0 0 0	1 0 1 1	1 0 0 1
148	0 0 1 0	0 0 0 0	0 0 1 1	0 1 1 0
149	0 0 1 1	0 0 0 0	0 0 1 1	0 0 0 0
14A	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 0
14B	0 0 1 0	0 0 0 0	0 0 0 0	1 0 0 1
14C	0 0 1 0	1 1 0 0	0 1 0 0	0 0 1 1
14D	0 0 1 1	0 0 0 0	0 1 0 0	0 0 0 0
14E	0 0 1 0	0 0 0 0	0 1 0 0	0 0 0 0
14F	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 0
150	0 0 1 0	0 0 0 0	1 0 1 1	1 0 0 1
151	0 0 1 0	0 0 0 0	0 0 1 1	0 0 1 1
152	0 0 1 1	0 0 0 0	0 0 1 1	0 0 0 0
153	0 0 1 0	0 0 0 0	0 0 1 1	0 0 0 0
154	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 0
155	0 0 1 0	0 0 0 0	0 0 0 0	1 0 0 1
156	0 0 1 0	1 1 0 0	0 1 0 0	0 1 0 0
157	0 0 1 1	0 0 0 0	0 1 0 0	0 0 0 0
158	0 0 1 0	0 0 0 0	0 1 0 0	0 0 1 1
159	0 0 1 0	0 0 0 0	0 1 0 0	0 0 0 0
15A	0 0 1 0	0 0 0 0	0 1 0 0	0 0 0 0
15B	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 0
15C	0 0 1 0	0 0 0 0	1 0 1 1	1 0 0 1
15D	0 0 1 0	0 0 0 0	0 0 1 1	0 1 0 0
15E	0 0 1 1	0 0 0 0	0 0 1 1	0 0 0 0
15F	0 0 1 0	0 0 0 0	0 0 1 1	0 0 1 1
160	0 0 1 0	0 0 0 0	0 0 1 1	0 0 0 0
161	0 0 1 0	0 0 0 0	0 0 1 1	0 0 0 0
162	0 0 1 0	0 0 0 0	0 0 0 0	1 1 1 0
163	0 0 1 0	1 1 1 0	1 0 1 0	1 0 0 1
164	0 0 1 0	1 1 0 0	0 1 0 1	1 0 1 1
165	0 0 0 1	1 0 1 0	0 1 0 1	0 1 0 1
166	0 0 0 0	1 0 1 1	0 1 0 1	0 0 0 0
167	0 0 0 0	0 0 0 0	0 1 0 1	0 1 0 0
168	0 0 0 0	1 0 0 1	0 1 0 1	0 0 0 0
169	0 0 0 0	0 0 0 0	0 1 0 1	0 0 1 1
16A	0 0 0 0	1 0 0 0	0 1 0 1	0 0 0 0
16B	0 0 0 0	0 0 0 0	0 1 0 1	0 0 0 0
16C	0 0 0 0	0 0 0 0	0 1 0 1	1 1 1 0
16D	1 0 1 0	1 1 1 0	1 0 0 1	1 1 1 0

Table 2-18
Active Sequence Control Signals - Divide Floating-point Doubleword

Step	PROM MB84		PROM MB81	
	Field A	Field B	Field C	Field D
1	HSTOPCLKEN		LDPYEN	L1CNSTEN
2	HMULTCLR		LSUBTEXPEN	LLDEXP
3		LDPDIV	LQUALCOMPEN	LFRCPOS
4		LSETAE	LSPZEN	NOP
5			LABSYEN	LPROMSTRT
6	HMULTCLR		LABSYEN	NOP
7			LSPZEN	LLDEXP
8			L2-XYEN	LFRCPOS
9			LTEN	LPROMSTRT
10	HMULTCLR		LTEN	NOP
11			LSPZEN	LLDEXP
12			LSPZEN	LFRCPOS
13		LREN	LABSYEN	LR1STRT
14	HMULTCLR		LABSYEN	NOP
15			LABSYEN	NOP
16			LSPZEN	LLDEXP
17			L2-XYEN	LFRCPOS
18			LTEN	LR1STRT
19	LMULTCLR		LTEN	NOP
20			LTEN	NOP
21			LSPZEN	LLDEXP
22			LSPZEN	LFRCPOS
23		LREN	LABSYEN	LR2STRT
24	HMULTCLR		LABSYEN	NOP
25			LABSYEN	LR1STRT
26			LABSYEN	NOP
27			LABSYEN	NOP
28			LSPZEN	LLDEXP
29			L2-XYEN	LFRCPOS
30			LTEN	LR2STRT
31	HMULTCLR		LTEN	NOP
32			LTEN	LR1STRT
33			LTEN	NOP
34			LTEN	NOP
35			LSPZEN	LLDEXP
36		LDPDIV	LQUALCOMPEN	LFRCPOS
37		LREN	LDPNUMEN	LR4STRT
38	LTADDEN HMULTCLR	LTPPA4STRT	LDPNUMEN	LR3STRT
39	LTADDEN	LTPPA3STRT	LDPNUMEN	NOP
40	LTADDEN		LDPNUMEN	LR2STRT
41	LTADDEN	LTPPA2STRT	LDPNUMEN	NOP
42	LTADDEN		LDPNUMEN	LR1STRT
43	LTADDEN	LTPPA1STRT	LDPNUMEN	NOP
44	LTADDEN		LDPNUMEN	NOP
45	LTADDEN		LDPNUMEN	LLDEXP
46	HDONE	LDPDIV	LRNDEN	LLDOUT

logic at the beginning of each fixed-point instruction execution sequence.

For the MPMB, MPMH, MPMW and MPI instructions, the memory operand (or immediate operand for an MPI instruction) is transferred from the CPU to the M/D unit on the EDB bus. The register operand is input from the file register to the M/D unit on the OPR bus.

For the MPR instruction, both register operands are transferred from the FPA file register to the M/D unit on the EDB and OPR buses, respectively.

For all fixed-point instructions, the operand input from the EDB bus is the multiplicand. All 32 bits (00 through 31) of the multiplicand are loaded into the fixed-point multiplicand register (sheet 3). The operand input from the OPR bus is always the multiplier. All 32 bits (00 through 31) of the multiplier are loaded into the dividend/multiplier (D/M) register (sheet 10).

2.2.3.3 Destination File Address

The destination file address is handled in the same manner as previously described for the floating-point multiply instruction (paragraph 2.2.1.4).

2.2.3.4 Sign Bit Manipulation

Sign bit manipulation for fixed-point operation is limited to conditioning of the two's complement logic (sheets 8 and 15). In addition to being loaded into the operand input registers (D/M register and fixed-point multiplicand register) as part of the integer value, the sign bit of each operand is input directly, from the EDB and OPR buses, to the staging register shown on sheet 29. The output of the staging register (HNEGDIVSR and HNEGNUM) is used to generate the X and Y sign bit signals (LENXT and LENYT) that condition the two's complement logic.

The two's complement logic and the function of the sign bit conditioning signals are explained in the text for floating-point multiply (paragraphs 2.2.1.6 and 2.2.1.7.2) and illustrated in Figure 2-7.

2.2.3.5 Operand Multiply

The multiplication of fixed-point operands is performed in basically the same manner as that described for multiplication of floating-point fractions (paragraphs 2.2.1.7 and 2.2.1.7.2).

Operand multiplication begins the cycle after operand loading when the sequence control logic enables the output of the multiplier (X), from the D/M register, and the multiplicand (Y), from the fixed-point multiplicand register, onto the X and Y buses, respectively. The entire 32-bit multiplicand is output on the Y bus in bit positions 08 through 39; the remaining bit positions (40 through 63) of the Y bus are filled with zeros, output from the zero-extension register. One byte of the multiplier is output onto the X bus for each multiply iteration. Four iterations are required to step through the entire 32-bit multiplier, beginning with the least-significant byte (bits 24 through 31).

2.2.3.6 Answer Output

The doubleword integer product is loaded into the fixed-point multiply output register (sheet 7) under control of the sequence control logic (LLDOUT signal, sheet 32). The fixed-point multiply output register receives RC bits 08 through 71 and, at the appropriate time, outputs one 32-bit word at a

time to the file register (A/S unit) by way of the EY bus (00 through 31). The fixed-point answer output is shown in Table 2-19.

Output to the EY bus is controlled by the output control logic (sheet 31). A description of this logic is provided in paragraph 2.2.1.9.

2.2.3.7 Sequence Control

An explanation of the sequence control logic is included in the text for floating-point multiply (paragraph 2.2.1.2).

Tables 2-20 and 2-21 provide the PROM control codes and sequence control signals applicable to all of the fixed-point multiply instructions.

Table 2-19
Fixed-point Answer Output

Fixed-point Multiply Out. Reg. 08-15 (sheet 28) Fixed-point Multiply Out. Reg. 16-39 (sheet 7)	HEY00-07 HEY08-31	Cycle n
Fixed-Point Multiply Out. Reg 40-71 (sheet 7)	HEY00-31	Cycle n+1

Table 2-20
Sequence Control PROM Coding - Fixed-point Multiply

Address	PROM MB84		PROM MB81	
	Field A	Field B	Field C	Field D
	8 7 6 5	4 3 2 1	8 7 6 5	4 3 2 1
080	0 1 1 0	1 0 0 1	0 0 1 0	0 0 0 1
081	0 0 0 1	0 0 0 0	0 0 1 0	0 0 0 0
082	0 0 0 0	1 0 0 0	0 0 1 0	0 0 0 0
083	0 0 0 0	0 0 0 0	0 0 1 0	0 0 1 0
084	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
085	1 0 1 0	1 1 1 0	0 0 0 0	1 1 1 1

Table 2-21
Active Sequence Control Signals - Fixed-point Multiply

Step	PROM MB84		PROM MB81	
	Field A	Field B	Field C	Field D
1	HSTOPCLKEN	LTPPA2STRT	LFXPYEN	LX1STRT
2	HMULTCLR LTADDEN		LFXPYEN	NOP
3	LTADDEN	LTPPA1STRT	LFXPYEN	NOP
4	LTADDEN		LFXPYEN	LX2STRT
5	LTADDEN		LSPZEN	NOP
6	HDONE	LDPDIV	LSPZEN	LLDOUT

APPENDIX A

Table A-1
MB84 PROM Control Code Definitions - M/D Unit

MB84 Field A	MB84 Field B
1 x x x - HDONE	1 0 0 0 - LTPPA1STRT
x 1 x x - HSTOPCLKEN	1 0 0 1 - LTPPA2STRT
x x 0 x - LTADDEN	1 0 1 0 - LTPPA4STRT
x x x 1 - HMULTCLR	1 0 1 1 - LTPPA3STRT
	1 1 0 0 - LREN
	1 1 0 1 - LSETAE
	1 1 1 0 - LDPDIV
	1 1 1 1 - Not Assigned

Table A-2
MB81 PROM Control Code Definitions - M/D Unit

MB81 Field C	MB81 Field D
0 0 0 0 - LSPZEN	0 0 0 0 - NOP
0 0 0 1 - LDPYEN	0 0 0 1 - LX1STRT
0 0 1 0 - LFXPYEN	0 0 1 0 - LX2STRT
0 0 1 1 - LTEN	0 0 1 1 - LR1STRT
0 1 0 0 - LABSYEN	0 1 0 0 - LR2STRT
0 1 0 1 - LDPNUMEN	0 1 0 1 - LR3STRT
0 1 1 0 - Not Assigned	0 1 1 0 - LPROMSTRT
0 1 1 1 - LSUBTEXPEN	0 1 1 1 - L1CNSTEN
1 0 0 0 - Not Assigned	1 0 0 0 - Not Assigned
1 0 0 1 - LRNDEN	1 0 0 1 - LFRCPOS
1 0 1 0 - LQUALCOMPEN	1 0 1 0 - LX3STRT
1 0 1 1 - L2-XYEN	1 0 1 1 - LR4STRT
1 1 0 0 - LSELMULT	1 1 0 0 - L0CNSTEN
1 1 0 1 - Not Assigned	1 1 0 1 - Not Assigned
1 1 1 0 - LDPDIVEN	1 1 1 0 - LLDEXP
1 1 1 1 - NOP	1 1 1 1 - LLDOUT

APPENDIX B

Table B-1
Connector J1 Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
01	GND	18	HEY10	35	GND
02	GND	19	HEY11	36	HEY22
03	HEY00	20	GND	37	HEY23
04	HEY01	21	HEY12	38	GND
05	GND	22	HEY13	39	HEY24
06	HEY02	23	GND	40	HEY25
07	HEY03	24	HEY14	41	GND
08	GND	25	HEY15	42	HEY26
09	HEY04	26	GND	43	HEY27
10	HEY05	27	HEY16	44	GND
11	GND	28	HEY17	45	HEY28
12	HEY06	29	GND	46	HEY29
13	HEY07	30	HEY18	47	GND
14	GND	31	HEY19	48	HEY30
15	HEY08	32	GND	49	HEY31
16	HEY09	33	HEY20	50	GND
17	GND	34	HEY21		

Table B-2
Connector J2 Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
01	GND	18	HOPR10	35	GND
02	GND	19	HOPR11	36	HOPR22
03	HOPR00	20	GND	37	HOPR23
04	HOPR01	21	HOPR12	38	GND
05	GND	22	HOPR13	39	HOPR24
06	HOPR02	23	GND	40	HOPR25
07	HOPR03	24	HOPR14	41	GND
08	GND	25	HOPR15	42	HOPR26
09	HOPR04	26	GND	43	HORP27
10	HOPR05	27	HOPR16	44	GND
11	GND	28	HOPR17	45	HOPR28
12	HOPR06	29	GND	46	HOPR29
13	HOPR07	30	HOPR18	47	GND
14	GND	31	HOPR19	48	HOPR30
15	HOPR08	32	GND	49	HOPR31
16	HOPR09	33	HOPR20	50	GND
17	GND	34	HOPR21		

Table B-3
Connector J3 Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
01	LENBLFPA	18	LFPUBUSY	35	GND
02	HPHX2	19	LAEPEND	36	HCREG38
03	HEA00	20	GND	37	HCREG39
04	HEA01	21	HAEXCPEN	38	GND
05	GND	22	LFPAMDATA	39	LFPUOPRAORD
06	HEA02	23	GND	40	LFPUOPRBORD
07	HEA03	24	LFPREGAUPD	41	GND
08	GND	25	LFPREGBUPD	42	LQUALCREGCLK
09	HEB00	26	GND	43	SPARE
10	HEB01	27	SPARE	44	GND
11	GND	28	SPARE	45	HCPUOUT
12	HEB02	29	GND	46	HSTOPCLK
13	HEB03	30	LFILEEN	47	GND
14	GND	31	LAEEN	48	LSTOPCPU
15	LFPAAEDATA	32	GND	49	LFPUPRESENT
16	SPARE	33	HCREG36	50	GND
17	GND	34	HCREG37		

Table B-4
Connector J4 Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
01	GND	18	HEDB10	35	GND
02	GND	19	HEDB11	36	HEDB22
03	HEDB00	20	GND	37	HEDB23
04	HEDB01	21	HEDB12	38	GND
05	GND	22	HEDB13	39	HEDB24
06	HEDB02	23	GND	40	HEDB25
07	HEDB03	24	HEDB14	41	GND
08	GND	25	HEDB15	42	HEDB26
09	HEDB04	26	GND	43	HEDB27
10	HEDB05	27	HEDB16	44	GND
11	GND	28	HEDB17	45	HEDB28
12	HEDB06	29	GND	46	HEDB29
13	HEDB07	30	HEDB18	47	GND
14	GND	31	HEDB19	48	HEDB30
15	HEDB08	32	GND	49	HEDB31
16	HEDB09	33	HEDB20	50	GND
17	GND	34	HEDB21		

Table B-5
Connector J5 Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
01	GND	18	HRC10	35	GND
02	GND	19	HRC11	36	HRC22
03	HTEXP00	20	GND	37	HRC23
04	HTEXP01	21	HRC12	38	GND
05	GND	22	HRC13	39	HRC24
06	HTEXP02	23	GND	40	HRC25
07	HTEXP03	24	HRC14	41	GND
08	GND	25	HRC15	42	HRC26
09	HTEXP04	26	GND	43	HRC27
10	HTEXP05	27	HRC16	44	GND
11	GND	28	HRC17	45	HRC28
12	HTEXP06	29	GND	46	HRC29
13	HTEXP07	30	HRC18	47	GND
14	GND	31	HRC19	48	HRC30
15	HRC08	32	GND	49	HRC31
16	HRC09	33	HRC20	50	GND
17	GND	34	HRC21		

M/D unit connector J5 is used for test purposes only.

Table B-6
Connector J6 Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
01	GND	18	HRC42	35	GND
02	GND	19	HRC43	36	HRC54
03	HRC32	20	GND	37	HRC55
04	HRC33	21	HRC44	38	GND
05	GND	22	HRC45	39	HRC56
06	HRC34	23	GND	40	HRC57
07	HRC35	24	HRC46	41	GND
08	GND	25	HRC47	42	HRC58
09	HRC36	26	GND	43	HRC59
10	HRC37	27	HRC48	44	GND
11	GND	28	HRC49	45	HRC60
12	HRC38	29	GND	46	HRC61
13	HRC39	30	HRC50	47	GND
14	GND	31	HRC51	48	HRC62
15	HRC40	32	GND	49	HRC63
16	HRC41	33	HRC52	50	GND
17	GND	34	HRC53		

M/D unit connector J6 is used for test purposes only.

Table B-7
Connector P1A Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
01	GND	18	SPARE	35	GND
02	GND	19	SPARE	36	SPARE
03	SPARE	20	GND	37	SPARE
04	SPARE	21	SPARE	38	GND
05	GND	22	SPARE	39	SPARE
06	SPARE	23	GND	40	SPARE
07	SPARE	24	SPARE	41	GND
08	GND	25	SPARE	42	SPARE
09	SPARE	26	GND	43	SPARE
10	SPARE	27	SPARE	44	GND
11	GND	28	SPARE	45	SPARE
12	SPARE	29	GND	46	SPARE
13	SPARE	30	SPARE	47	GND
14	GND	31	SPARE	48	SPARE
15	SPARE	32	GND	49	SPARE
16	SPARE	33	SPARE	50	GND
17	GND	34	SPARE		

Table B-8
Connector P1C Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
01	GND	18	SPARE	35	GND
02	GND	19	SPARE	36	SPARE
03	HFADD00	20	GND	37	SPARE
04	HFADD01	21	SPARE	38	GND
05	GND	22	SPARE	39	SPARE
06	HFADD02	23	GND	40	SPARE
07	HFADD03	24	SPARE	41	GND
08	GND	25	SPARE	42	SPARE
09	HMULTBUSY	26	GND	43	SPARE
10	LUNITSTORE	27	SPARE	44	GND
11	GND	28	SPARE	45	SPARE
12	LAEDATA	29	GND	46	SPARE
13	LADDOUT	30	SPARE	47	GND
14	GND	31	SPARE	48	SPARE
15	HADDBUSY	32	GND	49	SPARE
16	SPARE	33	SPARE	50	GND
17	GND	34	SPARE		

Table B-9
Connector P1B Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
01	GND	78	GND	131	GND
02	GND	79	GND	137	+5V
03	+5V	83	GND	138	+5V
04	+5V	85	LCLKL	139	GND
10	GND	87	GND	140	GND
19	GND	88	LSTSC	146	GND
39	GND	91	GND	155	GND
45	GND	92	GND	166	GND
46	GND	93	GND	175	GND
47	+5V	94	GND	181	+5V
48	+5V	98	GND	182	+5V
54	GND	99	GND	183	GND
63	GND	104	LRESET	184	GND

P1B, the SelBUS connector, has a total of 184 pins. This table lists only those pins used by the floating-point accelerator. For a complete list of the standard P1B pin assignments, see the CONCEPT 32/67 CPU Technical Manual, publication order number 303-000410.

INDEX

Gould Inc., S.E.L. Computer Systems Division
6901 W. Sunrise Blvd., P.O. Box 9148,
Fort Lauderdale, FL 33310-9148
Phone (305) 587-2900

GOULD
Electronics

Users Group Membership Application

USER ORGANIZATION: _____

REPRESENTATIVE(S): _____

_____

_____

ADDRESS: _____

_____

TELEX NUMBER: _____ PHONE NUMBER: _____

NUMBER AND TYPE OF GOULD S.E.L. COMPUTERS: _____

_____

_____

_____

APPLICATIONS (Please Indicate)

1. EDP
   - A. Inventory Control
   - B. Engineering & Production Data Control
   - C. Large Machine Off-Load
   - D. Remote Batch Terminal
   - E. Other

2. Communications
   - A. Telephone System Monitoring
   - B. Front End Processors
   - C. Message Switching
   - D. Other

3. Design & Drafting
   - A. Electrical
   - B. Mechanical
   - C. Architectural
   - D. Cartography
   - E. Image Processing
   - F. Other

4. Industrial Automation
   - A. Continuous Process Control Op.
   - B. Production Scheduling & Control
   - C. Process Planning
   - D. Numerical Control
   - E. Other

5. Laboratory and Computational
   - A. Seismic
   - B. Scientific Calculation
   - C. Experiment Monitoring
   - D. Mathematical Modeling
   - E. Signal Processing
   - F. Other

6. Energy Monitoring & Control
   - A. Power Generation
   - B. Power Distribution
   - C. Environmental Control
   - D. Meter Monitoring
   - E. Other

7. Simulation
   - A. Flight Simulators
   - B. Power Plant Simulators
   - C. Electronic Warfare
   - D. Other

8. Other

Please return to:

Users Group Administrator

Date: _____

Gould S.E.L. Users Group . . .

The purpose of the Gould S.E.L. Users Group is to help create better User/User and User/Gould S.E.L. Communications.

There is no fee to join the Users Group. Simply complete the Membership Application on the reverse side and mail to the Users Group Administrator. You will automatically receive Users Group Newsletters, Referral Guide and other pertinent Users Group Activity information.

Fold and Staple for Mailing

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 947 FT. LAUDERDALE, FLORIDA 33310

POSTAGE WILL BE PAID BY ADDRESSEE

Gould Inc., S.E.L. Computer Systems Division
Attn: Users Group Administrator
6901 W. Sunrise Blvd., P.O. Box 9148
Ft. Lauderdale, FL 33310-9148

(Detach Here)

Fold and Staple for Mailing

GOULD
Electronics