THE HEWLETT-PACKARD F-SERIES FLOATING POINT PROCESSOR

THEORY OF OPERATION

NOTE

This document is part of the HP 1000 M, E, and F-Series Computers Engineering and Reference Documentation and is not available separately.

PREFACE

This document is written to explain how the Floating Point Processor (FPP) operates but not why each design decision was made. Thus, algorithm implementation is described but the algorithm investigation is not. In one sense this document is merely a reference to the schematics.

This document limits itself to the detailed inner workings of the processor. Thus, the power supply, the programming and microprogramming, microprogrammable processor port are not discussed here.

```
+----------------------------------------------------------+
|                                                          |
| CONTENTS                                                 |
|                                                          |
+----------------------------------------------------------+
```

```
+----------------------------------------------------------+
|                                                          |
| TABLES                                                   |
|                                                          |
+----------------------------------------------------------+
```

Title Page

```
+--------------------------------------------------------+
|                                                        |
| ILLUSTRATIONS                                          |
|                                                        |
+--------------------------------------------------------+
```

Title Page

```
+----------------------------------------------------------+-----------------+
|                                                          |                 |
|   GENERAL                                                |  SECTION   I    |
|                                                          |                 |
+----------------------------------------------------------+-----------------+
```

1.0 INTRODUCTION

The HP 1000 F-Series derives its computational power from the Hardware Floating Point Processor (FPP). As the F-Series has the same central processor as the E-Series computers, the floating point processor is the key contribution to the F-Series. This document presents the theory of operation for the floating point processor and describes the processor hardware in great detail. For the most part, the central processor, power supply and programming aspects of the floating point processor are not described, as there already exists ample documentation covering their operation. See the related documents section for the list of supporting documentation.

The reader of this document is assumed to be technically oriented and experienced in digital logic design.

As the intent of this document is to provide understanding of the FPP schematics, extensive reference to the schematics are made. Thus, the schematics should accompany this document in order to facilitate their understanding. Also, familiarity with Schottky TTL (74SXX) and Low-Power TTL (74LSXX) families is required.

2.0 RELATED DOCUMENTS

Schematics: D-12740-60001-51 through 56
D-12740-60002-51 through 55

Algorithms: This document makes extensive references to "The Logic of Computer Arithmetic" by Ivan Flores, Prentice-Hall, Inc., 1963.

Recommended reading from Ivan Flores:

Floating point number representation:	Sections 15.1, 15.2
Floating point addition and subtraction:	Section 15.3
Floating point multiplication:	Sections 10.2, 10.3, 10.5-10.8
Floating point division:	Section 12.1, 12.2, 13.1-13.4

Standard HP 1000 Floating Point Operations - formats and instruction codes:

HP 1000 F-Series Computer Operating and Reference Manual,
part no. 02111-90001

HP 1000 E-and F-Series Computer Microprogramming Reference Manual,
part no. 02109-90004

Microprogrammable Processor Port:

HP 1000 E-Series Microprogrammable Processor Port Application Note,
part no. 5953-0835

HP 1000 M/E/F-Series Computer I/O Interfacing Guide,
part no. 02109-90006

Installation:

HP 1000 F-Series Computer Installation and Service Manual,
part no. 02111-90002

Diagnostic:

HP 1000 F-Series Floating Point Processor Diagnostic Manual,
part no. 12740-90004

3.0 FPP RELATIONSHIP TO 2111F/2117F

The floating point processor is a hardware implementation of existing HP 1000 floating point arithmetic instructions. The processor performs these operations on 32 bit single precision, 48 bit extended precision, or 64 bit double precision operands which are represented in standard HP 1000 floating point number formats. In the 2111F computer the floating point processor boards are mounted inside the chassis over the card cages and power supply. In the 2117F computer, the processor boards are in a separate unit with their own power supply. The floating point processor communicates to the central processor across the microprogrammable processor port, thus allowing fast and direct microprogram control of the floating point processor.

4.0 DESIGN PHILOSOPHY

The goal of the F-Series computer was to achieve high computational performance through a floating point processor (FPP). The processor had to implement existing HP 1000 floating point arithmetic instructions. Thus, the processor had to execute 32 bit single precision and 48 bit extended precision floating point operations. Another design objective was to expand HP's floating point capability by also performing 64 bit double precision operations and conversion routines for 32 bit double integers. As all the registers and data paths were to fit on one 11x16 inch HP Corporate full module standard circuit board, they would have to be implemented in MSI circuits. The execution time objective for the floating point operations was 5 to 10 times faster than firmware operations on the E-Series.

Floating Point Processor

As stated above, one of the primary design goals of the floating point processor was to implement the operations add, subtract, multiply, divide, fix to integer, and float from integer with a minimum register and data path configuration. This configuration had to fit on a single 11 by 16 inch printed circuit board. Floating point addition and subtraction algorithms involve shifting mantissas right to equalize exponents, adding and subtracting mantissas and then shifting the result left to normalize it. Thus, the minimum hardware configuration had to include bidirectional shift registers and arithmetic logic units (ALUs). In light of this configuration, the algorithms chosen for multiplication and division were ones that consist of sequences of shift cycles and arithmetic cycles.

In addition to the register and data path board, hence called arithmetic board, there is the control board which controls the operations of the arithmetic board. Since the floating point operations consist of sequences of shift and arithmetic (ALU) cycles, a state machine is used to direct the sequences. In order to make the ALU cycles and shift cycles as short as the hardware circuits permit, the state machine consists of high speed shift registers clocked at 40 MHz. At 40 MHz a particular state is active for only 25 nanoseconds (ns).

Thus, the floating point processor consists of control and arithmetic printed circuit boards. Also, a printed circuit backplane provides communication between the two boards.

5.0 REFERENCE INFORMATION

The HP 1000 F-Series Floating Point Processor consists of two 11 by 16 inch six layer and one 4 1/2 by 11 inch four layer printed circuit assemblies (PCAs). Schematics illustrate the electronics design and assembly drawings illustrate component loading for the PCAs. The backplane drawings listed below are located in Section VII.

BOARD	SCHEMATICS	ASSEMBLY DRAWING
Arithmetic	D-12740-60001-51 through 56	F-12740-60001-1
Control	D-12740-60002-51 through 55	F-12740-60002-1
Backplane	C-12740-60004-51	D-12740-60004-1

5.1 Binary Signal Levels

Most logic used in the computer is implemented with Schottky or low-power Schottky TTL components, and positive logic is employed. High levels are +2.5 to +3.5V normally. Low levels are 0.0 to 0.8V. Some circuitry may depart from these values in special circumstances. The actual values to be expected may be determined from the type, load, and condition of the component. Logically, "1" is "high" or "true" and "0" is "low" or "false".

5.2 Logic Circuits

Logic circuits in the theory and on the schematic are drawn to aid in the understanding of the logical function. "Bubbles" on inputs or outputs indicate active low logic levels.

5.3 Signal Names

Signal names are alphanumeric identifiers selected to aid in understanding of the signal function. Signal names are followed with a "+" if they are active high and a "-" if they are active low. If there are no "+" or "-" following the name, then the signal is active high. Busses are named by a sequence of letters followed by a number indicating bit significance.

Many signal names have suffixes that help distinguish the origin or purpose of the signal. For instance:

B	- buffered
EN	- enable, as in tri-state ooutput enable
FF	- flip-flop
CK	- clock
RS	- reset
CTL	- one of the control states
OP	- operation

5.4 Cross-References

Each page of the schematic is broken up into 24 areas whose boundaries are marked as grid locations. For instance the horizontal axis is marked 1 through 6 and the vertical axis is marked A through D. Signals which run from sheet to sheet have the coordinates of their source listed next to their name. Also, as this document discusses specific circuits, it will indicate the coordinates of the circuit area in brackets; for example, PWRST- (U183-3 at 13B).

5.5 Abbreviations and Mnemonics

The following abbreviations and mnemonics occur frequently in the text. Their meanings are given below:

AEXP	- exponent register of operand A
ALU	- arithmetic logic unit. Also, ALU cycle indicates a cycle where operands must pass through the arithmetic logic units.
AMAN	- mantissa register of operand A
BEXP	- exponent register od operand B
BMAN	- mantissa register of operand B
CMAN	- mantissa register register of operand C
CPU	- central processor unit
FPP	- floating point processor

IR	- instruction register of the floating point processor
LSB	- least significant bit of mantissa
MPP	- microprogrammable processor port
SWAMP-	condition where operand exponents are too far apart to do meaningful arithmetic.

5.6 CPU-FPP Block Diagram (Figure I-1)

The F-Series computer uses the same control processor as the E-Series computer with the floating point processor connected directly to the main data bus via the microprogrammable processor port. The floating point processor contains data and control logic dedicated to floating point operations, so that the control processor no longer needs to perform these calculations in firmware or software routines.

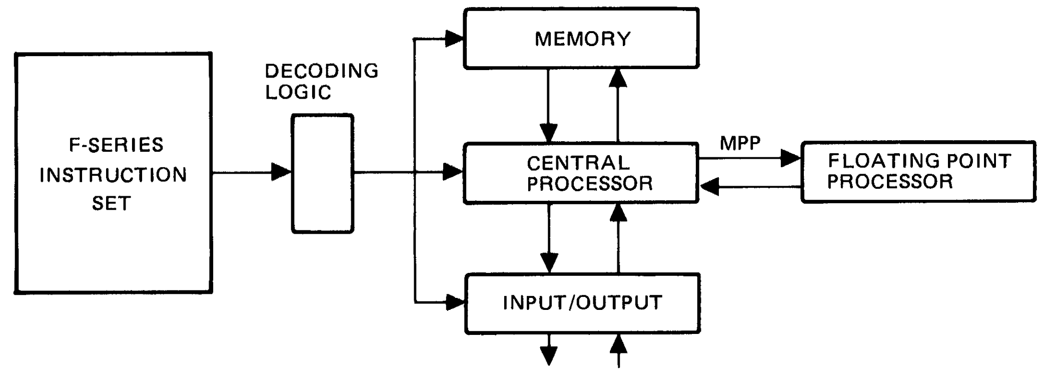


Figure I-1. CPU-FPP Simplified Block Diagram

6.0 FUNCTIONAL CHARACTERISTICS

6.1 Floating Point Number Formats (Figure I-2)

All floating point operands consist of a signed mantissa and signed exponent represented in two's complement HP 1000 series computer format as shown in Figure I-2. Single, extended, and double precision mantissas are sign plus 23, 39 or 55 bits long respectively. The exponents of these operands are always seven bits plus the exponent sign. Thus, exponents range from 2 exp (127) to 2 exp (-128). The first word of an operand holds the most significant mantissa bits. The operand's last word consists of the 8 least significant mantissa bits and the 8-bit exponent. Note that the exponent magnitude is in bits 7 through 1 and the exponent sign bit is rotated to bit 0 of the last word.

If an overflow or underflow result occurs during an operation, the FPP sets the CPU overflow bit. Overflow occurs if the result of a floating point operation lies outside the range of -2exp(127) through (1 - 2exp(-n)) * 2exp(127), where "n" equals two less than the number of mantissa bits in an operand format. Underflow occurs if the result lies within the range -(1+2exp(-n))*2exp(-129). The overflow result returned is the maximum positive floating point number, 2exp(128) minus 2exp(-n), and the underflow result returned is all zeros.

The single integer generated by fix and used by float is the standard HP 1000 16-bit two's complement integer, and the double integer is the 32 bit two's complement integer.

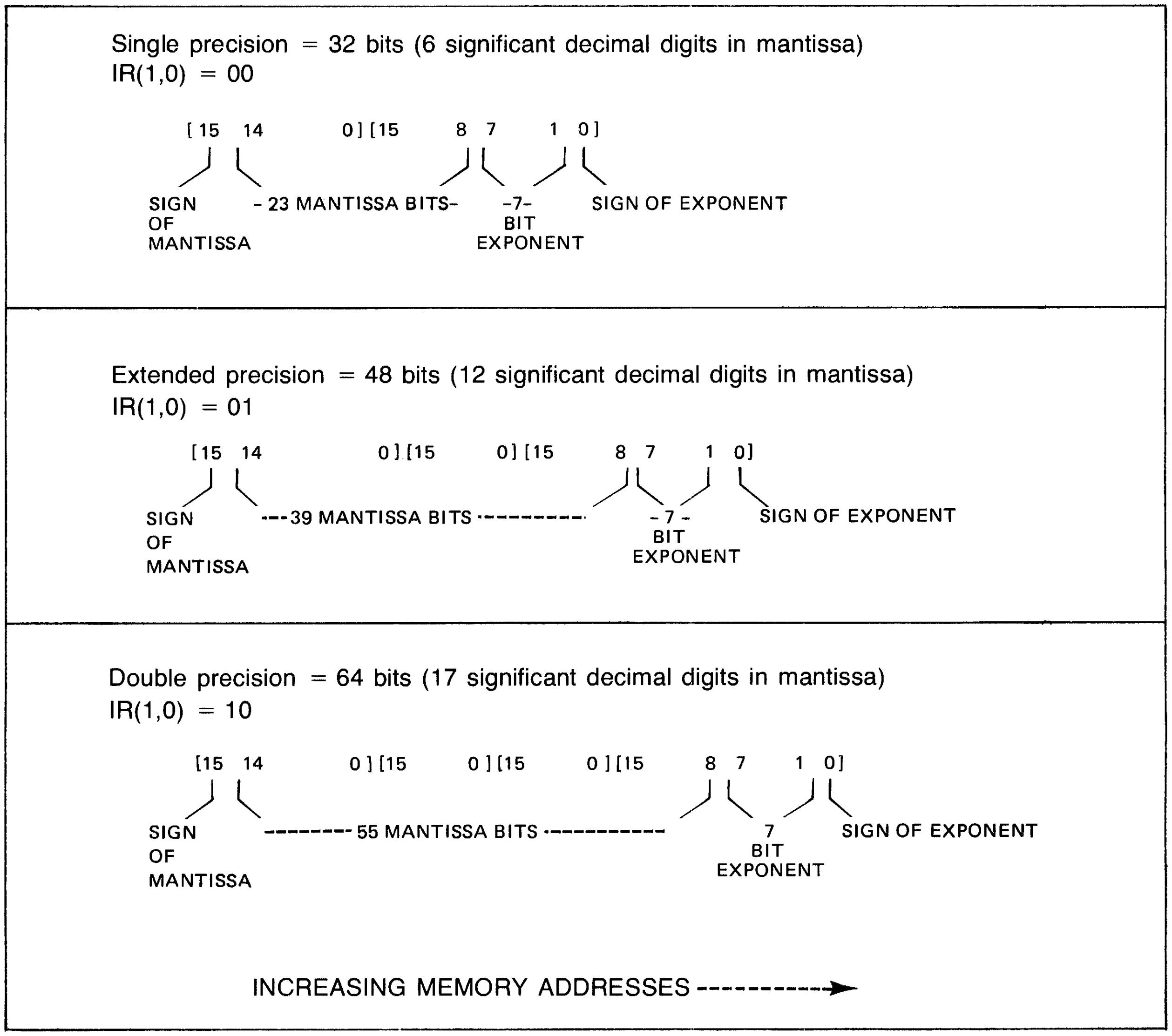

Single precision = 32 bits (6 significant decimal digits in mantissa)
IR(1,0) = 00
[15 14 0][15 8 7 1 0]
SIGN OF MANTISSA
- 23 MANTISSA BITS-
-7- BIT EXPONENT
SIGN OF EXPONENT
Extended precision = 48 bits (12 significant decimal digits in mantissa)
IR(1,0) = 01
[15 14 0][15 0][15 8 7 1 0]
SIGN OF MANTISSA
---39 MANTISSA BITS----------
-7- BIT EXPONENT
SIGN OF EXPONENT
Double precision = 64 bits (17 significant decimal digits in mantissa)
IR(1,0) = 10
[15 14 0][15 0][15 0][15 8 7 1 0]
SIGN OF MANTISSA
--------55 MANTISSA BITS-----------
7 BIT EXPONENT
SIGN OF EXPONENT
INCREASING MEMORY ADDRESSES

Figure I-2. Floating Point Number Formats

6.2 Instruction Codes

The hardware Floating Point Processor (FPP) executes the standard HP 1000 computer 32 bit floating point operations and the Fast Fortran Processor executes 48 bit extended precision floating point operations, and offers as well, 64 bit double precision floating point operations. Additionally, the FPP instruction set includes fix to double integer and float from double integer instructions which are available in all three floating point formats. The following lists the instruction codes for the standard floating point operations and calling sequences.

OPERATION	INSTRUCTION		
	2 word	3 word	4 word
addition	105000	105001&	105002*
subtraction	105020	105021&	105022*
multiplication	105040	105041&	105042*
division	105060	105061&	105062*
fix to single integer	105100	105101*	105102*
fix to double integer	105104*	105105*	105106*
float from single integer	105120	105121*	105122*
float from double integer	105124*	105125*	105126*

* indicates new instruction code
& indicates instruction code different from Fast FORTRAN Processor's code

CALLING SEQUENCES:

```
2 word operation     3 word operation     4 word operation

OCT 105nn0           OCT 105nn1           OCT 105nn2
DEF OPND2            DEF RSULT            DEF RSULT
                     DEF OPND1            DEF OPND1
                     DEF OPND2            DEF OPND2
```

The microcode for the standard 32 bit, 48 bit and 64 bit hardware floating point instructions resides in module 3 of the F-Series control store, which on M and E-Series contains the microcode for the firmware 32 bit floating point instructions. If the FPP should fail, the module 3 microcode returns the error result to the result location and sets the CPU overflow flip-flop. The error result has 177777B, which is an invalid floating point number, in the first word of the result.

While the above operations may be executed from software, the microprogrammer is able to use many more features of the FPP. The microprogrammer controls the operations of the FPP through an eight bit instruction which must be loaded into the instruction register (IR) of the FPP. The Microprogramming Reference Manual describes how to initiate operations, send operands to, or retrieve results from the FPP. The following list describes what operations are possible through FPP's eight bit instruction. The instruction codes consist of three fields which dictate the operation, the operand source, and the operand length. If instruction register bit 7 is set, 8-bit exponent overflow detection is inhibited. The use of bit 7 is discussed in paragraph 6.4.

FPP INSTRUCTION REGISTER FORMAT:

IR bits 7	6,5,4	3,2	1,0
overflow	operation	operand source	operand length

Operation field opcodes:	IR bits 654
add	000
subtract	001
multiply	010
divide	011
fix*	100
float*	101
diagnostic operation	110
diagnostic operation	111

The 5 word operand is used with the expanded exponent which is discussed in paragraph 6.5.

	IR bits 65432
*fix to single integer	100a0
fix to double integer	100a1
float from single integer	100a0
float from double integer	101a1

Where a=1 accumulator or a=0 fetched operand

Operand source field opcodes:	IR bits 32
fetched operand * fetched operand	00
fetched operand * accumulator	01
accumulator * fetched operand	10
accumulator * accumulator	11

Here "*" indicates an operation requiring two operands. In the case of fix or float which use only one operand, IR bit 3 specifies whether or not the operand is from the accumulator.

Operand length field opcodes:	IR bits 10
2 word operation	00
3 word operation	01
4 word operation	10
5 word operation	11

Most operations require sending two operands to the FPP. The order that these operands are sent to the FPP is significant, and is shown in the following table.

Operation vs. operand sequence

	first operand IR 3	second operand IR 2
addition	augend	addend
subtraction	minuend	subtrahend
multiplication	multiplier	multiplicand
division	dividend	divisor

6.3 Features

The floating point processor is designed to be functionally equivalent to the HP 1000 software and firmware floating point subroutines. Thus, the only effects to existing programs are shorter execution times. However, the FPP has several features which allow the user to further decrease execution time of floating point calculations. These features include the simple microprogrammed interface, the flexibility of mixing 32, 48 or 64 bit operations in consecutive instructions, and the accumulator function of the processor.

The FPP is interfaced to the HP 1000 F-Series central processor through the microprogrammed processor port. A description of the interface is provided in the I/O Interfacing Guide which a user may refer to in order to generate his own interface. Since only eight signals are required to control the FPP, its interface is easily understood. Also refer to the HP 1000 E-Series and F-Series Computer Microprogramming Reference Manual for a description of the MPP control signals.

The processor's internal registers may function as an accumulator register in order to store intermediate results, which reduces memory overhead time during successive floating point operations. Since the result of any floating point operation is held in the FPP, it may be used as an operand in a subsequent operation. Thus, in chained calculations, there is no need to store the result in memory and then immediately fetch it for the next operation. Instead, the user may use the accumulator to function as either, or both, operands in a floating point operation. For example, the result of the previous floating point operation may serve as the divisor or as the dividend in a subsequent divide operation. The accumulator operations are controlled by bits two and three of the instruction register, IR(2,3), they are easily directed from microcode. Since the accumulator functions depend heavily on the user's application, they will not be furnished with the FPP microcode, but must be written by the user.

All operands in the floating point processor are held in 64 bit registers. For instance, a 32 bit operand is loaded with trailing zeros into the 64 bit wide register. This feature enables the user to switch between 32, 48 and 64 bit operations with little loss in data significance. Round-off errors in successive 32 bit operations can be reduced by performing 64 bit operations. The final result may be retrieved as a 32 bit operand. However, since accumulator operations depend on control information stored during the previous operation, the user must not switch operand length before an accumulator operation.

The floating point processor does not freeze the CPU, so that the user may initiate a floating point operation, concurrently perform other tasks in microcode, and later retrieve the result from the FPP. For instance, in order to reduce memory overhead between successive operations, the user can store operands in the CPU scratch pads. Then he can manipulate memory addresses, start an FPP accumulator operation, perform data reduction, retrieve the floating point result in scratch pads, start the next FPP operation, and store the previous result in memory. In these ways, it is possible to reduce the impact of floating point operations on overall program execution time.

6.4 FPP Control Signals

PIRST Instruction Register Store

PIRST clocks the lower eight bits of the current CPU S-bus into the instruction register on the FPP, if the FPP is not busy executing an operation (MPPCND low). If the FPP is busy, it ignores PIRST.

PP5 Phase 5 of the CPU Cycle

FPP uses PP5 to synchronize data transfers between the FPP and the CPU. PP2SP, MPBST, MPBEN, and PP1SP are examined at PP5. MPPCND and NSTOV are activated at the leading edge of PP5.

PP2SP Initiate an Operation

When PP2SP is received, the FPP prepares to execute the operation dictated by the current contents of the FPP instruction register. The FPP ignores all PIRST signals until it has completed the current operation, and asserts MPPCND (low) to indicate that it is not ready to accept the next instruction.

MPBST Store the Operand Word

At the signal MPBST, the FPP stores the MPPIO data bus in its input register. Operands are transferred to the FPP with successive MPBSTs. The most significant 16-bits of the mantissa or integer is sent over the bus first, followed by the remainder of the mantissa, and finally the exponent. Once an entire operand is compiled in the input register, the FPP transfers the operand to one of its internal registers.

MPBEN Retrieve the Operand Word

MPBEN indicates to the FPP that the CPU is ready for the next word of the result. Since an operand is transferred to the FPP in 16-bit words, MPBST or MPBEN must be asserted several times in order to transfer an entire operand. The most significant 16 bits of the mantissa or integer are transferred first, followed by the rest of the mantissa or integer. The last word of a floating point result has the least significant 8 mantissa bits and the exponent.

MPPCND Busy Flag

MPPCND is asserted (low) by PP2SP until the operation is completed. Once the operation has completed, MPPCND is deasserted (high) indicating that the user may retrieve the results or initiate a new operation.

NSTOV Set the CPU Overflow Flip-Flop

If an operation causes an exponential overflow, or underflow the CPU overflow flip-flop is set at the trailing edge of the first MPBEN received by the FPP.

PP1SP FPP Reset and Qualifier

PP1SP resets the FPP and clocks the current FPP internal bus contents into the FPP output register. Only the diagnostic uses this operation.

PP1SP must be asserted whenever MPBST or MPBEN are asserted to guarantee proper operation during memory refresh. During memory refresh the CPU deasserts MPBEN and MPBST but not PP1SP. The FPP will not transfer data unless PP1SP is received.

PLR0 CPU Latch Register 0

This control signal addresses the FPP. When the FPP is not addressed, it ignores all MPP signals, except for PIRST, and its output signals are disabled from driving the port. A jumper on the 12740 control board controls the FPP address, which is either PLR0 high or PLR0 low. However, for the base set microcode to control the standard floating point operations the jumper must be installed so that the FPP is addressed when PLR0 is high (CPU latch register 0=0).

FPP Control Signal--Microcode Opcode Relationship

The signals described above are activated or tested within the following HP 1000 F-Series microprogram opcodes:

	SPECIAL	JMP CNDX	STORE	S-BUS	
PLR0				L	Word Types 1 or 2
PIRST				IRCM	Word Types 1 or 2
PP2SP	MPP2				Word Types 1 or 2
MPBST			MPPB		Word Types 1 or 2
MPBEN				MPPB	Word Types 1 or 2
MPPCND		MPP			Word type 3
NSTOV		OVFL			Word Type 3
PP1SP	MPP1				Word Types 1 or 2

6.5 Special Operations

6.5.1 Instruction Register Bit 7

The FPP exponent logic circuits can hold up to a 10-bit exponent. The exponent logic will detect overflow from an 8-bit exponent (normal operation) or 10-bit exponent (special operation). Instruction register bit 7 indicates whether an 8-bit or 10-bit exponent is being manipulated. When IR (7) is 1, the FPP sets the CPU overflow flip-flop and returns the overflow result, only if the exponent overflows or underflows a 10 bit value. If the exponent exceeds an 8 bit value but not a 10 bit value, the FPP does not go through the overflow or underflow sequence.

6.5.2 Expanded Exponent and 5 Word Format

The FPP maintains a 10 bit exponent in its exponent register, so that calculations may temporarily exceed 8 bits without losing accuracy. The 5 word operand format allows the user to retrieve and restore an intermediate result whose exponent exceeds 8 bits. For example, in a square root of the sum of the squares routine, it does not matter if the sum overflows an 8 bit exponent, so long as the square root result is in range. The routine, with IR (7) set, saves the operand in a temporary location using the 5 word format. After the routine squares the second operand [IR (7) set], it returns the first square to the FPP and forms the sum of the squares whose exponent may exceed 8 bits. The routine proceeds to take the square root of the sum, leaving IR (7) set, until the last operation when the resultant exponent is checked for 8 bit overflow or underflow. Thus, the end result is fully accurate, even though the sum of the squares might have exceeded the standard range of HP 1000 floating point numbers.

7.0 ALGORITHMS

The addition and subtraction algorithms are the standard equalize exponents - add or subtract mantissas - normalize mantissa procedures. The hardware detects exponents which are too far apart to be equalized for meaningful addition or subtraction, and transfers the larger operand to the output, or result, register. In the case of exponent overflow or underflow during normalization, the FPP returns the maximum number, or zero, respectively, and sets the CPU overflow bit. Section II.2 discusses these algorithms in detail.

The algorithm used in multiplication is the method of shifting over strings of zeros or ones, and detecting isolated ones or zeros as special cases. Thus, the algorithm guarantees that for n digit mantissas, at most n/2 and typically fewer than n/2 additions or subtractions are performed. In the case of 48 bit operands, the partial product is truncated once if exceeds 64 bits. However, 32 bit operands produce the full 64 bit product which may be used in subsequent 64 bit operations. Section II.5 discusses the multiplication algorithm.

Division is implemented by a non-restoring algorithm which shifts over strings of zeros and ones. N digit mantissas require at most n additions and subtractions, and typically, less than n/2. Since all operands are normalized, the divisor requires no preprocessing, and since no remainders are generated, no remainder correction is necessary. If division by zero is attempted, the CPU overflow bit is set, and the maximum number is returned as the quotient. Section II.6 provides further discussion of the division algorithm.

8.0 BASIC HARDWARE CONFIGURATION

The FPP hardware is divided between two printed circuit boards; one board performs the arithmetic functions, while the second provides all the control signals to the arithmetic board. The arithmetic board holds the operands, ALUs and shift registers, as well as provides status information to the control board. Correspondingly, the control board contains the logic which directs the sequence of functions that the operations undergo in completing an instruction. The two boards communicate across two fifty pin connectors and a printed circuit board backplane. All of the status and control interboard signals, as well as additional status signals at a special diagnostic port connector, may be monitored externally, thus enhancing trouble shooting capability.

8.1 Arithmetic Board

The basic arithmetic structure, as shown in Figure I-3, consists of two sections, one to handle exponent calculations, and the other to manipulate the mantissas. There are three main registers in the mantissa section (AMAN, BMAN, and CMAN), and two registers in the exponent area (AEXP and BEXP). These registers hold various operands depending upon the function performed.

AEXP and BEXP are counter/registers used for exponent calculations. They are incremented or decremented during multiplication and division. A comparator connected between AEXP and BEXP gives exponent range information which is used in the addition, subtract and fix instructions. A second comparator is used in overflow detection.

AMAN, BMAN and CMAN registers hold the operand mantissas. AMAN contains the augend in addition, the minuend in subtraction, the product in multiplication, and the dividend and partial remainders in division. BMAN holds the addend in addition, the subtrahend in subtraction, the multiplicand in multiplication and the divisor in division. CMAN, which is not used in addition or subtraction, holds the multiplier in multiplication and the quotient in division.

All three registers are implemented in shift registers, so that the mantissa ALU is used only for arithmetic operations. During multiplication, multiplexers at the output of the ALUs effectively shift the partial product twice to the right. Therefore, on each pass through the ALUs, 100 nanoseconds of shifting in the registers is eliminated, which speeds multiplication.

The arithmetic board requires two data busses, the processor port and the P-bus. The processor port bus transmits data to and from the CPU. The P-bus routes operands from the input registers to the FPP A, B, and C registers, and to the output registers. The P-bus is the main processor bus, and it provides all communications within the arithmetic section. It is a 64 bit tri-state bus, whose high output impedance in the off state is used to generate the all ones condition of the overflow result.

The processor port bus is a 16-bit bidirectional data bus which reflects the CPU S-bus. The S-bus usually drives the port bus through latches which are clocked by the CPU phase 5 timing signal. The port bus drives the S-bus when the opcode MPPB is specified in the S-bus field of a microprogram instruction. The computer front panel holds the latches and drivers that interface the S-bus and the port, bus. The port bus is tied to eight FPP separately enabled groups of 16 registers which form the input and output registers. A 32, 48 or 64 bit operand is built in the input register from 16 bit data words received over the processor port bus. The exponent portion of the operand must reside in the last data word sent over the bus. Similarly, at the end of an operation, the result in the output register is broken down to 16 bit words, and the exponent is packed in the trailing word before its transfer to the CPU.

8.2 Control Board

Control of the floating point operation is implemented through a 60 state sequential machine. This state machine consists of a chain of 60 serially connected and clocked flip-flops. One active pulse is passed along the register chain activating various procedures and functions. Conceptually, the execution of an instruction passes through up to four control phases where each phase is comprised of several procedures and sub-operations. The four execution phases are the receive operand sequence, the group one execution sequence, the group two execution sequence and the termination sequence. The flow through the phases varies for each instruction, which the diagram of Figure I-4 summarizes.

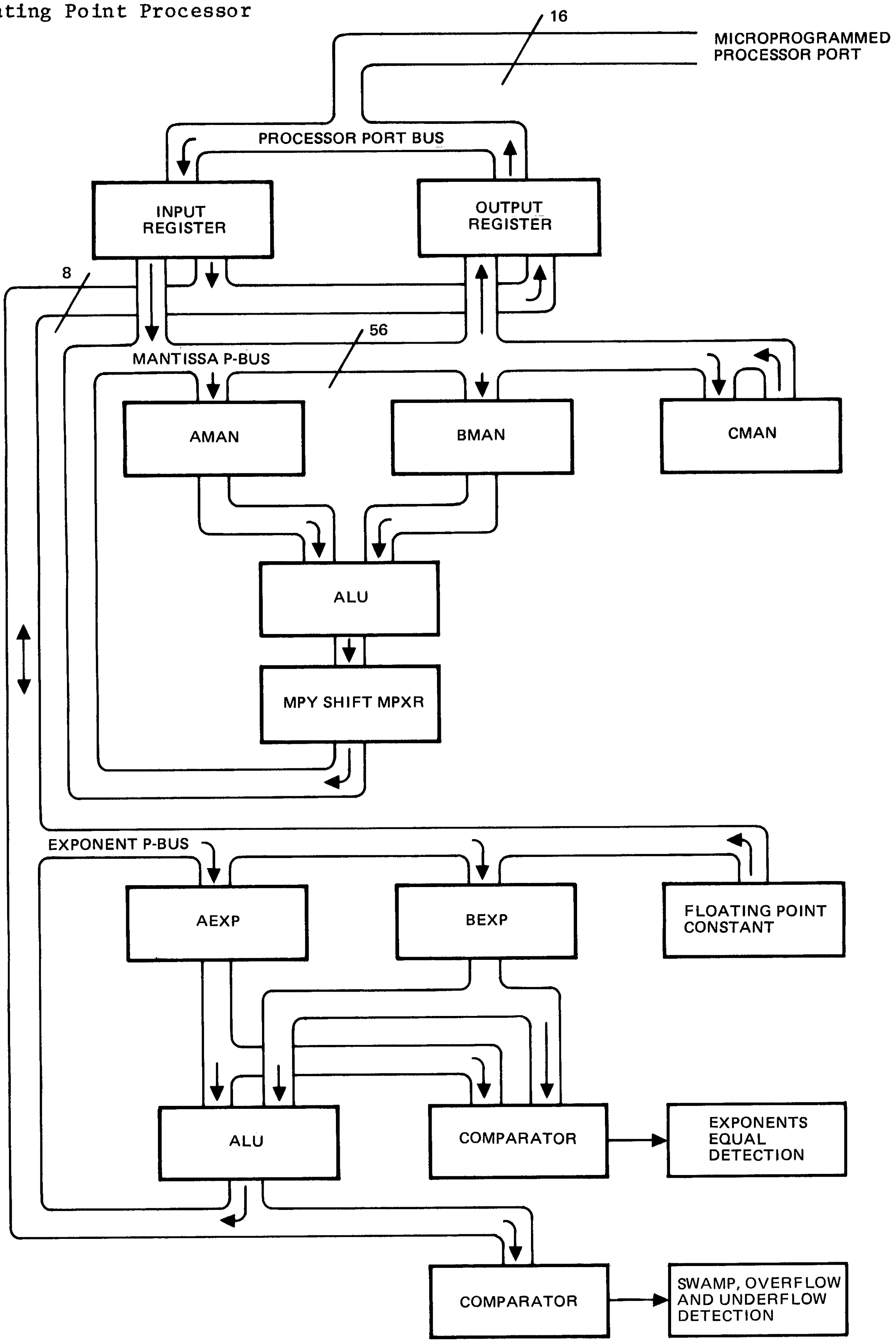


Figure I-3. Arithmetic Section of the Floating Point Processor

(HP 1000 M/E/F-SERIES ERD)

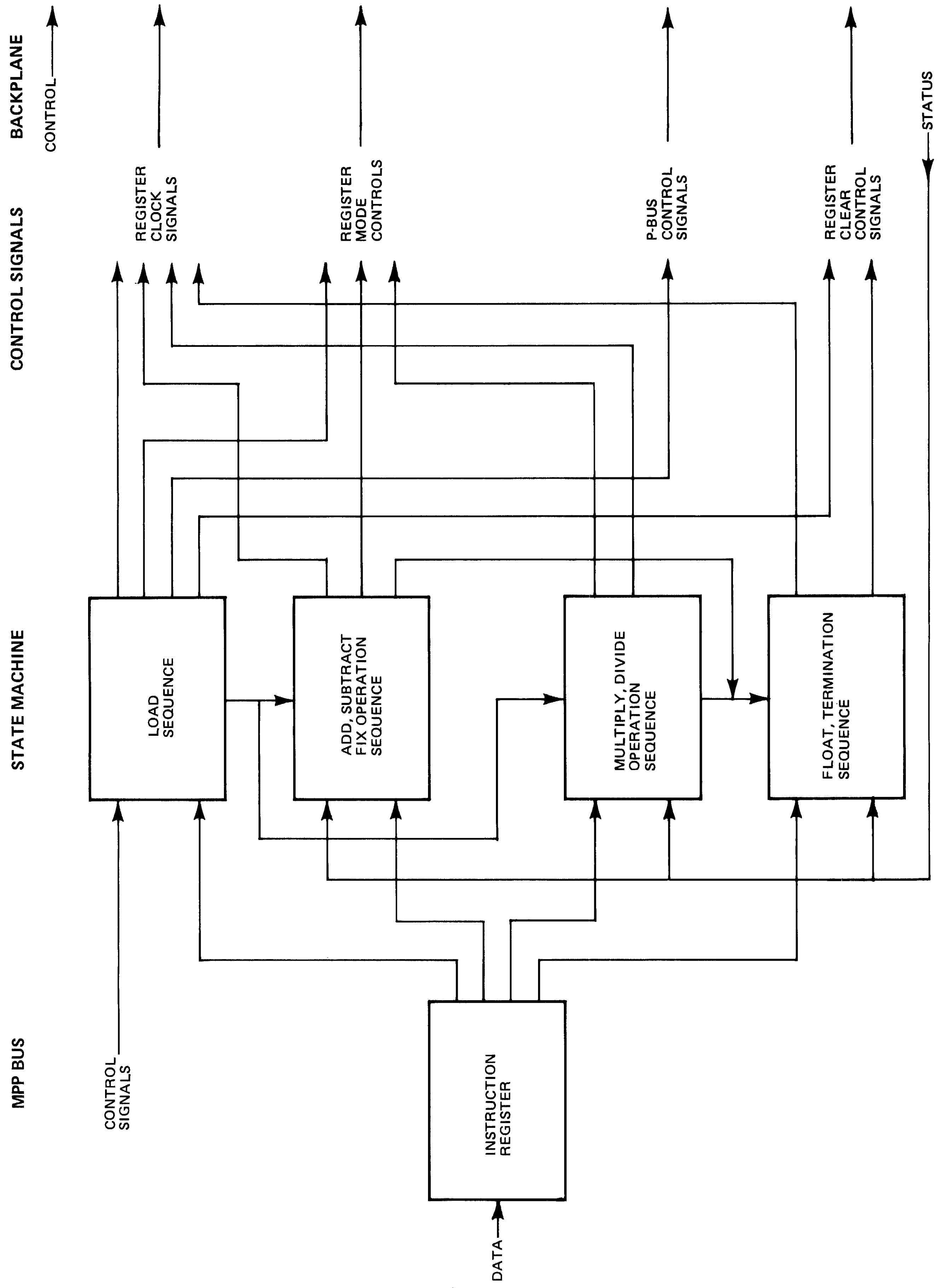


Figure I-4. Overall State Machine

```
+--------------------------------------------------------+----------------+
|                                                        |                |
|   DETAILED DISCUSSION OF THE FPP ALGORITHMS            |  SECTION  II   |
|                                                        |                |
+--------------------------------------------------------+----------------+
```

1.0 FLOATING POINT NUMBER REPRESENTATION

Before discussing floating point arithmetic, there are some aspects of floating point numbers which should be pointed out.

First of all, a floating point number consists of a mantissa or fraction multiplied by a power of two or exponent. Positive numbers have a zero in the sign bit whereas negative numbers have a 1 in the sign bit. Negative numbers are represented in two's complement form. The range of positive mantissas (m) is 1/2 less than or equal to 1 and the range of negative mantissas is -1 less than or equal to -1/2. In binary form for a six bit mantissa the range is 1.000000 < m < 0.111111 and 1.000000 < m < 1.011111. Mantissa are in normalized form when they are in these ranges. Note that the sign bit is opposite in value from the bit just to the right of the binary point. Mantissas are maintained in normalized form in order to provide as much precision or fractional information as possible.

The exponents of the FPP is either seven or nine bits plus sign. The HP 1000 standard format is seven bits plus sign. However, if the user needs more than seven bits, then a nine bit exponent plus sign may be utilized in the FPP. The standard 7-bit overflow detection circuitry must be inhibited by setting IR (7) of the FPP, also the expanded 5-word operand mode must be used to transfer the expanded 9-bit exponent in or out of the FPP. Note that the overflow detection circuits always detect expanded exponent overflows, irregardless of IR (7).

2.0 ADDITION AND SUBTRACTION (Figure II-1)

2.1 Addition/Subtraction Simple Case 1. Mantissa Overflow

Consider the simple case in addition or subtraction where the exponents of the operands are equal. Add the mantissas and check if any corrections need be made to the result.

```
For example:  3/4   in binary form:   0.11000
            + 1/2                   + 0.10000
            ------                  ----------
              5/4                     1.01000
```

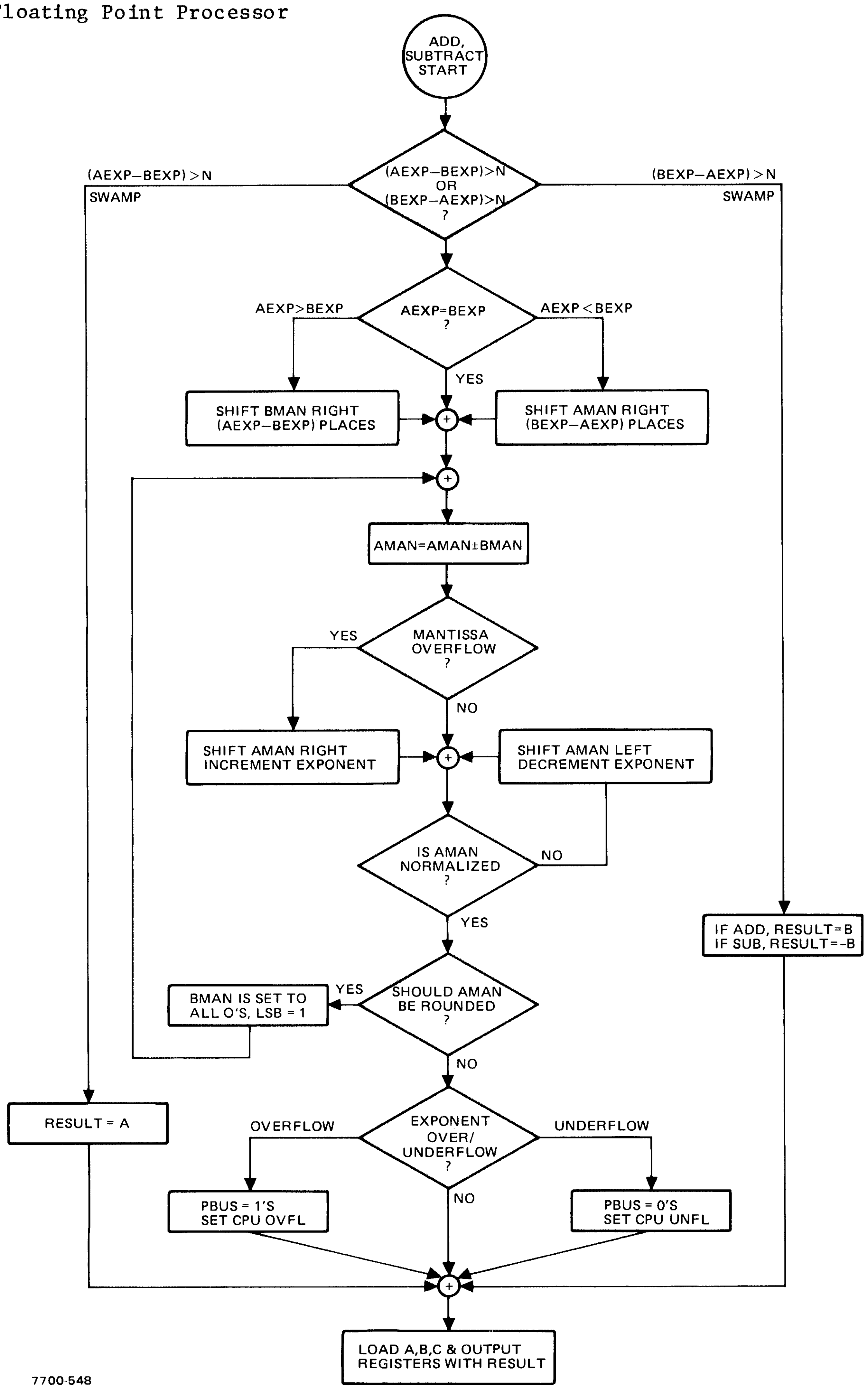


Figure II-1. Addition and Subtraction Flowchart
(HP 1000 M/E/F-SERIES ERD)

Here is a case of mantissa overflow; the mantissa exceeds the acceptable range. The mantissa is corrected by shifting it to the right one place while shifting in the proper sign. Thus, 1.01000 becomes 0.10100. Note that shifting the mantissa to the right one place has the effect of halving the mantissa and, conversely, shifting the mantissa to the left has the effect of doubling the mantissa. In order to preserve the value of the floating point number, shifts to the mantissa must be accompanied by changes to the exponent. In order to correct the overflowed mantissa, the mantissa is shifted to the right or halved and the the the corresponding exponent is incremented.

To correct mantissa overflow   m >=1 or m < -1

1. shift mantissa right shifting in proper sign
2. increment exponent

2.2 Addition/Subtraction Simple Case 2. Normalization or Mantissa underflow

Consider the case of addition -1 plus 1/2.

```
   -1          in binary form:      1.0000
 + 1/2                          +  0.10000
 -----                          ----------
  -1/2                             1.10000
```

Here the result, -1/2, is not in the range of proper mantissas. Since, the sign bit has the same sense as the bit to the right of the decimal point. It is an underflowed mantissa. The result is corrected or normalized by shifting it left one place which has the effect of multiplying it by 2. In order to preserve the value of the floating point number, the exponent must be decremented. Sometimes it is necessary to shift the mantissa several places to bring it to normalized form. For each left shift the exponent should be decremented.

To correct an unnormalized mantissa   -1/2 <= m < 1/2

1. shift mantissa left until it is normalized
2. decrement exponent for each shift

2.3 Exponent Equalization

The previous cases assume that the exponents of the floating point numbers are equal. The exponents can be made equal by incrementing the smaller exponent while shifting its mantissa left. Since the mantissa is halved at each increment to the exponent, the value of the floating point number is maintained.

Example:

Add 8 to -3/8. 8 = 1/2 x 2exp(4); -3/8 = -3/4 x 2exp(-1)

Equalize the exponents by shifting its mantissa and incrementing the exponent five times

```
             m     exp              m    exp
        8 = 1/2  | 4        -3/8 = -3/4  | -1
         0.10000|0100            1.0100000|1111
                                 1.1010000|0000 shift 1
                                 1.1101000|0001 shift 2
                                 1.1110100|0010 shift 3
                                 1.1111010|0011 shift 4
                                 1.1111101|0100 shift 5

      1/2    | 4                 0.1000000|0100
     -3/128  | 4               + 1.1111101|0100
     -----------             -----------------------
     61/128  | 4                 0.0111101|0100
```

The result is not in normalized form. Thus, shift the mantissa left and decrement the exponent.

Result = 0.1111010 | 0011 = 61/64 x 2exp(3) = 61/8 = 7 5/8

2.4 Swamp

As mentioned above, before mantissas can be added or subtracted, exponents of the operands must be equal. Exponents are equalized by shifting the smaller mantissa right while its exponent is incremented until the exponents are equal. However, there is a limit to the number of places an operand should be shifted to equalize exponents. This limit corresponds to the length of the mantissa. Once the smaller operand is shifted off the register it does not effect the larger operand during the operation. In the single precision case, once the 24 bit mantissa is shifted right 24 places, it contributes no significance to the result. Thus, before equalizing the exponents, their difference is checked against the length of the mantissa register. If the exponent difference exceeds the number of bits in the mantissa, then the swamp condition exists. Essentially the operands are so far apart that their sum equals the larger operand. Therefore, in swamp cases the result is merely the larger operand. Note that in subtraction if the larger operand is the subtrahend, the result is the two's complement of the original subtrahend, since it essentially is subtracted from zero.

2.5 Rounding

The arithmetic process may generate a mantissa which is longer (has more bits) than the original mantissa which was input as an operand. For instance, in the exponent equalization process a mantissa is shifted to the right beyond the other mantissa. The resulting mantissa is a combination of the shifted mantissa and the other mantissa. Then this mantissa must be truncated or rounded to the standard floating point format, which consists of 24, 40, or 56 bits depending on the precision. The round or truncate decision is based on the bits to the right of the least significant bit (LSB). For instance, positive mantissas are rounded if the bit just to the right of the LSB, which is called a guard bit, is a one. On the other hand, in order to maintain symmetry about zero, negative mantissas are rounded only if the guard bit is a one and there is at least another one to the right of the guard bit. The rounding process adds a one to the LSB of the mantissa.

For example, consider the four bit mantissa case where 0.10110000 is rounded to 0.110 (3/4), but its complement 1.0101000 is truncated to 1.010 (-3/4); 0.10101111 is truncated to 0.101 (5/8), but its complement 1.01010001 is rounded to 1.011 (-5/8).

Thus, after the resultant sum or difference is adjusted for mantissa overflow or underflow, it may be rounded, which means it must pass through the ALUs a second time. Consider another rounding case of our four bit mantissa: rounding 0.111111 to 1.000 causes mantissa overflow. If mantissa overflow occurs during rounding the result must be corrected. The mantissa is shifted to the right and its exponent is incremented forming a proper result.

2.6 Exponent Overflow/Underflow

After all the exponent adjustments are completed, the resultant exponent is checked to ensure that it is in the proper range. For instance, if 1/2 x 2exp(127) is added to 3/4 x 2exp(127) mantissa overflow occurs which causes the exponent to be incremented out of range (5/8 x 2exp(128)). When exponent overflow occurs, the overflow constant of 1-1/2exp(N) x 2exp(127) (where N=24, 40 or 56 depending on the precision of the result) is the result. Also, the CPU overflow flip-flop is set, indicating that it is not a valid result.

Conversely, say that 1/2 x 2exp(-128) is added to -3/4 x 2exp(-128) Since this mantissa result of -1/4 undergoes normalization, the exponent is decremented to -129 which is out of range. In the case of exponent underflow, the result of all zeros is returned to the CPU and the CPU overflow flip-flop is set. However, if the exponent is in range, and the operation is complete, then the CPU overflow flip-flop is not set.

2.7 Summary of Addition/Subtraction

1. Load operands into registers.
2. Check that the exponent difference is less than the swamp constant.
3. Equalize the exponents by shifting the smaller mantissa to the right, and incrementing its exponent.
4. Add/subtract the mantissas.
5. If mantissa overflow occurs, shift in the proper sign and increment the exponent.

6. If the mantissa is not normalized, shift it left while decrementing the exponent.
7. Round the mantissa if necessary. If mantissa overflow occurs, correct it.
8. Check for exponent overflow or exponent underflow.
9. Load the result into all registers.
10. Set the CPU overflow flip-flop if an overflow or underflow occurred.

3.0 FIX TO SINGLE OR DOUBLE INTEGER (Figure II-2)

The fix to integer operation converts a number from 32, 48 or 64 bit floating point format to single or double integer format. The conversion operation uses many of the same sequences that addition or subtraction use in the floating point processor. For instance, before the conversion can begin, the floating point number has to be checked that it is in the range of integers. The range of 16 bit single integers is [0, 2exp(15)-1] and [-1, -2exp(15)] while the range of 32 bit double integers is [0, 2exp(31)-1] and [-1, -2exp(31)]. The fix/float constant is 15 for conversion to single integers and is 31 for double integers if the operand's exponent is negative, the floating point processor returns zero as the result. If the exponent is greater than 15 (single integers) or 31 (double integers), the FPP returns a result of 2exp(15)-1 (single) or 2exp(31)-1 (double) and sets the CPU overflow flip-flop.

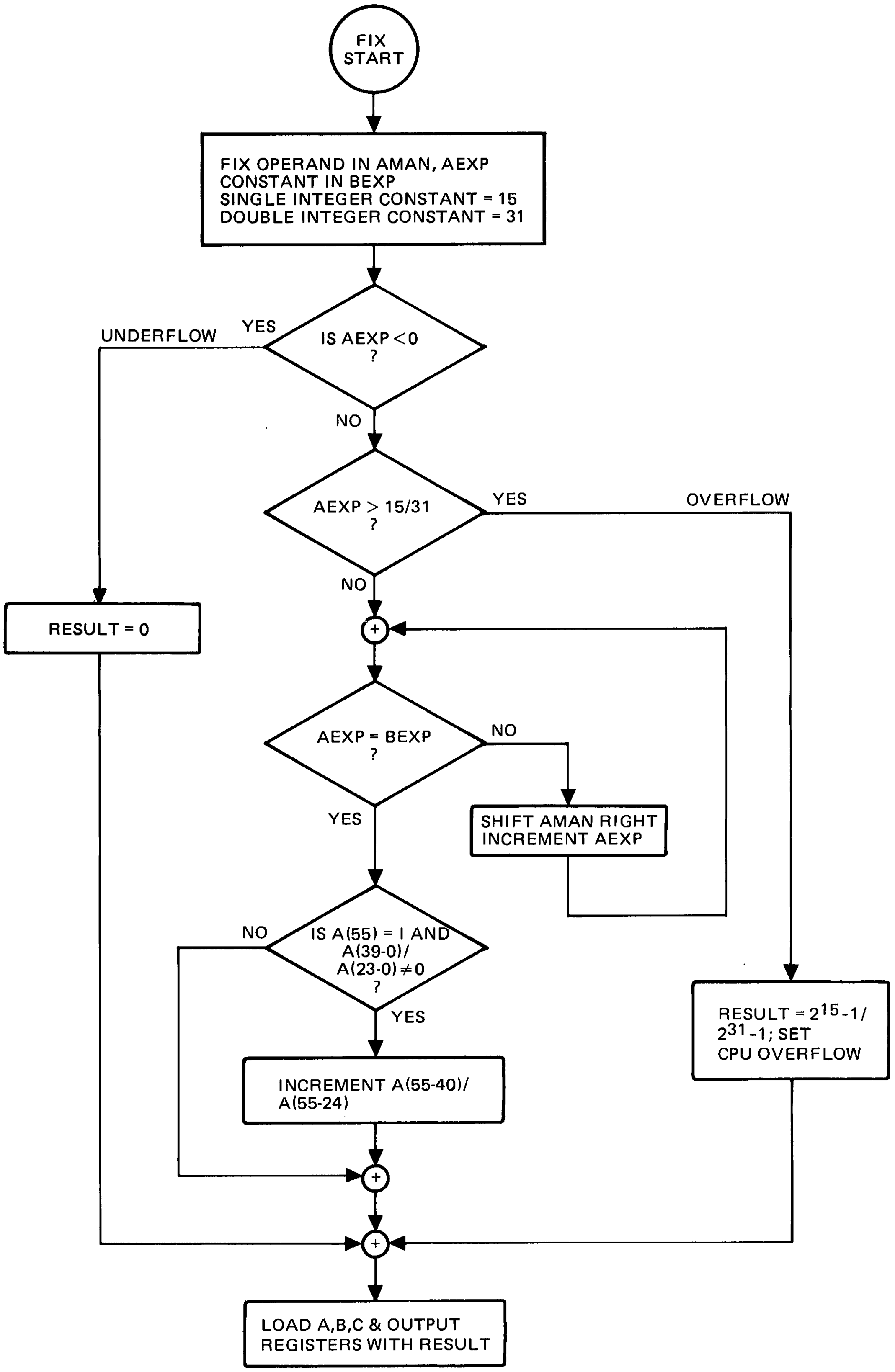

FIX START
FIX OPERAND IN AMAN, AEXP
CONSTANT IN BEXP
SINGLE INTEGER CONSTANT = 15
DOUBLE INTEGER CONSTANT = 31
IS AEXP < 0 ?
YES
UNDERFLOW
NO
AEXP > 15/31 ?
YES
OVERFLOW
NO
RESULT = 0
AEXP = BEXP ?
NO
SHIFT AMAN RIGHT
INCREMENT AEXP
YES
IS A(55) = I AND A(39-0)/ A(23-0) ≠ 0 ?
NO
YES
INCREMENT A(55-40)/ A(55-24)
RESULT = 2^15-1/ 2^31-1; SET CPU OVERFLOW
LOAD A,B,C & OUTPUT REGISTERS WITH RESULT

Figure II-2. Fix to Single or Double Integer Flowchart

(HP 1000 M/E/F-SERIES ERD)

However, if the operand is in integer range, the fix operation undergoes exponent equalization for the conversion process. The operand is in AMAN and AEXP while a constant, 15 or 31, is loaded into BEXP. AMAN is shifted to the right until AEXP equals the constant in BEXP. For instance, in the single integer case if the floating point operand is 5/8 x 2exp(3), the mantissa would be shifted 15-3 or 12 places to the right. The fix operation result is the uppermost sixteen bits of AMAN or in this case, the integer 5.

The fix operation results are checked for rounding also. Positive integers are always truncated, but negative integers are rounded if there are any "1"s to the right of the integer's least significant bit in AMAN. This is the case if the original operand's mantissa had "1"s beyond the integer's LSB or shifting the mantissa right put "1"s beyond the integer's LSB. This rounding procedure maintains symmetry about zero. Note that +1 1/2 is truncated to +1, while -1 1/2 is rounded to -1.

Summary of Fix to Single/Double Integer

1. Load floating point operand in AMAN, AEXP.
2. Load the integer test constant, 15/31, into BEXP.
3. Check that AEXP is positive and not greater than 15/31 to ensure that the floating point operand can be converted to a valid integer.
4. Go through the exponent equalization sequence.
5. Round result if necessary.
6. Load result in all registers.

4.0 FLOAT FROM SINGLE OR DOUBLE INTEGER (Figure II-3)

The float operation converts a single or double integer into a 32, 48 or 64 bit floating point number. The integer is loaded into AMAN while a constant is loaded into AEXP and BEXP. The exponent constant is 15 for single integers and 31 for double integers. Float is the simplest of all operations, since the operand merely undergoes normalization. For example, say that a single integer operand is 12 which is 000014 (octal). This operand must be shifted left eleven places to be normalized. Since the normalization sequence decrements the exponent with each shift, the final exponent is 15-11 = 4. Thus, the resulting operand is 3/4 x 2exp(4) = 12. The FPP never rounds a FLOAT result. Also, since all single and double integers are in the floating point number range, overflow and underflow can never occur.

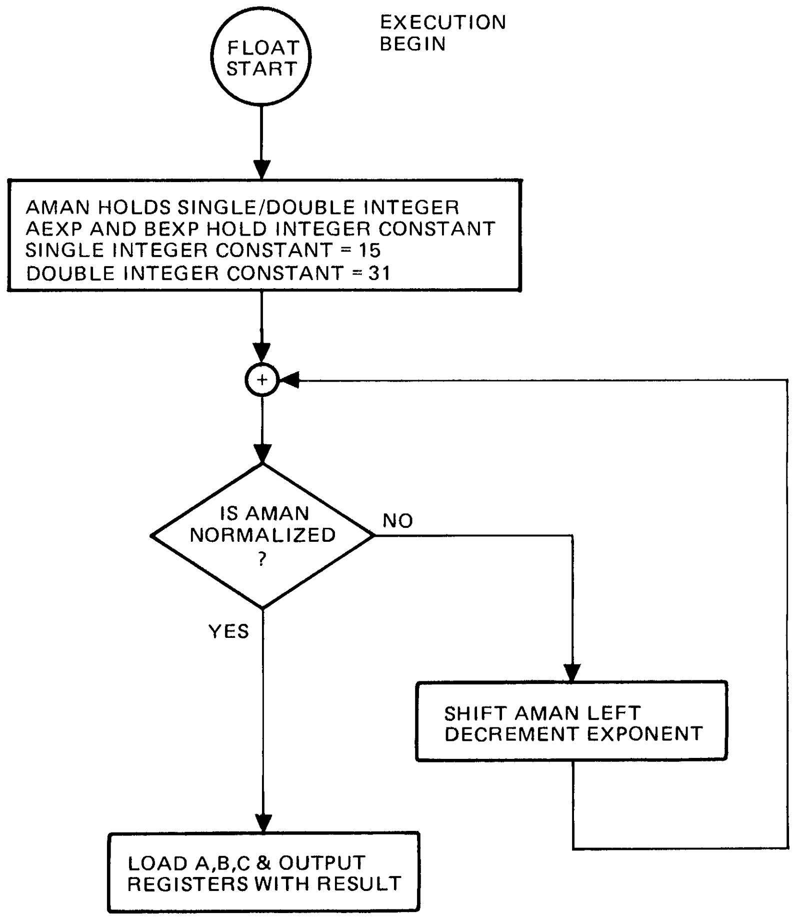


Figure II-3. Float from Single or Double Integer

(HP 1000 M/E/F-SERIES ERD)

5.0 MULTIPLICATION

One of the primary design goals of the floating point processor was to implement the operations add, subtract, multiply, divide, fix to integer and float from integer with a minimum register and data path configuration which would all fit on a single printed circuit board. Floating point addition and subtraction algorithms involve shifting mantissas right to equalize exponents, adding or subtracting mantissas and then shifting the resultant mantissa left to normalize it. Thus, the minimum hardware configuration had to include bidirectional shift registers and arithmetic logic units (ALUs). In light of this configuration, the subsequent algorithm investigation focused on multiplication and division algorithms that dictated sequences of shift cycles and arithmetic cycles.

5.1 Elaboration of Multiplication Algorithm

Multiplication is the most complex of all the floating point operations. Not only does it follow the most complicated of the FPP's algorithms, but also it uses special circuits which make it go faster.

For example, the multiply algorithm shifts over strings of zeros and ones while detecting and correcting for isolated zeros or ones. The description of this algorithm will start with and build upon the simplest type of multiply methods. The simplest multiply algorithm scans the multiplier and adds a copy of the multiplicand to the partial product at each "1" bit position of the multiplier. Observe that a bit pattern in the multiplier of ...1000010... is equivalent to 2exp(n+5) + 2exp(n). Also, ...0111110... equals 2exp(n+4)+2exp(n+3)+2exp(n+2)+2exp(n+1)+2exp(n) or, more importantly, 2exp(n+5)-2exp(n); note that one addition and one subtraction replaces four additions. Since any multiplier can be reduced to a string of ones and zeros, multiplication can be a process of add or subtract cycles and shift cycles. Since FPP shift cycles take only 50 nanoseconds while arithmetic cycles take 125 nanoseconds, the algorithmic goal is to perform as few ALU cycles as possible. With this in mind, what happens in the sequence ...0001000...? If the lone one is treated as a string, the above method dictates +2exp(n+1)-2exp(n) -- one addition and one subtraction. Obviously, one addition should suffice. If a history bit (H) is used to indicate the type of string, a string of ones or a string of zeros, that is being shifted over, the isolated bit can be detected, and the single addition will be performed.

Since multiply undergoes an arithmetic cycle only at the start and end of strings and once at isolated bits, the processor will never perform two consecutive arithmetic cycles. Thus, an arithmetic cycle is always followed by a shift cycle. Also, each arithmetic cycle includes a shift operation. This means that each time a partial product passes through the arithmetic circuits, the algorithm shifts it twice. The FPP accomplishes this double shift through multiplexers which are placed at the output of the arithmetic circuitry. With the use of the multiplexers, every arithmetic cycle eliminates two shift cycles.

5.2 Multiplier Bits Truth Table

Here is the truth table from which the algorithm dictates the sequence of arithmetic and shift cycles which is based on the multiplier bits. The multiplication process scans the multiplier from right to left looking at two bits (C(n+1),C(n)) at a time. Note that the multiplier is in tne FPP's C-register and C(n+1), C(n) are the two least significant bit portions of the C-register. The history bit (H) records what type of string has been shifted over. The history bit and partial product are cleared before multiplication begins.

C(n+1)	C(n)	H	Next H	Significance	Operation
0	0	0	0	continue string of 0's	Shift partial product right
0	0	1	0	end of string of 1's; start of string of 0's	Add multiplicand to partial product, shift
0	1	0	0	isolated 1 case	Add multiplicand to partial product; shift
0	1	1	1	continue string of 1's	Shift partial product right
1	0	0	0	continue string of 0's	Shift partial product right
1	0	1	1	isolated zero	Subtract multiplicand from partial product, shift
1	1	0	1	end of string of 0's; start string of 1's	Subtract multiplicand from partial product, shift
1	1	1	1	continue string of 1's	Shift partial

5.3 Multiplication Decision Equations

The following equations derived from the truth table control the multiplication process. In the equations a plus sign (+) represents a logical OR, and a colon (:) represents a logical AND.

1. Decision on whether to start a shift cycle or an ALU cycle next.

   Shift cycle = H $\overline{\text{XOR}}$ C(n)
   ALU cycle = H XOR C(n)

2. Decision on whether to add or subtract the multiplicand from the partial product.

   Add = ALU cycle : $\overline{C(n+1)}$
   Subtract = ALU cycle : C(n+1)

3. Determination of the next history bit. Next H = [C(n+1):C(n)] + C(n):H + C(n+1):H

For more information on the multiplication algorithm see "The Logic of Computer Arithmetic" by Ivan Flores, Chapter 10.

Figure II-4 displays a sample multiplier and the operations which would be performed during its multiplication. Figure II-5 shows an example of multiplying 7 by -11463.

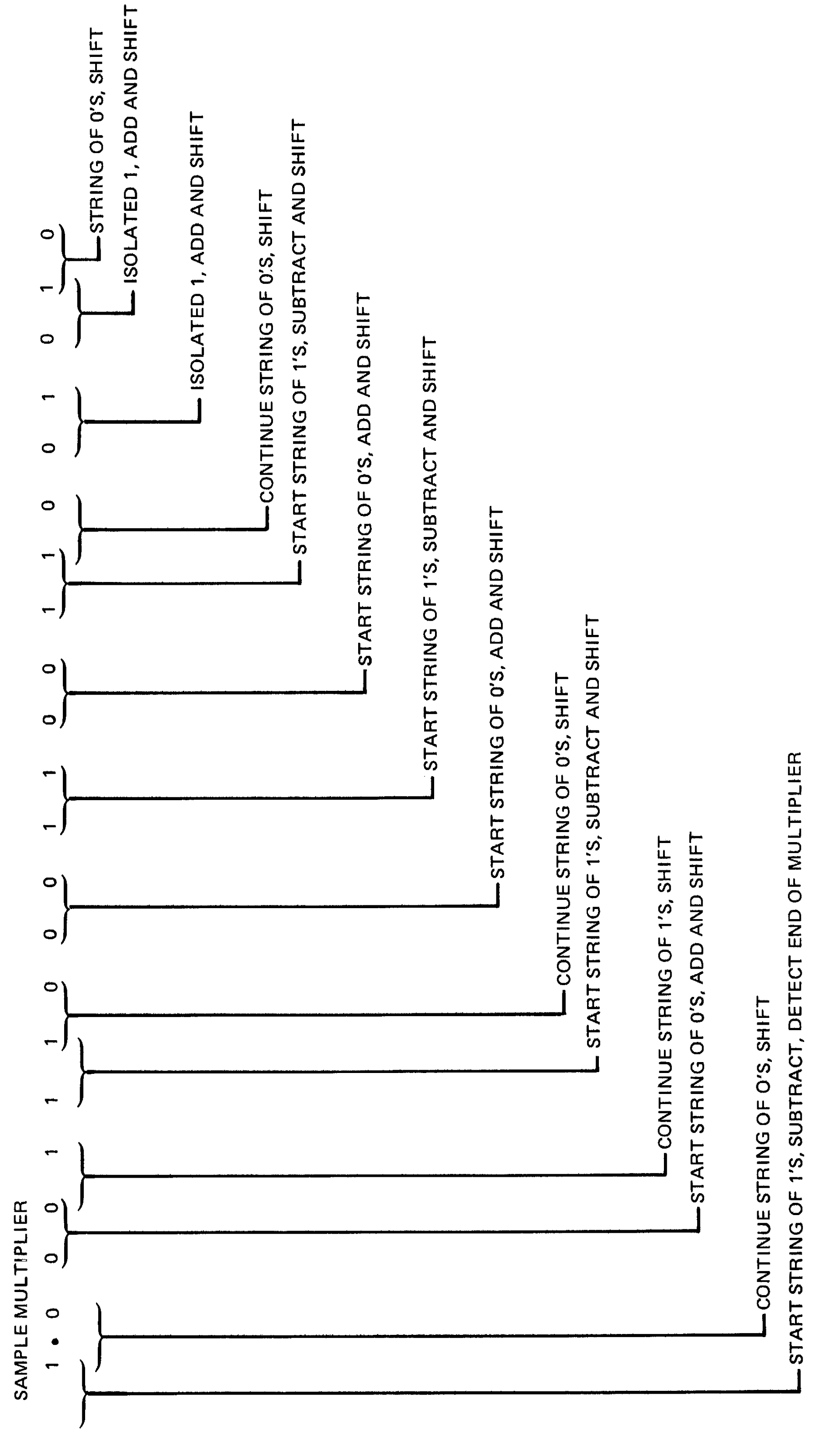

SAMPLE MULTIPLIER
1 • 0 0 0 1 1 1 0 0 0 1 1 0 0 1 1 0 0 1 0 1 0
START STRING OF 1'S, SUBTRACT, DETECT END OF MULTIPLIER
CONTINUE STRING OF 0'S, SHIFT
START STRING OF 0'S, ADD AND SHIFT
CONTINUE STRING OF 1'S, SHIFT
START STRING OF 1'S, SUBTRACT AND SHIFT
CONTINUE STRING OF 0'S, SHIFT
START STRING OF 0'S, ADD AND SHIFT
START STRING OF 1'S, SUBTRACT AND SHIFT
START STRING OF 0'S, ADD AND SHIFT
START STRING OF 1'S, SUBTRACT AND SHIFT
CONTINUE STRING OF 0'S, SHIFT
ISOLATED 1, ADD AND SHIFT
ISOLATED 1, ADD AND SHIFT
STRING OF 0'S, SHIFT

Figure II-4. Multiplication--Sequence Operation Example

MULTIPLIER = −11463 = $\frac{-11463}{16384} \times 2^{14}$ = 1.01001100111001, 00001110

MULTIPLICAND = 7 = 7/8 × 2^3 = 0.11100$\bar{0}$, 00000011

COUNTER	C_1	C_0	H_i	NEXT H	
15	0	1	0	0	1. ADD MULTIPLICAND TO PARTIAL PRODUCT AND SHIFT TWICE
13	1	0	0	0	2. SHIFT PARTIAL PRODUCT
12	1	1	0	1	3. SUBTRACT (OR ADD THE 2'S COMPLEMENT) MULTIPLICAND FROM PARTIAL PRODUCT AND SHIFT TWICE
10	0	1	1	1	4. SHIFT PARTIAL PRODUCT
9	0	0	1	0	5. ADD MULTIPLICAND TO PARTIAL PRODUCT AND SHIFT TWICE
7	1	1	0	1	6. SUBTRACT (OR ADD THE 2'S COMPLEMENT) MULTIPLICAND FROM PARTIAL PRODUCT AND SHIFT TWICE
5	0	0	1	0	7. ADD MULTIPLICAND TO PARTIAL PRODUCT AND SHIFT TWICE
3	0	1	0	0	8. ADD MULTIPLICAND TO PARTIAL PRODUCT AND SHIFT TWICE
1 0	1	1	0	1	9. SUBTRACT (OR ADD THE 2'S COMPLEMENT) MULTIPLICAND FROM PARTIAL PRODUCT SINCE COUNTER = 1, SHIFT ONCE, NOT TWICE

COUNTER = 0 → END OF PRODUCT FORMING PROCESS
PRODUCT IS NOT NORMALIZED:
SHIFT MANTISSA LEFT, DECREMENT EXPONENT

RESULT = $-\frac{80241}{131072} \times 2^{17}$ = −80241

```
MULTIPLICAND                          MULTIPLIER
    BMAN                              C-REGISTER
0.11100̄ = −(1.00100̄)                  1 . 0 1 0 0 1 1 0 0 1 1 1 0 0 1
                                      └─┘ └─┘ └─┘ └─┘ └─┘ └─┘ └─┘
AMAN = PARTIAL PRODUCT                 9   8   7   6   4   2   1
                                                     └─┘ └─┘
    0.00000000                                        5   3
1.  0.11100000
    ----------
    0.11100000
    0.00111000000
2.  0.000111000000
3. +1.0010000
   -------------
    1.0011110000000                   PRODUCT EXPONENT =
    1.11001111000000                   MULTIPLICAND EXPONENT +
4.  1.11100111100000                   MULTIPLICAND EXPONENT + 1
5. +0.11100̄
   ----------------
    0.110001111100000                  PRODUCT EXPONENT =
    0.0011000111100                     00000011
6. +1.00100                            +00001110
   ----------------                    ---------
    1.010100011100                      00010001
    1.1101010001111                           +1
7. +0.11100̄                            ---------
   ----------------                     00010010
    0.1011010001111
    0.001011010001111
8. +0.11100̄
   ----------------
    1.000001010001111
    0.01000001010001111
9. +1.00100̄
   -----------------------
    1.01100001010001111              NORMALIZATION:
    1.10110000101000111 1              00010010
                                            -1
                                       --------
    1.01100001010001111                00010001  RESULT
   ==============================================
```

Figure II-5. Multiplication Example 7 x 11463

5.4 Rounding Techniques in Multiplication

Since multiplication produces a double length mantissa product, the product must be rounded or truncated back to the original, standard length. For instance, in a 32 bit single precision multiply, the 24 bit mantissa operands produce a 48 bit mantissa product. The 48 bit product must be rounded or truncated to form a proper 24 bit result. The FPP develops all partial products in the A mantissa register (AMAN). This register is 56 bits wide, AMAN (55-0), and partial products are oriented so that AMAN (55) is the sign bit and AMAN (54) is the first bit to the right of the binary point. Thus, the single precision product occupies AMAN (55-32), extended precision AMAN (55-16) and double precision AMAN (55-0).

The rules for rounding in multiplication are the same as the rounding rules in addition or subtraction. Namely, positive mantissas are rounded if the first guard bit (the bit to right of the least significant bit) is a one. On the other hand, negative mantissas are rounded only if the first guard bit is a one and there is another one to the right of that guard bit. The round information circuits hold the guard bit and the "sticky" bit. The sticky bit is a latch register which gets set if it detects a one to the right of the guard bits. Since a product may be normalized, or shifted left, at most two places, the round circuits hold three guard bits which represent the first three bits to the right of the product′s LSB. Thus, during normalization the bit in the third guard bit position may be shifted left to the first guard bit position.

Multiplication sets up the rounding information register in two ways. First of all, as the partial product is shifted right during shift cycles, the guard bits are also shifted right. The bit from the partial product′s LSB is shifted into the first guard bit. Since the final product′s LSB position, depends on the precision of the operation, multiplexers are used to select the proper LSB for the guard bit. Thus, the guard bit may be fed from AMAN (32), AMAN (16) or AMAN(0) depending on the precision of the operation.

Multiplication also sets up the round information register during arithmetic cycles. Remember that in arithmetic cycles the output of the ALUs is shifted to the right twice through multiplexers. Thus, the two least significant partial product bits must be routed to the first two guard bits. As the twice shifted ALU results are loaded into the partial product register (AMAN), the two bits to the right of the LSB are loaded into the first two guard bits of the round register. The outputs of the first two guard bits are routed to the inputs of the third guard bit and sticky bit every time the round register is loaded. In this way the round register is effectively shifted twice to the right. Thus, every multiplication ALU cycle has the effect of shifting two bits into the round register. Again, since the partial product′s LSB position depends on the precision of the operation, another set of multiplexers are used to select either ALU(33,32), ALU(17,16) or ALU (1,0) to be loaded in the round register. Thus for any precision, the guard and sticky bits always accurately represent the bits to the right of the LSB, even while the partial product is shifted right once, shifted right twice or left shifted. A multiplication flowchart is shown in Figure II-6.

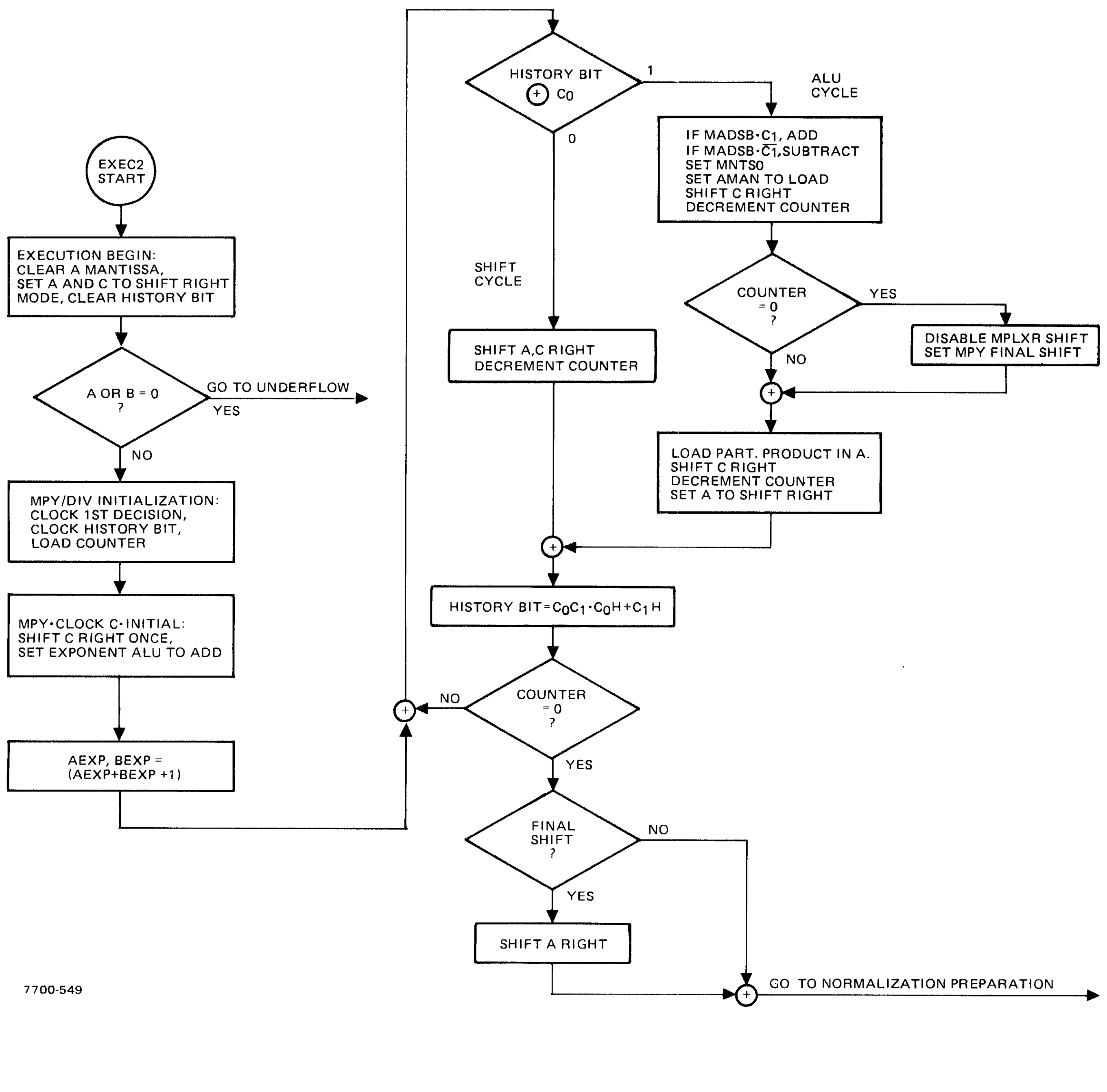


Figure II-6. Multiplication Flowchart

(HP 1000 M/E/F-SERIES ERD)

6.0 DIVISION

6.1 Division Process Fundamentals

The division process is similar to the multiplication process, in that it consists of ALU and shift cycles. The A register holds the original dividend which becomes the partial remainder. The B register holds the divisor, while the quotient is developed in the C register. Division uses a non-restoring algorithm which shifts over strings of ones and zeros in the partial remainder. At the end of the strings the divisor is added to or subtracted from the partial remainder. The algorithmic goal is to reduce the partial remainder to zero.

Looking at the division process in greater detail, the partial remainder is shifted left until it is in normalized form. Then if the divisor and partial remainder are both positive or both negative, the divisor is subtracted from the partial remainder. Otherwise, the divisor is added to the partial remainder. The sign of the arithmetic result, which is the new partial remainder, determines the sense of the quotient bit for that cycle. If the new partial product is not normalized, [the case when the sign bit, AMAN(55), is the same sense as AMAN(54)], then it is shifted left until it is normalized. Since the divisor is normalized, the partial remainder should be left in normalized form. Otherwise, it would not be significant to perform arithmetic with an operand that is not normalized (under range). Also, shift cycles take 50ns, whereas ALU cycles take 175ns, so that division is faster if shift cycles are performed whenever possible. Each shift cycle as well as an ALU cycle forms a quotient bit. However, in shift cycles the quotient bit is determined by the Exclusive OR function of the signs of the divisor and partial remainder. A counter controls the number of quotient bits formed and signals the completion of division. The equations and circuitry which direct the division process are summarized below.

6.2 Division Decision Equations

1. The decision on whether to perform an ALU cycle or shift cycle:

   If AMAN(55) = AMAN(54), do a shift cycle. Otherwise, do an ALU cycle.

2. The decision of whether to add or subtract the divisor from the partial product in an ALU cycle:

   If AMAN (55) = BMAN (55), subtract. Otherwise, add.

3. Quotient bit determination:

   During shift cycles: quotient bit = AMAN(55) XORed with BMAN(55) During ALU cycles: quotient bit = ALU(55) XORed with BMAN(55)-

6.3 Corrections to the Quotient

When the quotient is formed, it may have the wrong sign. The wrong quotient sign is produced when the first ALU cycle is successful, which is when the |divisor| < |dividend|. For example, in the case of 3/4 divided by 1/2, the first ALU cycle subtraction results in 3/4-1/2=1/4 which means the first quotient bit is a 1. Since the first quotient bit ends up in the quotient sign position, this quotient will have the wrong sign. If the quotient has the wrong sign, it is sent through the mantissa overflow sequence to correct it. This sequence shifts the mantissa right while shifting in the proper sign, and increments the exponent.

The division algorithm develops a one's complement representation of the quotient. Thus, negative quotients may have to be incremented in order to convert them to the two's complement representation. The floating point processor combines the conversion with the rounding procedure. The floating point processor develops an extra quotient bit called the guard bit which is used in the decision of whether or not to round the quotient. Note that if the quotient has the wrong sign, it is shifted right in the mantissa overflow sequence, so that the LSB is shifted into the guard bit position. The rounding decision is made after a quotient sign adjustment. The quotient is rounded if the guard bit is a one, regardless of whether the quotient is positive or negative. By combining the two's complement conversion step with the rounding step after the sign adjustment sequence, all division results are properly rounded. Figure II-7 provides a division example and the flowchart in Figure II-8 summarizes the division process.

125 ÷ −5; 125 = 125/128 × 2^7; −5 = −5/8 × 2^3

DIVIDEND → A REGISTER AMAN = 125/128 = 0.1111101; AEXP = 7 = 00000111
DIVISOR → B REGISTER BMAN = −5/8 = 1.0110000; BEXP = 3 = 00000011
QUOTIENT DEVELOPED IN C REGISTER

THIS EXAMPLE CONCERNS 8 BIT REGISTERS	PARTIAL REMAINDER A	DIVISOR B	QUOTIENT C	EXPONENT
1a. AMAN IS NORMALIZED, SO DO ALU CYCLE b. SIGN OF AMAN ≠ SIGN OF BMAN, SO ADD	0.1111101 + 1.0110000	1.0110000		00000111 − 00000011
c. ALU RESULT SIGN ≠ BMAN SIGN, C_0 =0; SHIFT PARTIAL REMAINDER	0.0101101		0	00000100
2a. AMAN IS NORMALIZED → ALU CYCLE b. AMAN SIGN ≠ BMAN SIGN → ADD	0.1011010 + 1.0110000			
c. ALU SIGN ≠ BMAN SIGN → C_0 =0, SHIFT PARTIAL REMAINDER	0.0001010		00	
3a. AMAN IS NOT NORMALIZED → SHIFT CYCLE b. SHIFT, AMAN SIGN ≠ BMAN SIGN → C_0 =1	0.0010100		001	
4a. AMAN IS NOT NORMALIZED, SHIFT CYCLE b. SHIFT, AMAN SIGN ≠ BMAN SIGN → C_0 =1	0.0101000		0011	
5a. AMAN IS NORMALIZED → ALU CYCLE b. AMAN SIGN ≠ BMAN SIGN → ADD	0.1010000 + 1.0110000			
c. ALU SIGN ≠ BMAN SIGN → C_0 = 0, SHIFT	0.0000000		00110	
6a. AMAN IS NOT NORMALIZED → SHIFT CYCLE b. AMAN SIGN ≠ BMAN SIGN, C_0 = 1, SHIFT	0.0000000 0.0000000		001101	
7a. AMAN IS NOT NORMALIZED → SHIFT CYCLE b. AMAN SIGN ≠ BMAN SIGN, C_0 = 1, SHIFT	0.0000000		0011011	
8a. AMAN IS NOT NORMALIZED → SHIFT CYCLE b. AMAN SIGN ≠ BMAN SIGN, C_0 = 1, SHIFT	0.0000000		00110111	00000101
END OF QUOTIENT FORMING PROCESS — MOVE QUOTIENT TO AMAN. QUOTIENT HAS WRONG SIGN; SHIFT IN PROPER SIGN AND INCREMENT	0.0110111 0.0011011	1 ← GUARD BIT POSITION		00000101
FIRST GUARD BIT = 1, SO ROUND QUOTIENT	+1 1.0011100			00000101

∴ FINAL QUOTIENT = 1.0000000(MANTISSA), 00000101(EXPONENT)
= −25/32 × 2^5 = −25

Figure II-7. Division Example 125/-5

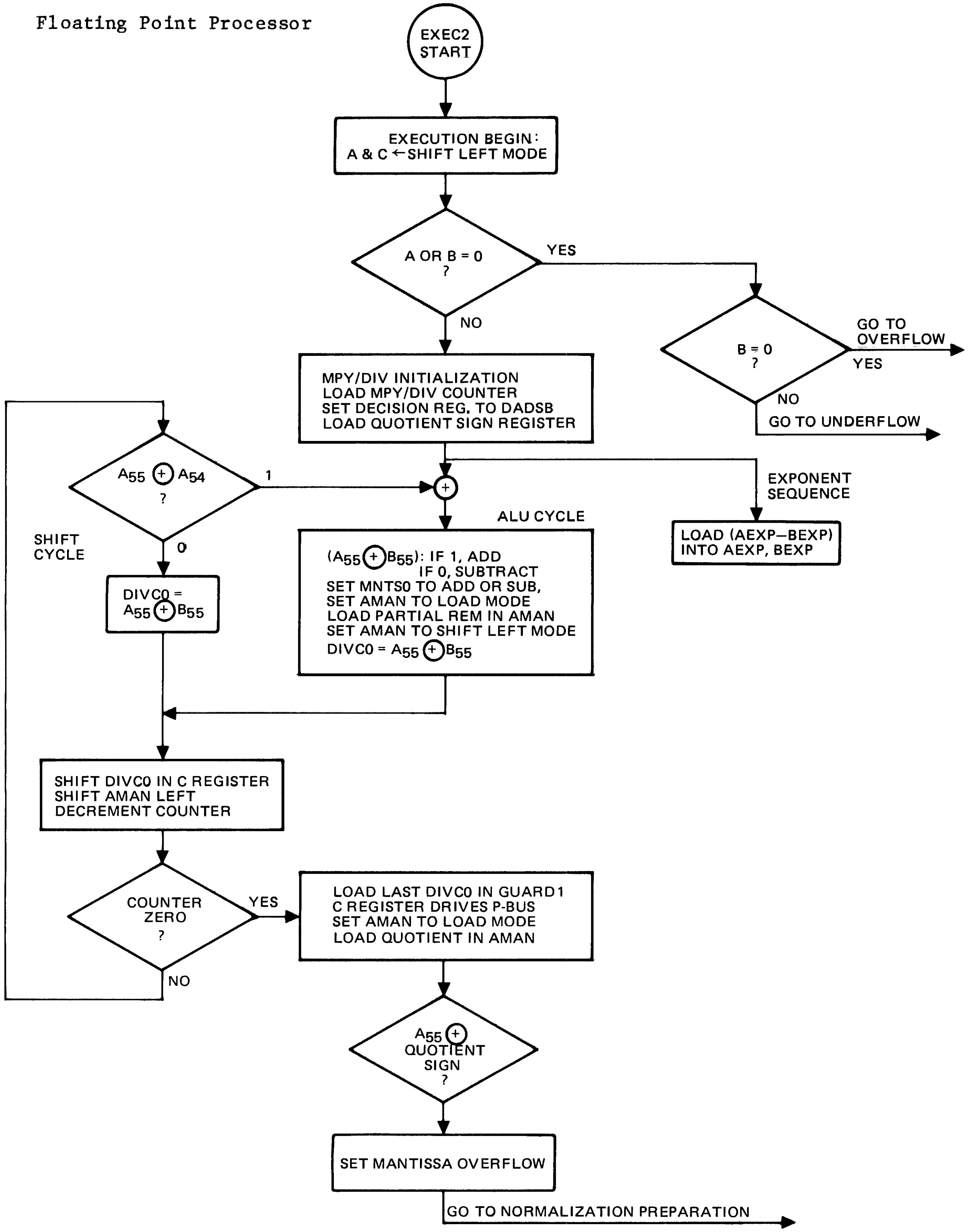


Figure II-8. Division Flowchart

(HP 1000 M/E/F-SERIES ERD)

7.0 ROUNDING TECHNIQUES (Figure II-9)

In order to minimize error propagation in a floating point calculation, each floating point operation must produce results that are as accurate as possible. Some operations generate mantissa results that have more than 24, 40 or 56 bits. For instance, multiplication of two 24 bit mantissas generates a 48 bit product. The excess bits are used in the decision of whether to truncate or round the result to form a proper length mantissa. Rather than use expensive double length registers, the FPP holds information about the extra bits in a single 4 bit register.

This rounding information register holds three guard bits, which represent the three bits to the right of the resulting mantissa's LSB. Also, the round decision uses a sticky bit to indicate if there are any ones in the bits to the right of the guard bits. The sticky bit latch is an RS flip-flop which is set by a "one" that is right shifted out of the guard bits. It is cleared between operations. Although the FPP maintains a single round register, rounding information is routed to the register in four ways.

First of all, as operands are shifted right, the bit from the LSB position is shifted into the round register. Since the LSB position depends on the precision of the floating point operation, multiplexers are used to shift either the thirty-second, sixteenth, or the zero bit into the first guard bit.

A refinement is made on the shift-right-multiplex-LSB process in the subtraction case where the subtrahend is undergoing exponent equalization. In subtraction, since the subtrahend is complemented and then added to the minuend, the subtrahend bits entering the round register have to be complemented. One method of forming the two's complement of a binary number is to start at the right end of the number and move left. Until a 1 is encountered, leave all zeros as they are, then take the one's complement of all the bits to the left of the first 1. For example, the bits 0101000 would be passed to the first guard bit as 1011000. For greater detail, the FPP passes the bits shifted out of the LSB as they are until a 1 is detected, which sets a latch. The set latch causes all succeeding bits to be complemented. In this way, the round register effectively maintains a complemented subtrahend.

Multiplication sets up the round register in a third way. Since the partial product is shifted twice to the right during ALU cycles, the two LSBs must be loaded into the round register. Again, as the LSB position depends on the precision of the operation, a second set of multiplexers are used to sort out the proper LSB for the round register.

In contrast to the other floating point operations, division uses different information in its rounding decision. The division process develops one extra quotient bit which is loaded into the first guard bit. After the quotient and round register are adjusted for mantissa overflow or normalization, and if the first guard bit is a 1, the quotient is rounded.

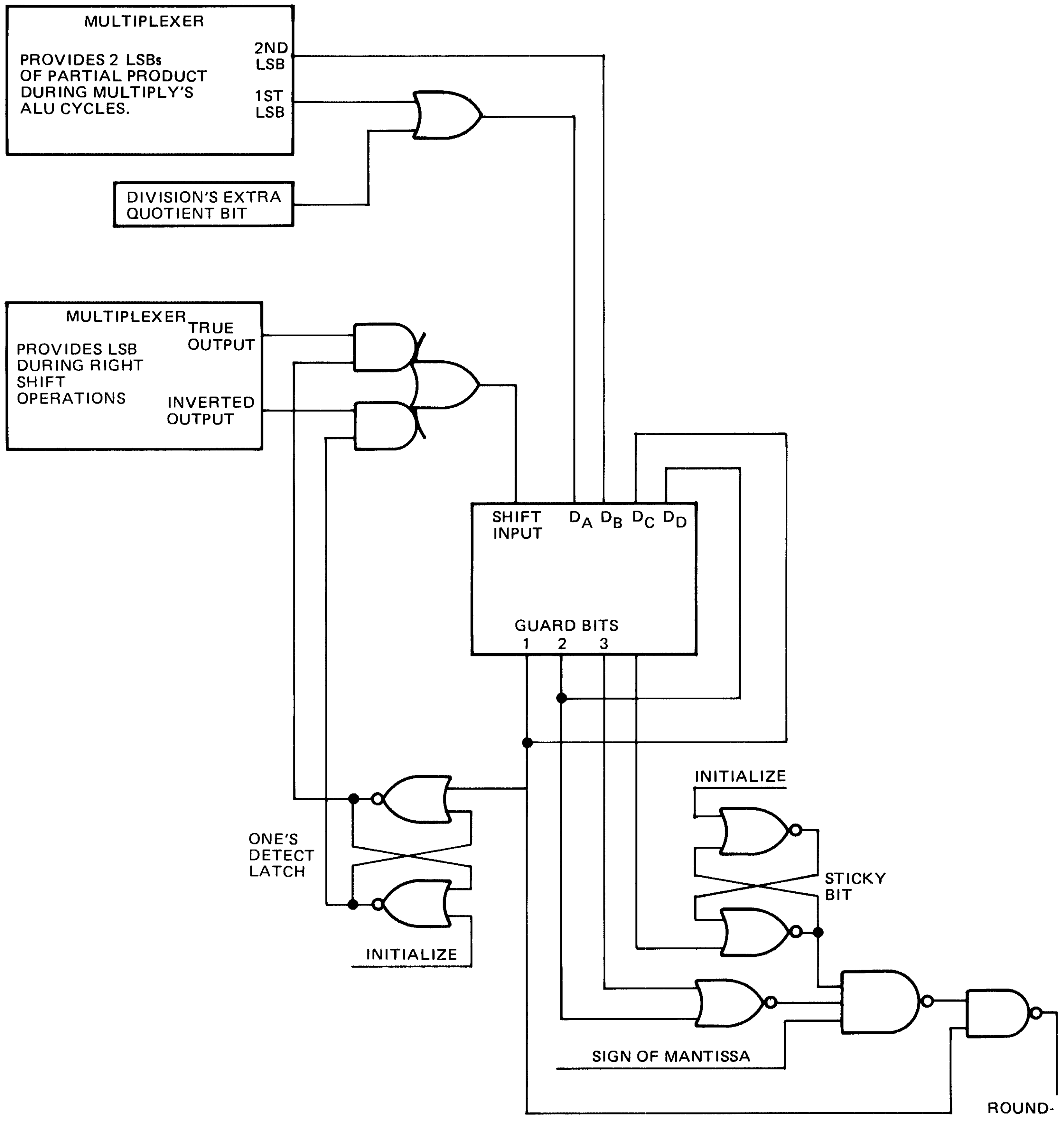


Figure II-9. Rounding Techniques

(HP 1000 M/E/F-SERIES ERD)

```
+------------------------------------------------------+-----------------+
|                                                      |                 |
|  DETAILED DISCUSSION OF FLOATING POINT PROCESSOR     |  SECTION   III  |
|                                                      |                 |
+------------------------------------------------------+-----------------+
```

1.0 INTRODUCTION

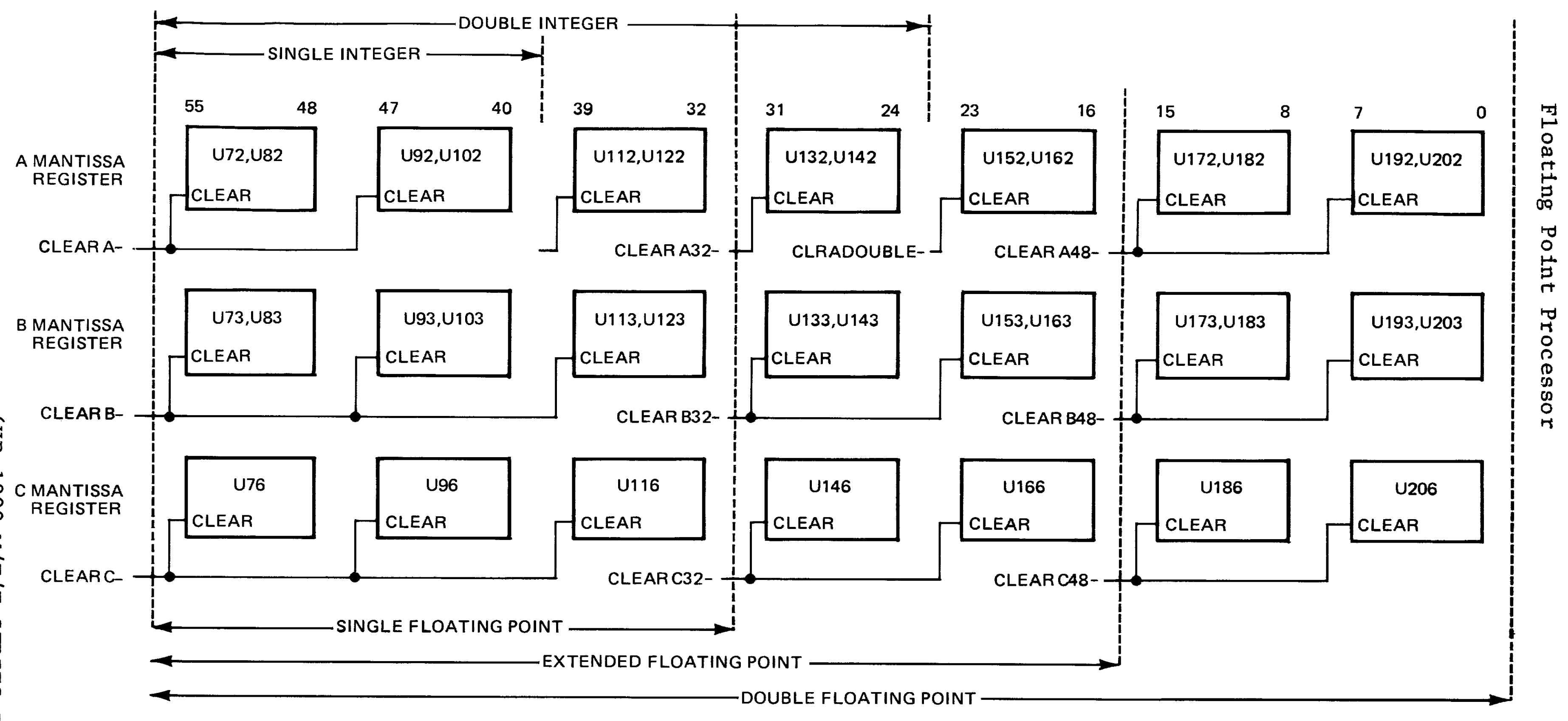
Section III describes the operation of the floating point processor hardware. Section II presented the processes the floating point operations undergo, and Section III discusses merely how the hardware works. The implementation discussion moves from flowchart summaries to circuit models including detailed references of the schematics.

The floating point processor consists of two printed circuit assemblies (PCAs), the 12740-60001 Arithmetic PCA and the 12740-60002 Control PCA. Accordingly, the schematics are part number D-12740-60001-51 through -56 and part number D-12740-60002-51 through -55 respectively. This section first discusses the circuitry and layout of the arithmetic PCA and then the circuitry of the control PCA.

2.0 ARITHMETIC PCA 12740-60001

The arithmetic PCA holds the operand registers, data paths and arithmetic logic units (ALUs) of the floating point processor. The arithmetic PCA is divided between a mantissa section and an F,OFFent section as shown in Figure I-3. Each section holds F=OFF, output, operand registers and ALUs which are linked through an internal bus named the P-bus. The input and output registers transmit data to and from the CPU across the microprogrammable processor port. Besides input and output registers, there are three main registers in the mantissa section (AMAN, BMAN and CMAN) and two registers in the exponent section (AEXP and BEXP).

In the mantissa section AMAN, BMAN and CMAN registers hold the operand mantissas. These registers are 56 bits wide to accommodate 55 bits plus the sign of double precision mantissas. AMAN (55-0) consists of one 74S194 and thirteen 74LS194As which are all four bit bidirectional universal shift registers. AMAN (55-52) holds the most significant bits and is the 74S194 part. AMAN(55-53) drive the control board. BMAN (55-00) is made up of fourteen 74LS194A parts. CMAN(55-0) consists of eight 74S299 parts. The CMAN parts are eight bit universal shift/storage registers. Since at most only a couple of bits of the C register are used at any time during an operation, it can be implemented with octal parts whose inputs and outputs are multiplexed on the same pins. In all of the mantissa registers bits 55-32, bits 55-16 and bits 55-0 hold, respectively, single, extended and double precision mantissas (Figure III-1).



$\text{CLEAR A48-} = \overline{[\overline{\overline{\text{IREG3}} \cdot \text{EXECBGN+}} + \text{FIXOP-} \cdot \overline{(\text{ROUNDEN-} \cdot \text{TERMCTL1-})}]} \cdot \text{3WDOP} \cdot \text{CLRADOUBLE-}$

$\text{CLRADOUBLE-} = [(\text{FLOAT+} \cdot \text{FPCONSTANTEN+}) + \text{FIXOP-} \cdot \overline{(\text{ROUNDEN-} \cdot \text{TERMCTL1-})}] \cdot \text{CLEAR A32-}$

$\text{CLEAR A32-} = \overline{\text{2WDOP} \cdot \text{FLOATDBL-} \cdot [\overline{(\overline{\text{IREG3}} \cdot \text{EXECBGN-})} + \text{FIXOP-} \cdot (\text{ROUNDEN-} \cdot \text{TERMCTL1-})]} \cdot \text{CLRASINGLE-}$

$\text{CLRASINGLE-} = \overline{[(\text{FLOATSGL} \cdot \text{FPCONSTANTEN+}) + \text{FIXSGL+} \cdot \overline{(\text{ROUNDEN-} \cdot \text{TERMCTL1-})}]} \cdot \text{CLEARA-}$

$\text{CLEAR A-} = \overline{(\text{MPYOP+} \cdot \text{EXECBGN+})} \cdot \overline{(\text{EQUALA+} \cdot \text{SWAMPCTL4+})}$

$\text{CLEAR B48-} = \overline{(\text{3WDOP} \cdot \text{BRGCKI+})} \cdot \text{CLEAR B32-}$

$\text{CLEAR B32-} = \overline{(\text{2WDOP} \cdot \text{BRGCKI+})} \cdot \text{CLEARB-}$

$\text{CLEARB-} = \text{FPCONSTANTEN-} \cdot \overline{(\text{EQUALB+} \cdot \text{SWAMPCTL4+})} \cdot \text{CLEARBCTL-}$

$\text{CLEARC48-} = \text{3WDOP+} \cdot [\overline{(\overline{\text{IREG3}} \cdot \text{EXECBGN-})} + \text{FIXOP-} \cdot \overline{(\text{ROUNDEN-} \cdot \text{TERMCTL1-})}] \cdot \text{CLEARC48-}$

$\text{CLEARC32-} = \overline{\text{2WDOP+} \cdot [\overline{(\overline{\text{IREG3}} \cdot \text{EXECBGN-})} + \text{FIXOP-} \cdot \overline{(\text{ROUNDEN-} \cdot \text{TERMCTL1-})}]} \cdot \text{CLEARC-}$

$\text{CLEARC-} = \text{UNDERFLOW-}$

Figure III-1. Mantissa Register Logic-Clear

The C register is implemented with octal parts whose inputs and outputs are multiplexed, so that the part can achieve eight bit capability within a 20 pin package. Additional high speed registers duplicate some of the C-register in order to accelerate multiplication. The multiplication decision algorithm use the two least significant bits of the multiplier which is held in the C register. As the multiply decision on whether to perform an ALU or shift cycle must be formed within 50 nanoseconds, Schottky registers must be used to provide the multiplier bits. In order to provide two multiplier bits within a 50 nanosecond shift cycle, the eight least significant bits of the C register multiplier are duplicated in two 74S194 four bit bidirectional shift registers. Since the precision of the operation determines the eight least significant bits (bits 47-40, 23-16 or 7-0 for single, extended or double precision, respectively), 74S153 dual 4-line-to-1-line data multiplexers route the appropriate eight bits to the duplicated multiplier look ahead registers.

As to the exponent section, AEXP and BEXP hold the exponents of the operands. Since exponent equalization, mantissa overflow and normalization processes increment or decrement the exponent, AEXP and BEXP are implemented in 74S169 parts which are synchronous four bit up/down counters. As AEXP and BEXP hold up to ten bits, these registers are formed from three 74S169 parts.

The function of the mantissa and exponent registers depends on the floating point operation being executed. For instance, AMAN and AEXP hold the augend in addition, the minuend in subtraction, the partial product in multiplication, the dividend and partial remainder in division and the floating point operand in fix to single/double integer. Also, AMAN holds the single integer (held in AMAN(55-40)) or double integer (held in AMAN(55-24)) operand in the float operation. BMAN and BEXP hold the addend in addition, the subtrahend in subtraction, the multiplicand in multiplication and the divisor in division. The third mantissa register, CMAN, holds the multiplier in multiplication, the quotient in division and is not used in any of the other floating point operations.

Both the mantissa and exponent sections have a bank of arithmetic logic units, 74S381. In the mantissa group, these four bit ALUs are used only in the A plus B or A minus B modes. Since the mantissa may contain 56 bits, fourteen 74S381s are required. Also, due to the long length of the mantissa, the speed of generating and propagating carries across the mantissa is accelerated through the use of two levels of 74S182 look-ahead carry generator circuits. On the other hand, as the exponent section requires only three 74S381s, its generate and propagate logic is implemented discretely with AND-OR-INVERT and NAND gates. The four modes of the 74S381s that the exponent section uses are the A plus B, A minus B, Inclusive-OR and clear modes.

In the mantissa section the outputs of the ALUs are routed to inputs of 74S257 quadruple 2-line to 1-line data selectors/ multiplexers. In one mode which is usually active, the output of an ALU is routed directly to the same bit position of the P-bus (ALU(50)) passes to P-bus(50)). In the other mode, which is active during multiplication, the selectors route ALU output data from two bit positions to the right to a P-bus position (ALU(52)) passes to P-bus(50)). Thus, the partial product is effectively right shifted twice as it passes through the multiplexer circuits (Figure III-2).

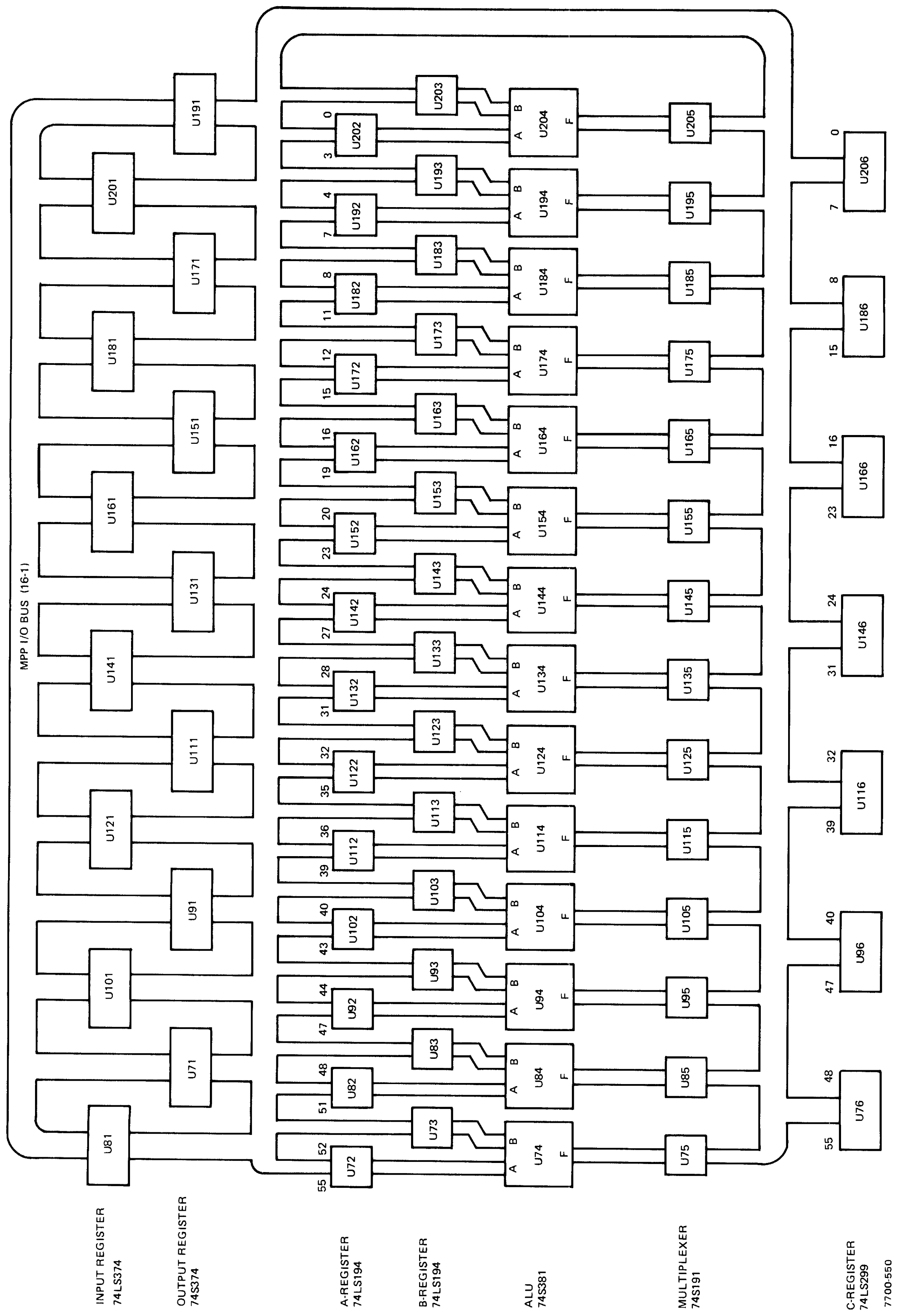
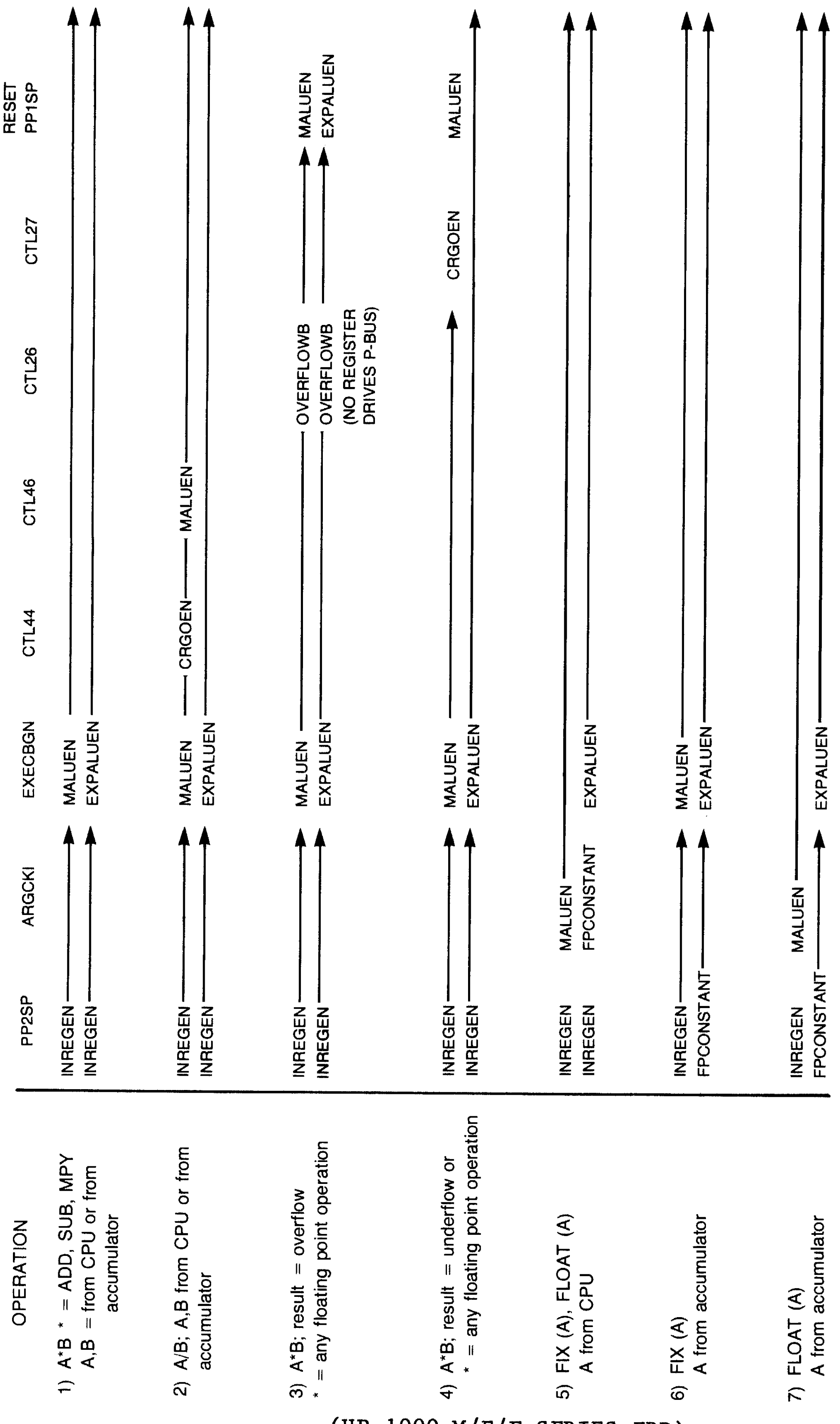


Figure III-2. Mantissa Registers

The least significant bits of the mantissa registers send information to the round information circuits. The round circuits consist of a guard bit register, two "ones" detect latches and multiplexers that feed the inputs to the guard bit register. For instance, a 74S151 8-line to 1-line multiplexer selects the precision dependent least significant bit of either AMAN or BMAN for the right shift input of the guard bit register. The guard bit register is a 74S194 four bit bidirectional universal shift register. During multiplication ALU cycles a 74S153 dual 4-line to 1-line multiplexer selects the two appropriate precision-dependent least significant bits of the outputs to the ALUS.

The P-bus was designed to be a tristate bus, so that several registers could drive it. For instance, the input register, ALU multiplexer outputs or the C register may drive the mantissa P-bus. Note that these registers all have tristate outputs. The mantissa P-bus drives the inputs of the output register, AMAN, BMAN and CMAN registers and the zero detect circuits. The zero detect logic consists of twelve 5-input NOR gates whose output tie to the inputs of a 13-input NAND gate. If the mantissa P-bus is all zeros, the output of the NAND gate is high. Each P-bus bit is tied to a 1000 ohm pull-up resistor, so that when no register drives the P-bus, the P-bus is all ones. The all ones condition is used to generate the overflow constant. The underflow sequence clears the C register and then activates its outputs, so that the C register drives all zeros onto the P-bus. Therefore, in sum, the P-bus at anytime either holds the contents of the input register, the contents of the ALU multiplexers, the contents of the C register which may be all zeros, or if no register is driving it, the P-bus is all ones (Figure III-3).



(HP 1000 M/E/F-SERIES ERD)

Figure III-3. P-Bus Mode Controls

Similarly, the exponent P-bus may driven by the input register, the ALUs or by a floating point constant set up by buffers. Since the ALUs (74S381) do not have tristate outputs, their outputs are routed to the P-bus through tristate buffers. In order to generate the underflow constant of all zeros, the underflow sequence sets the ALUs to the clear mode and enables the ALUs' output buffers to drive the P-bus. As the P-bus bits are tied high through 1000 ohm pull-up resistors, the P-bus is in the all ones state when is not driven. The overflow sequence disables all registers and buffers from driving the P-bus in order to create the overflow constant.

The exponent section contains five 74S85 four bit magnitude comparators. These comparators detect the swamp condition, exponents equal condition and exponent overflow or underflow condition.

The following summarizes the circuitry depicted on each page of the arithmetic PCA schematics part number D-12740-60001-51 through-56.

Schematics page 51:	mantissa section bits 55-40
page 52:	mantissa section bits 39-24
page 53:	mantissa section bits 23-8
page 54:	mantissa section bits 7-0, zero detect logic, round decision circuits
page 55:	exponent section
page 56:	look ahead carry generator circuits, multiplier look ahead registers

3.0 CONTROL PCA 12740-60002

The control PCA holds the state machine which dictates the sequence of operations that the arithmetic PCA performs. Besides the state machine, the control PCA contains the microprogrammable processor port (MPP) interface, the instruction register, P-bus mode control logic, register and ALU control logic and operation decision logic.

3.1 Execution Control

Control of execution is implemented through a 60 state sequential machine. This state machine consists of a chain of 60 serially connected clocked flip-flops in which one signal is passed along the chain activating various procedures and functions. Conceptually, the execution of an instruction passes through up to four control phases where each phase is comprised of several procedures and sub-operations. The four control phases are the loading sequence, the exec 1 group phase, the exec 2 phase and the termination sequence. The flow through the four phases varies for each instruction, and is combined in the flowchart in Figure III-4.

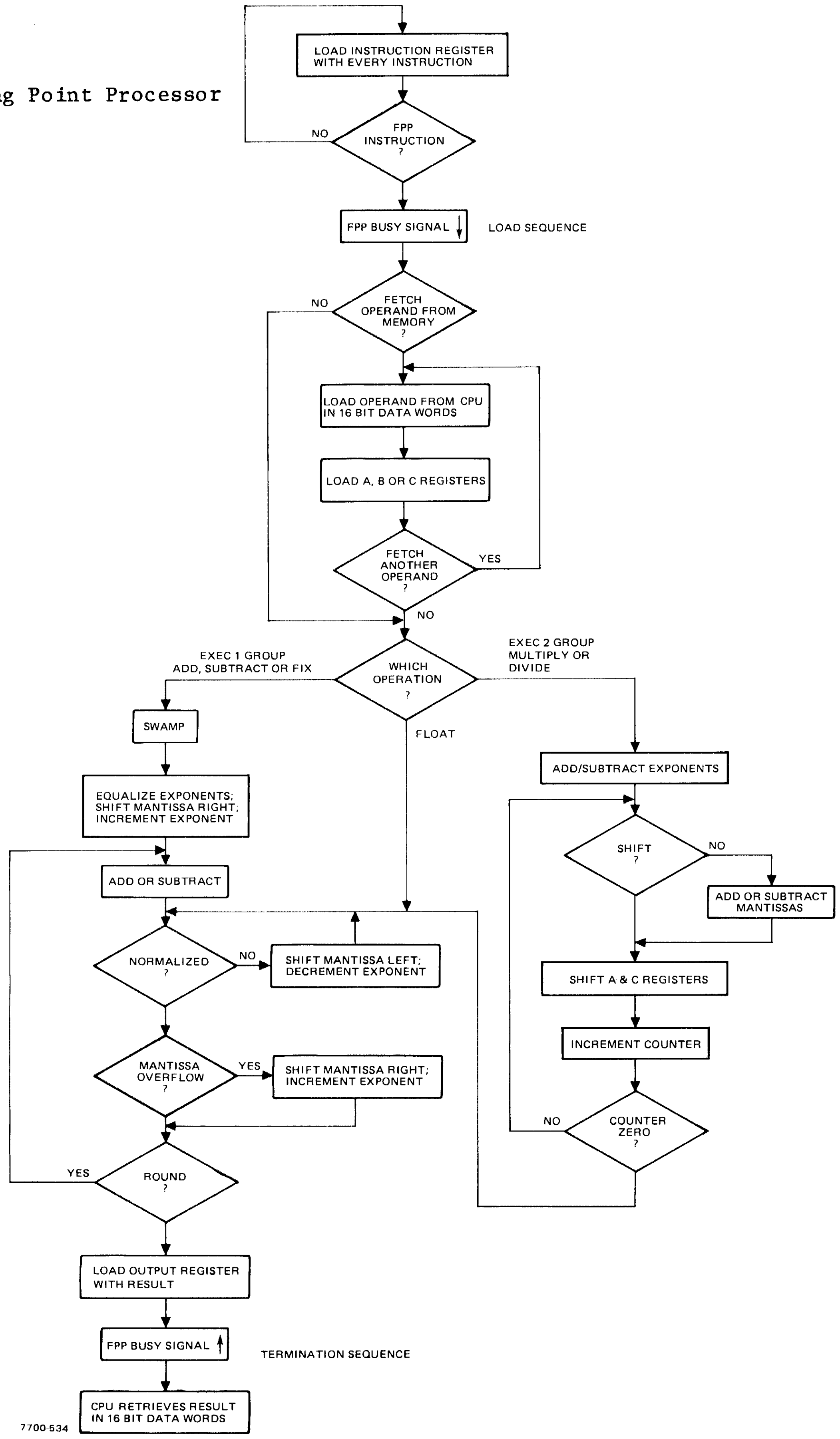

LOAD INSTRUCTION REGISTER WITH EVERY INSTRUCTION
FPP INSTRUCTION ?
NO
FPP BUSY SIGNAL ↓
LOAD SEQUENCE
FETCH OPERAND FROM MEMORY ?
NO
LOAD OPERAND FROM CPU IN 16 BIT DATA WORDS
LOAD A, B OR C REGISTERS
FETCH ANOTHER OPERAND ?
YES
NO
WHICH OPERATION ?
EXEC 1 GROUP ADD, SUBTRACT OR FIX
EXEC 2 GROUP MULTIPLY OR DIVIDE
FLOAT
SWAMP
EQUALIZE EXPONENTS; SHIFT MANTISSA RIGHT; INCREMENT EXPONENT
ADD OR SUBTRACT
NORMALIZED ?
NO
SHIFT MANTISSA LEFT; DECREMENT EXPONENT
MANTISSA OVERFLOW ?
YES
SHIFT MANTISSA RIGHT; INCREMENT EXPONENT
ROUND ?
YES
ADD/SUBTRACT EXPONENTS
SHIFT ?
NO
ADD OR SUBTRACT MANTISSAS
SHIFT A & C REGISTERS
INCREMENT COUNTER
COUNTER ZERO ?
NO
LOAD OUTPUT REGISTER WITH RESULT
FPP BUSY SIGNAL ↑
TERMINATION SEQUENCE
CPU RETRIEVES RESULT IN 16 BIT DATA WORDS
7700-534

Figure III-4. Execution Control Flowchart

3.1.1 Loading Sequence

The first phase, the loading sequence, which is show in Figure III-5, is initiated from the processor port signal PP2SP, which directs the FPP to execute the current instruction. The FPP proceeds to reset its control logic and assert the FPP busy (MPP) signal. The FPP may load from the CPU both or either of the FPP internal A and B registers or bypass the loading sequence entirely depending on IR(3,2). If IR(3,2) equals 11 designating an accumulator * accumulator operation, no operands from the CPU should be loaded, and control passes directly to the next phase of execution. Otherwise, if IR(3) is 0, the A register is loaded with the first operand from the computer CPU. Similarly, if IR(2) is 0, the B register is loaded from the CPU. IR(1,0) control the number of 16 bit words accepted by the input register in building an operand. Also, the FPP must detect the last word of an operand, since it contains the exponent which must be unpacked from the mantissa. Once IR(3,2) have been checked, and all registers loaded, the begin execution signal becomes active. Control then passes to the exec 1 group phase in the case of add, subtract, fix and float, or passes to the exec 2 group phase in the case of multiply and divide.

INSTRUCTION CODING

I_7 I_6 I_5 I_4	
0 0 0 0	ADD
0 0 0 1	SUBTRACT
0 0 1 0	MULTIPLY
0 0 1 1	DIVIDE
0 1 0 0	FIX
0 1 0 1	FLOAT

I_1 I_0	
0 0	32 BIT OPERATION
0 1	48
1 0	64
1 1	64 BIT -5 WORD FORMAT

I_3 I_2	
0 0	MEMORY*MEMORY
0 1	MEMORY*ACCUMULATOR
1 0	ACCUMULATOR*MEMORY
1 1	ACCUMULATOR*ACCUMULATOR

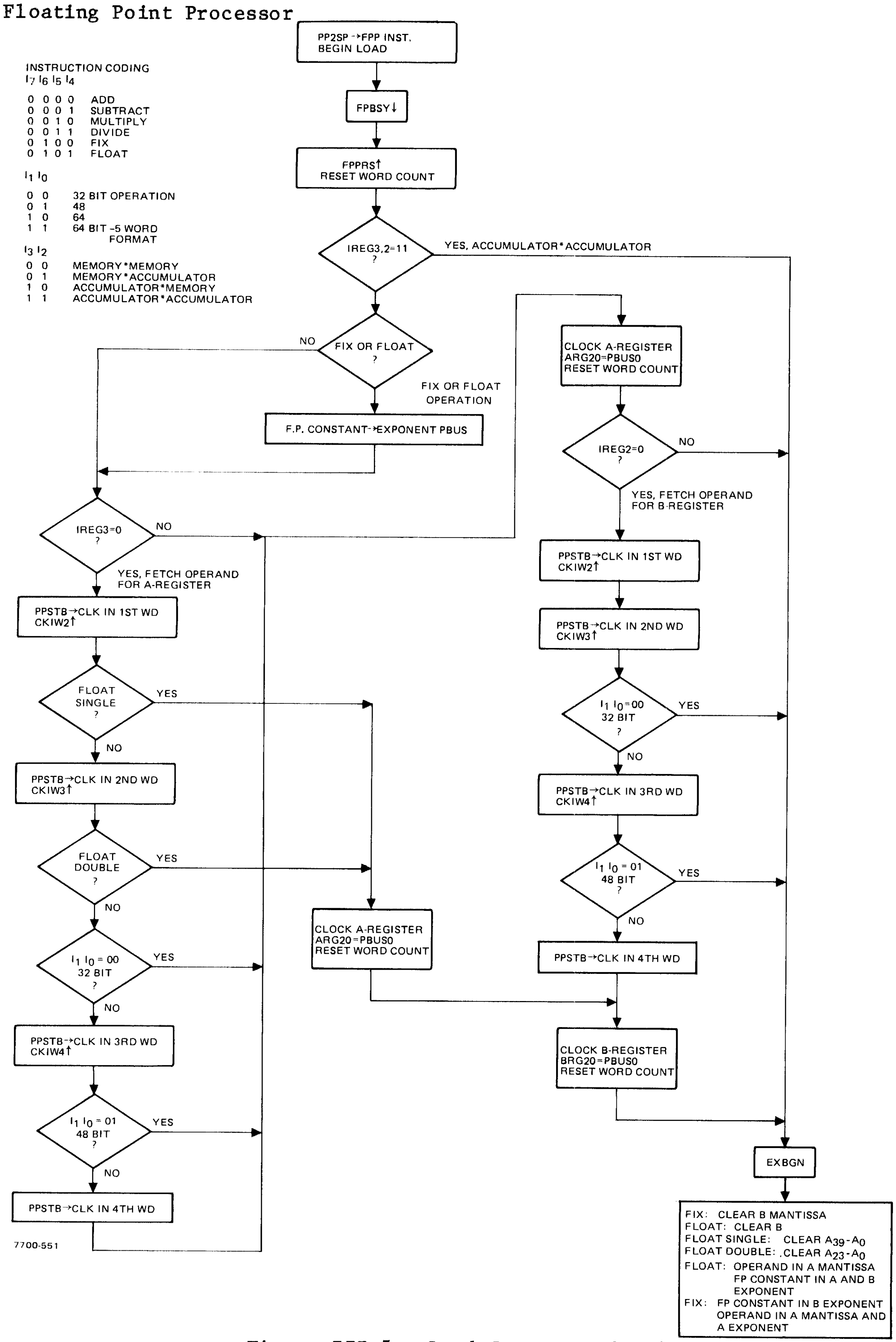

PP2SP→FPP INST. BEGIN LOAD
FPBSY↓
FPPRS↑ RESET WORD COUNT
IREG3,2=11 ?
YES, ACCUMULATOR*ACCUMULATOR
FIX OR FLOAT ?
NO
FIX OR FLOAT OPERATION
F.P. CONSTANT→EXPONENT PBUS
IREG3=0 ?
NO
YES, FETCH OPERAND FOR A-REGISTER
PPSTB→CLK IN 1ST WD CKIW2↑
FLOAT SINGLE ?
YES
NO
PPSTB→CLK IN 2ND WD CKIW3↑
FLOAT DOUBLE ?
YES
NO
I1 I0 = 00 32 BIT ?
YES
NO
PPSTB→CLK IN 3RD WD CKIW4↑
I1 I0 = 01 48 BIT ?
YES
NO
PPSTB→CLK IN 4TH WD
7700-551
CLOCK A-REGISTER ARG20=PBUSO RESET WORD COUNT
CLOCK A-REGISTER ARG20=PBUSO RESET WORD COUNT
IREG2=0 ?
NO
YES, FETCH OPERAND FOR B-REGISTER
PPSTB→CLK IN 1ST WD CKIW2↑
PPSTB→CLK IN 2ND WD CKIW3↑
I1 I0=00 32 BIT ?
YES
NO
PPSTB→CLK IN 3RD WD CKIW4↑
I1 I0 = 01 48 BIT ?
YES
NO
PPSTB→CLK IN 4TH WD
CLOCK B-REGISTER BRG20=PBUSO RESET WORD COUNT
EXBGN
FIX: CLEAR B MANTISSA
FLOAT: CLEAR B
FLOAT SINGLE: CLEAR A39-A0
FLOAT DOUBLE: CLEAR A23-A0
FLOAT: OPERAND IN A MANTISSA FP CONSTANT IN A AND B EXPONENT
FIX: FP CONSTANT IN B EXPONENT OPERAND IN A MANTISSA AND A EXPONENT

Figure III-5. Load Sequence Flowchart

Fix and float are two special cases to the loading sequence. Fix and float differ from the other instructions in that they operate on one operand, and both use a constant in the exponent register. For example, float loads the integer operand into the mantissa portion of the A register and then undergoes normalization while decrementing the constant in the exponent, in order to produce a floating point number. On the other hand, fix loads its floating point operand into the A mantissa and exponent registers and then equalizes A's exponent against the constant in BEXP, thereby generating an integer result. Thus, the loading sequence for fix and float loads an operand into A and a constant into the exponent before proceeding to the exec 1 group phase.

3.1.2 Execution Group One

The execution group one phase is a sequence of procedures, some of which are used by each operation. The procedures are swamp check, exponent equalization, ALU operation, normalization, round checking and exponent range checking. Add and subtract are the only operations that undergo all of these procedures. In fix, if the operand's exponent is positive, and the operand can be converted to an integer, the operand merely goes through the exponent equalization process of the exec 1 phase. Otherwise, if the fix single operand exceeds 32767 (maximum 16-bit integer), or the fix double operand exceeds 2,147,483,653 (maximum 32-bit integer) the maximum integer is returned and the overflow bit set.

In a float operation, the integer operand is merely normalized to become a floating point number. Thus, once a fix operand has been equalized and perhaps rounded, or a float operand is normalized, and the other operation results have completed the rounding check, control enters the exponent range check and the termination sequence.

3.1.3 Execution Group Two

The shift and add-or-subtract algorithms of multiplication and division are performed in the execution group two phase of execution. A counter, which is incremented on each shift or pass through the ALU, determines the end of this phase. For example, in a 32 bit divide the shift/ALU process is completed once 24 shifts and/or passes through the ALU have occurred. Once a product or a quotient have been formed, control passes to the normalization procedure of the exec group one phase, and then continues through to the termination sequence.

3.1.4 Termination

During the termination sequence, the final properly formed result is loaded into the output register as well as into all of the FPP internal registers. Thus, in a subsequent accumulator operation, the accumulator may be specified as either or both operands. Again, bits zero and one of the instruction register control the number of words returned to the CPU and are responsible for packing the exponent into the last word of the operand. At this point, the floating point processor lowers its FPP busy flag indicating that the computer may fetch the instruction result, or initiate a new operation. And so ends the execution of a floating point instruction.

SCHEMATIC SUMMARY III-I
LOAD CONTROL SEQUENCE

Schematic reference D-12740-60002-51 through-55

1. Load instruction register (U81) with every instruction while (IRST:PP5:FPPBUSY) is true
   activate INSTRCK+ (U31-6 at 12-A)

2. PP2SP indicates FPP will perform operation if (PP2SP:LBIT 0 ADDRESS:FPPBUSY- high), activate FPP operations (U171-3 at 12-B)

   (a) Set FPPBUSY flip-flop U155-3,4,5,6; U112-1,2,3 at 13-A) to busy state to lock up instruction register; note U30-1,2 held low.

   (b) Activate FPPRS+ (U181-6, at 13-A):
   Clear round control (U124-4 at 25-C)

   (c) Activate PWRST- (U182-3 at 13-A):
   Disable swamp delay control state 1 (U204-4 at 21-B)
   Disable equalize control state 1(U184-4 at 22-B)
   Disable swamp control state 4 (U194-4 at 22-C)
   Disable normalize preparation control state 2 (U164-4 at 24-B)
   Disable normalization control state 3 (U144-4 at 24-C)
   Disable round decision control state 1 (U154-4 at 25-B)
   Set the equalize A/B register to EQUALA+ (U82-4 at 41-B)
   Set the mantissa ALU mode S0 flip-flop to Add - S0 high
   Disable the C register output enable (U176-1,2,12,13; U166-4,5,6 at 52-C)

   (d) Activate PWRST2 - (U182-5 at 14-A):
   Set load A/B to load A (U141-10 at 14-B)
   Disable execution begin state (U141-4 at 15-A)
   Enable fix round flip-flop (U74-4,5,6,8,9,10 at 26-C)
   Activate CLEAR ROUND- (P3-19, U84-8 at 26-C)
   ALU board actions:
   Disable one's detect flipflop (U87-4,5,6,8,9,10 at 44-C)
   Disable sticky bit (U77-1,2,3,4,5,6 at 44-C)
   Disable complement carry-in flip-flop (U52-4,5,6,11,12,13, at 43-D)
   Clear guard 3 save register (U106-4 at 44-C)
   Clear guard bits register (U63 at 43-C)
   Disable MDY/DIV initializaion register (U30-10 at 31-A)
   Set counter load flipflop to load (U41-8,9,10,11; U83-8,9,10 at 32-A)
   Disable MPY/DIV shift control state 1 (U52-1 at 32-B)

Disable mantissa overflow detect register (U13-4-at 32-D)
Disable MPY/DIV shift control state 2 (U52-10 at 32-B)
Disable MPY/DIV ALU control state 5 (U32-4 at 34-B)
Set EXPALUS0 (U15-8 at 42-D) P1-10 low
Set EXPALUS1 (U16-3 at 42-D) P1-13 high
Set EXPALUS2 (U55-8 at 42-D) P1-15 low
Set exponent registers to load mode: EXDLD- low (P1-4, U26-6 at 44-D)
Set exponent registers to count up mode: EXPCNTUP+ high (P1-6,U36-6 at 44-C)
Set ARGS0 high (P1-42, U45-9, 44-B)
Set ABRGS1 high (P1-44, U35-5 at 44-A)
Set CRGS0 high (P1-46, U56-12 at 44-C)
Set CRGS1 high (P1-48, U56-8 at 44-C)

(e) Activate PP2RS+ (U161-8 at 13-B):
If IREG3:FIXORFLOAT+, activate INREGEN (U166-3 at 52-D)
Enable EXPINCK (U146-1 at 14-C)
Enable input word clocks (U135-2,5,10,13; U206-10 at 14-C)
If IRG00:IRG01, enable EXPUINEN- (P1-18, U196-6 at 12-D)
Disable MALUEN+ (P3-39, U205-4 at 53-C)
Disable EXPALUEN-(P3-23, U176-8 at 53-D)

(f) Activate PP2RS2- (U112-11 at 13-B):
Reset word count register (U145-1 at 12-C)

(g) If (IREG2:IREG3:FIXORFLT-), then goto execution begin

3. MPBST - store data currently on MPPBUS

(a) Clock word count register (U145-9 at 13C);
Activate a word input clock

(b) If last word, transfer operand from input register to A or B registers.

i. Clear word count register (U145-1 at 13-C)

ii. If load A/B has A enabled (U141-9 at 14-B HIGH), load A register via ARGCKI (U131-8,9 at 14-B).
ARGCKI;
If (PBUS55 XOR PBUS54)LOW, ARGZERO+ (A=0) activated (U152-8 at 15-B)
If FIXFLOAT+ disable INREGEN
enable FPCONSTANT+, FPCONSTANT-
Toggle load A/B register to enable B (U141-8 at 14-B high)
If (IREG2:FIXORFLT-), go to execution begin control state (U144-1,2,3,4,5,6 at 15-B)

```
iii. If load A/B register has B enabled (U141-8 at
       14-B HIGH)
     Load B registers, activate BRGCKI (U121-8,9 at
       14-B)
     BRGCKI:  If (PBUS55 XOR PBUS54) low, activate
       BRGZERO+ (U152-1,2,3,4,5,6 at 15-B) to indicate
        B=0
     Load exponent into BEXP
     If FIXORFLOAT-, load operand into BMAN
     go to execution begin control state
       (U141-1,2,3,4,5,6 at 15-B)
```

3.2 MPP Interface and FPP Instruction Register (Figures III-6 and III-7)

The floating point processor communicates to the CPU across the microprogrammable processor port (MPP or microport). The microport operations are directed by CPU microcode. The microcode may reside in four places: CPU base set ROMs, user control store ROMs on the Firmware Accessory Board, Firmware Expansion Module, or the Writeable Control Store Board. The microport consists of sixteen bidirectional tristate data lines and nine control signals. The nine control signals include an address bit (PLR0), a synchronization timing signal (PP5), five control signals driven by the CPU (PIRST, PP2SP, MPBST, MPBEN and PP1SP), and two tristate signals driven by external processors (NSTOV and MPPCNDX) which in this case would be the floating point processor.

4.0 IMPLEMENTATION OF ADDITION AND SUBTRACTION

Addition and subtraction are in the execution group one type as shown in Figure III-8. By the completion of the load operand sequence the A registers hold the augend in addition or minuend in subtraction. The B registers hold the addend in addition and the subtrahend in subtraction. Addition and subtraction undergo swamp check, exponent equalization, ALU cycle, mantissa overflow check and rounding in the execution group one sequence. After that sequence they proceed to the termination section. Table III-1 summarizes the addition and subtraction sequence and indicates the control states that are active at each step.

5.0 IMPLEMENTATION OF FIX TO SINGLE/DOUBLE INTEGER

The third operation within the execution group one is fix to single or double integer. At the end of the loading sequence, the floating point operand is in the A-register and a floating point constant is in the B exponent (refer to para. 3.1.1). This operation flows through a subset of the execution group one control states. Table III-2 lists the operation sequences that fix undergoes and the signals which control these sequences (Figure III-9).

THE MPP AND FPP ARE CONNECTED BY A 3M 50-WIRE RIBBON CABLE 18 INCHES LONG.

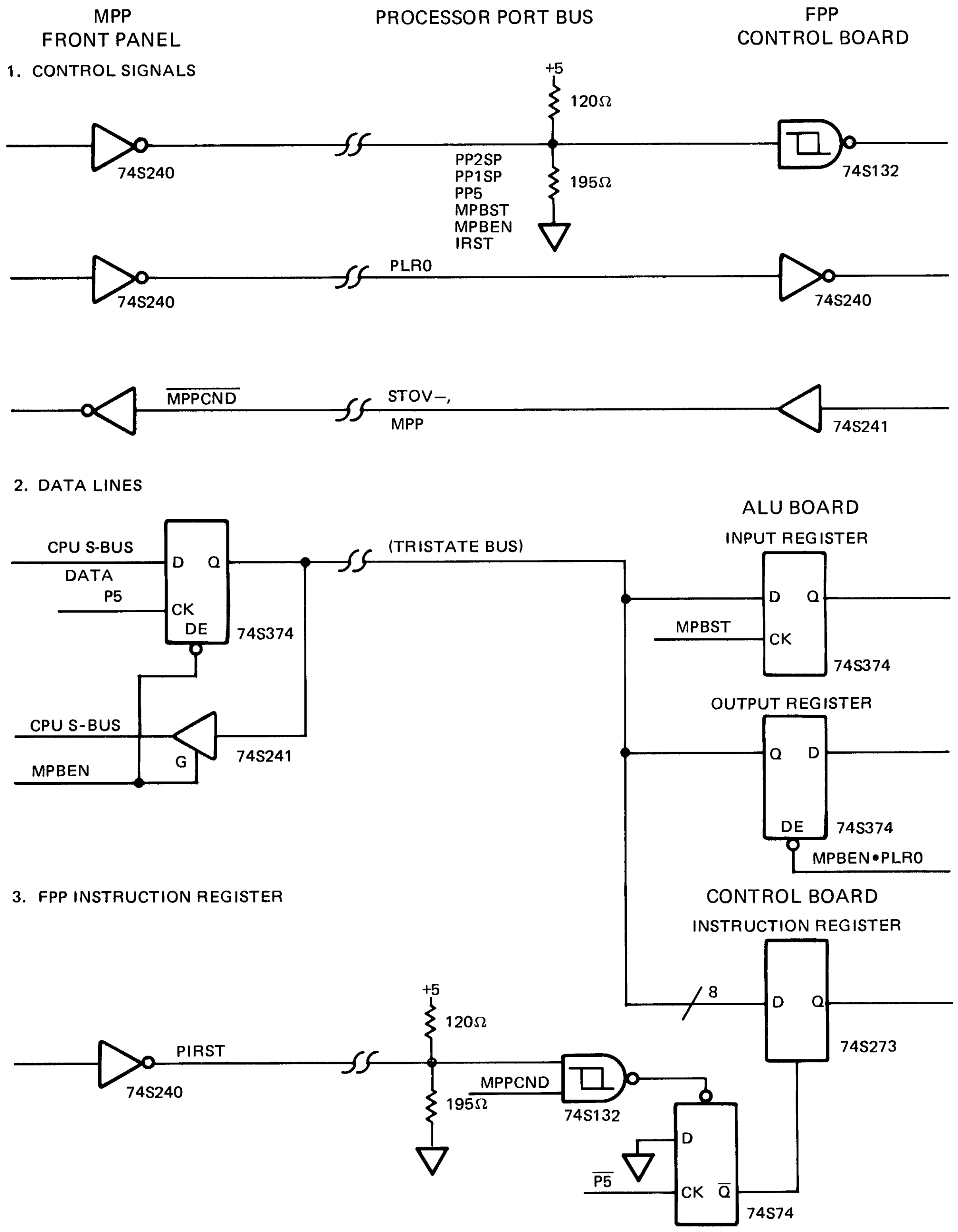


Figure III-6. MPP to FPP Simplified Circuit Diagram

(HP 1000 M/E/F-SERIES ERD)

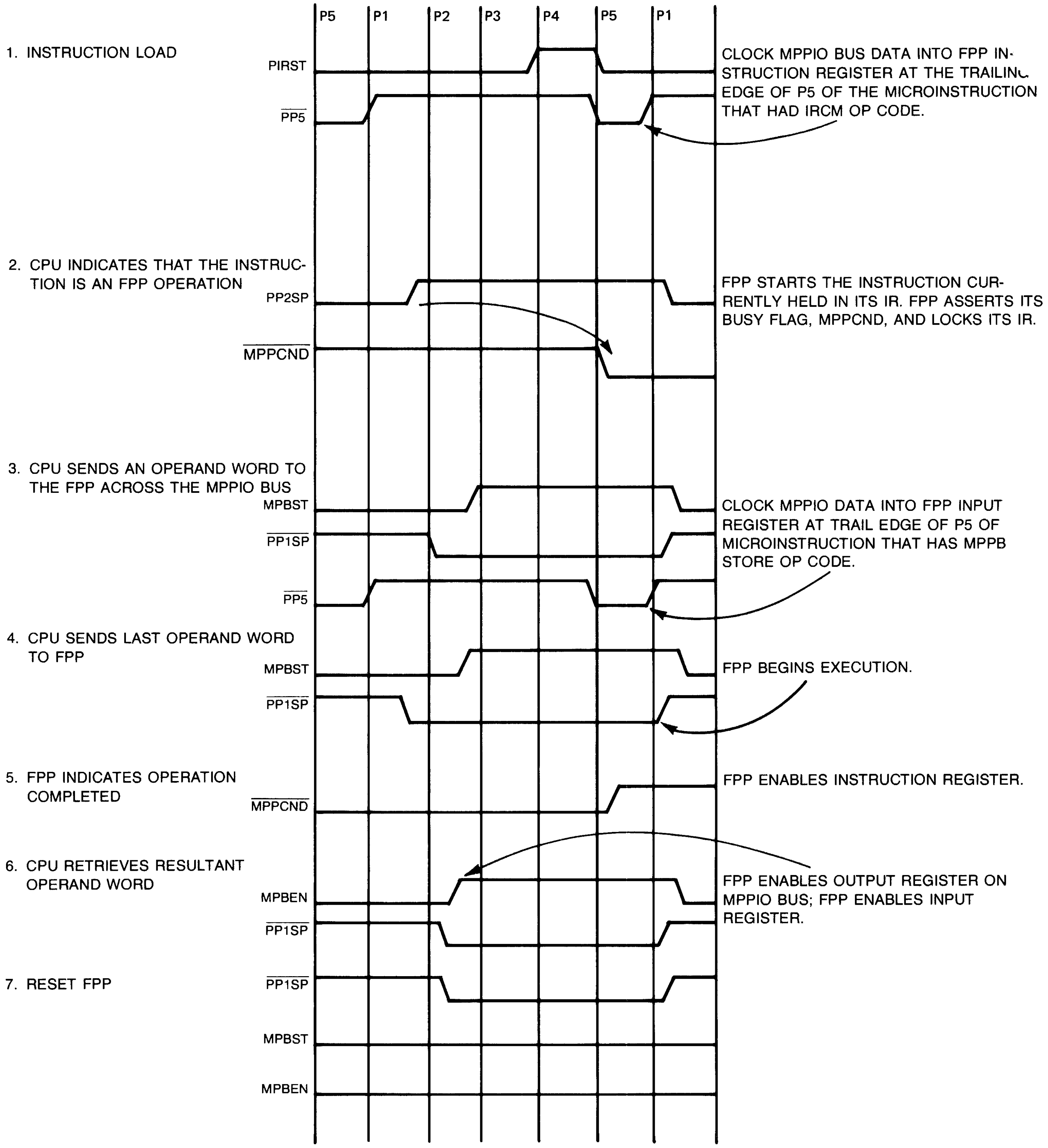

P5
P1
P2
P3
P4
P5
P1
1. INSTRUCTION LOAD
PIRST
PP5
CLOCK MPPIO BUS DATA INTO FPP INSTRUCTION REGISTER AT THE TRAILING EDGE OF P5 OF THE MICROINSTRUCTION THAT HAD IRCM OP CODE.
2. CPU INDICATES THAT THE INSTRUCTION IS AN FPP OPERATION
PP2SP
MPPCND
FPP STARTS THE INSTRUCTION CURRENTLY HELD IN ITS IR. FPP ASSERTS ITS BUSY FLAG, MPPCND, AND LOCKS ITS IR.
3. CPU SENDS AN OPERAND WORD TO THE FPP ACROSS THE MPPIO BUS
MPBST
PP1SP
PP5
CLOCK MPPIO DATA INTO FPP INPUT REGISTER AT TRAIL EDGE OF P5 OF MICROINSTRUCTION THAT HAS MPPB STORE OP CODE.
4. CPU SENDS LAST OPERAND WORD TO FPP
MPBST
PP1SP
FPP BEGINS EXECUTION.
5. FPP INDICATES OPERATION COMPLETED
MPPCND
FPP ENABLES INSTRUCTION REGISTER.
6. CPU RETRIEVES RESULTANT OPERAND WORD
MPBEN
PP1SP
FPP ENABLES OUTPUT REGISTER ON MPPIO BUS; FPP ENABLES INPUT REGISTER.
7. RESET FPP
PP1SP
MPBST
MPBEN

Figure III-7. FPP Communication Timing Diagram

(HP 1000 M/E/F-SERIES ERD)

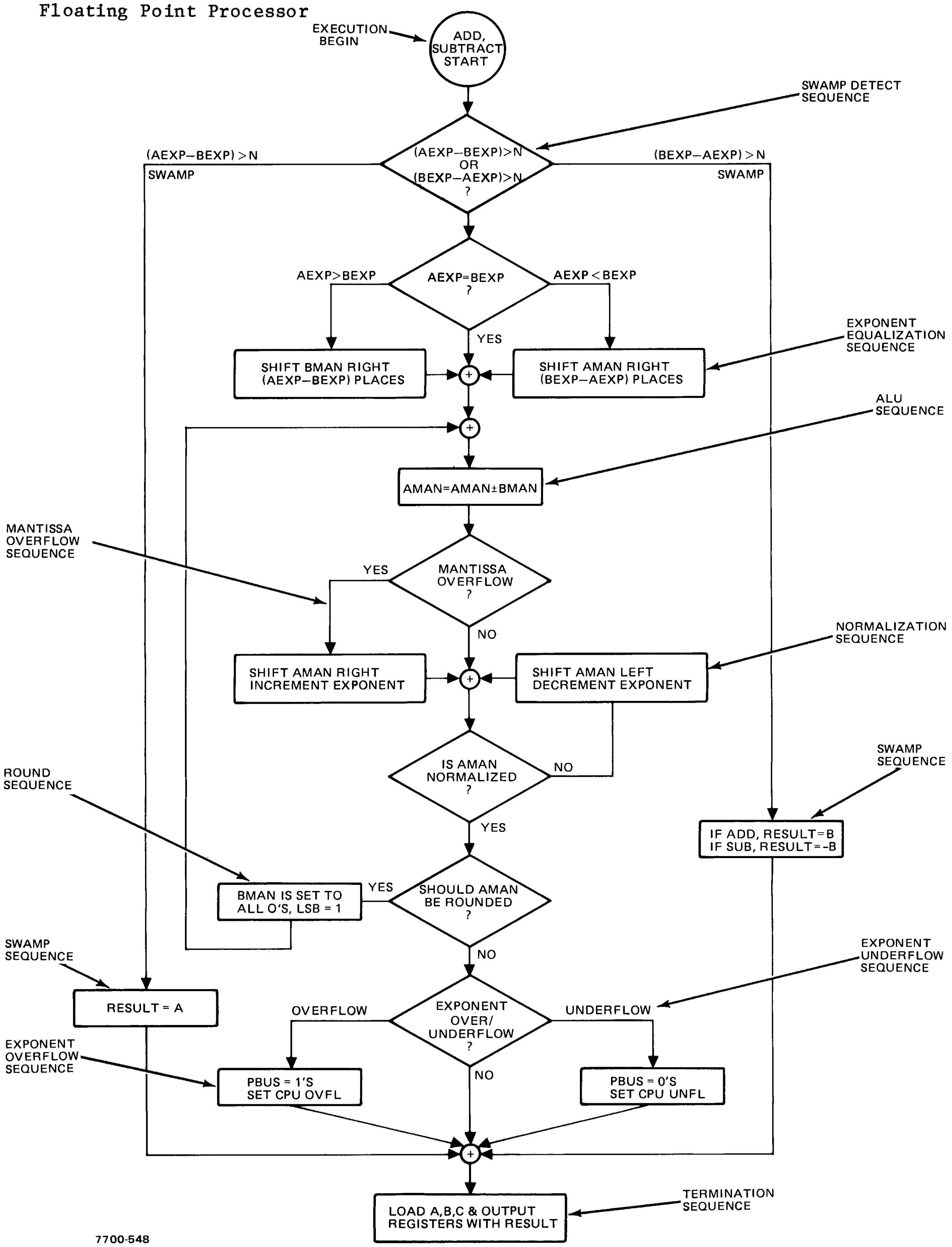

EXECUTION BEGIN
ADD, SUBTRACT START
SWAMP DETECT SEQUENCE
(AEXP–BEXP)>N SWAMP
(AEXP–BEXP)>N OR (BEXP–AEXP)>N ?
(BEXP–AEXP)>N SWAMP
AEXP>BEXP
AEXP=BEXP ?
AEXP<BEXP
YES
EXPONENT EQUALIZATION SEQUENCE
SHIFT BMAN RIGHT (AEXP–BEXP) PLACES
SHIFT AMAN RIGHT (BEXP–AEXP) PLACES
ALU SEQUENCE
AMAN=AMAN±BMAN
MANTISSA OVERFLOW SEQUENCE
YES
MANTISSA OVERFLOW ?
NO
NORMALIZATION SEQUENCE
SHIFT AMAN RIGHT INCREMENT EXPONENT
SHIFT AMAN LEFT DECREMENT EXPONENT
IS AMAN NORMALIZED ?
NO
YES
SWAMP SEQUENCE
IF ADD, RESULT=B IF SUB, RESULT=-B
ROUND SEQUENCE
BMAN IS SET TO ALL O'S, LSB = 1
YES
SHOULD AMAN BE ROUNDED ?
NO
SWAMP SEQUENCE
RESULT = A
EXPONENT UNDERFLOW SEQUENCE
OVERFLOW
EXPONENT OVER/ UNDERFLOW ?
UNDERFLOW
EXPONENT OVERFLOW SEQUENCE
PBUS = 1'S SET CPU OVFL
NO
PBUS = 0'S SET CPU UNFL
LOAD A,B,C & OUTPUT REGISTERS WITH RESULT
TERMINATION SEQUENCE
7700-548

Figure III-8. Addition and Subtraction Sequence Flowchart

Table III-1. State Machine Sequence for Add/Subtract

ACTION	CONTROLLING SIGNALS	CONTROL STATES	12740-60002 SCHEMATIC REFERENCE
1. Load operands into A,B registers	ARGCKI, BRGCKI	Operand load sequence	Page 51
2. Check for swamp condition	EXEC1 +	SWPDELAY(1,2,3,4)	Page 52-21-B
2a. If swamp, go through swamp sequence	SWAMP + (high if swamp condition exists)	SWAMP (1,2,3,4)	Page 52-22-C
3. If exponents differ, equalize exponents	EXPEQUAL – (low if exponents are equal)	EQUALCTL (1,2,3)	Page 52-22-B
4. Add/subtract mantissas	EXPEQUAL +.FIXOP	ALUDELAY (1,2,3,4,5)	Page 52-23-B
5. If mantissa overflow, correct mantissa overflow	MOVFL + (high if mantissa overflow exists)	MOVFCLT (1,2,3,4)	Page 52-24-B
6. If result not normalized, normalize it	ANORMLZD + (high if AMAN is not normalized)	NORMPREP (1,2), NORMCTL (1,2,3)	Page 52-24-B and C
7. If result needs rounding, round it	ANORMLZE +, MOVFL –, ROUND –, ROUNDEN +	ROUND DECISION (1,2) ROUND CONTROL, ALUDELAY (1,2,3,4,5)	Page 52-25-B Page 52-25-C,-23-B
7a. If rounding caused mantissa overflow, correct it	MOVFL + (high if mantissa overflow exists)	MOVFLCTL (1,2,3,4)	Page 52-24-B
8. If exponent overflow or underflow occurred, result equals over/underflow constant	EXPOVUFN – (low if exponents OK), EXPOVFL + (high if overflow), EXPUNFL + (high if underflow)	OVER/UNDERFLOW DECISION OVERFLOW UNDERFLOW	Page 52-25-B Page 52-25-A Page 52-25-D
9. Load final result in all registers	TERMCTL4 – (goes to clock of all registers)	TERMCTL (1,2,3,4)	Page 52-26-B

Table III-2. State Machine Sequence for Fix to Single/Double Integer

ACTION	CONTROLLING SIGNAL	ACTIVE CONTROL STATES	12740-60002 SCHEMATIC REFERENCE
1. Load operand into A register	ARGCKI	Operand load sequence	Page 51
2. Load Fix constant (15/31) into BEXP	BRGCKI	Operand load sequence	Page 51
3. Go through swamp delay sequence	EXEC1 +	SWPDELAY (1,2,3,4)	Page 52-21-B
4. If AEXP<0, go to underflow	FIXOP +•EXPSIGN(sign of AEXP, high if AEXP <0)	UNDERFLOW	Page 52-25-D
5. If AEXP>BEXP, go to overflow	FIXOP +.$\overline{\text{AGTB}}$ –(high if AEXP> BEXHP)	OVERFLOW	Page 52-25-A
6. If AEXP≠BEXP, equalize exponents	$\overline{\text{EXPEQUAL}}$ –(high if exponents differ)	EQUALCTL (1,2,3)	Page 52-22-B
7. If results needs rounding, round it	FIXSG0 – high if P bus (39-0) is not all zeros FIXDB0 – high if P bus (23-0) is not all zeros	ROUND DECISION ROUND CONTROL, ALUDELAY (1,2,3,4,5)	Page 52-25-B Page 52-25-C Page 52-23-B
8. Load final result in all registers	TERMCTL4 –(goes to clock of all registers)	TERMCTL (1,2,3,4)	Page 52-26-B

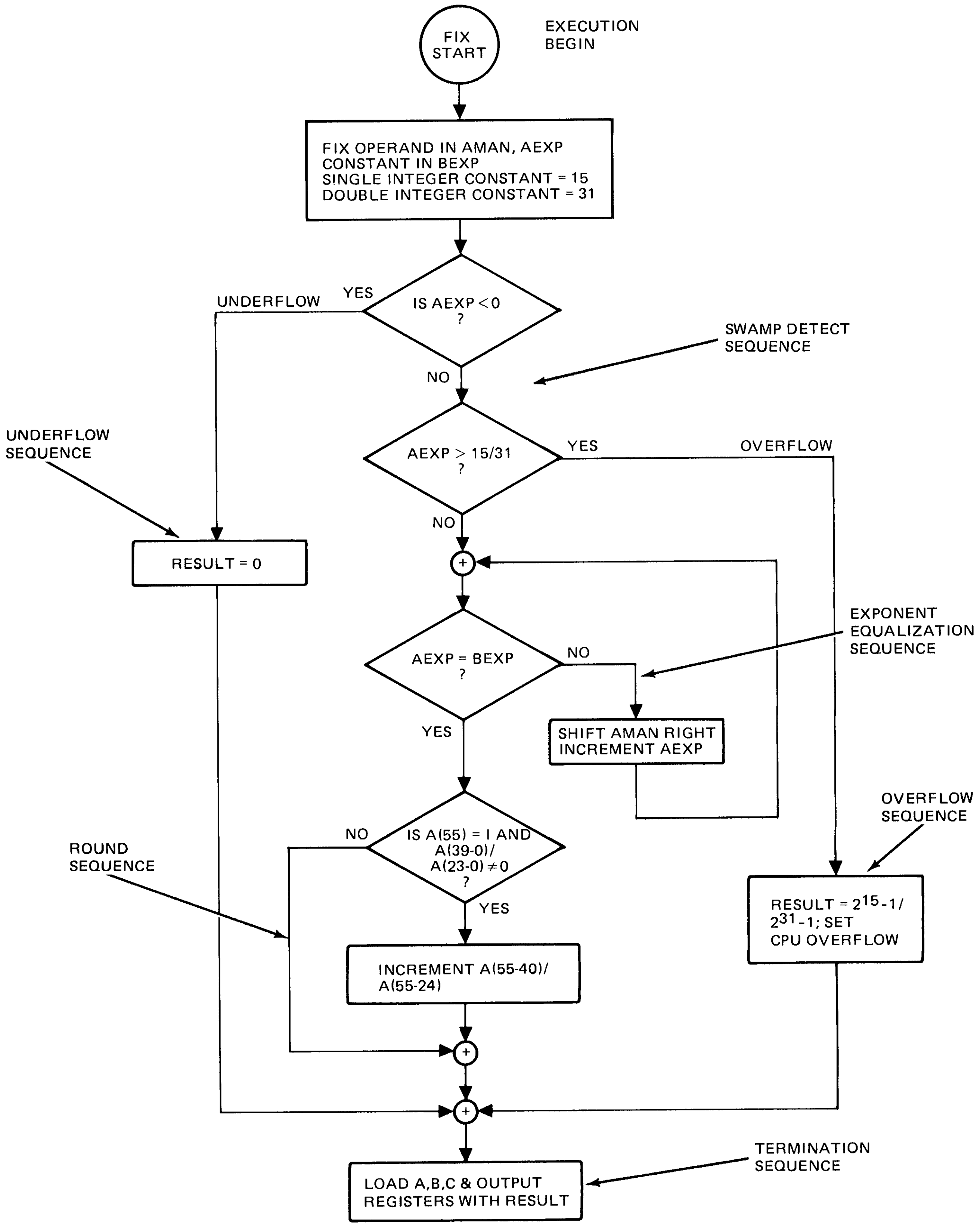

FIX START
EXECUTION BEGIN
FIX OPERAND IN AMAN, AEXP
CONSTANT IN BEXP
SINGLE INTEGER CONSTANT = 15
DOUBLE INTEGER CONSTANT = 31
IS AEXP < 0 ?
YES
UNDERFLOW
NO
SWAMP DETECT SEQUENCE
AEXP > 15/31 ?
YES
OVERFLOW
NO
UNDERFLOW SEQUENCE
RESULT = 0
AEXP = BEXP ?
NO
YES
EXPONENT EQUALIZATION SEQUENCE
SHIFT AMAN RIGHT
INCREMENT AEXP
OVERFLOW SEQUENCE
ROUND SEQUENCE
IS A(55) = I AND A(39-0)/ A(23-0) ≠ 0 ?
NO
YES
INCREMENT A(55-40)/ A(55-24)
RESULT = 2^15-1/ 2^31-1; SET CPU OVERFLOW
TERMINATION SEQUENCE
LOAD A,B,C & OUTPUT REGISTERS WITH RESULT

Figure III-9. Fix to Single/Double Integer Sequence Flowchart

(HP 1000 M/E/F-SERIES ERD)

5.1 Fix/Float Constant

In fix instructions, the floating point operand is equalized against a constant in BEXP. In float instructions the integer in AMAN and constant in AEXP and BEXP undergo normalization. During the loading sequence the fix or float constant has to be loaded into AEXP and BEXP, even if an accumulator operation (IR(3) equals 1) is specified. Thus, the constant is enabled on the exponent P-bus from the time of PP2RS+ if the operand is in the accumulator, or from ARGCKI+ if the operand comes from the CPU. The loading sequence for float always includes the sequence for BRGCKI which loads the constant into BEXP. The constant is loaded into AEXP at the execution begin contol state.

6.0 IMPLEMENTATION OF FLOAT FROM SINGLE/DOUBLE INTEGER (Figure III-10)

Since the float from single or double integer is the simplest of all of FPP's operations, it is not included in the execution group one or group two operations. During the loading sequence the integer operand is loaded into AMAN and constant is loaded into AEXP and BEXP.

From the execution begin state control passes to the normalization sequence, unless AMAN is already normalized. AMAN is shifted left while AEXP and BEXP are decremented until the contents of AMAN are normalized (AMAN(55) does not equal AMAN(54)). At this point or if AMAN originally was normalized, control passes to the round decision state, exponent overflow or underflow decision state and on to the termination sequence. Note that float performs no rounding. Also, exponent overflow or underflow can not occur during float since the input operand is a sixteen bit or thirty-two bit integer. Thus, float effectively goes from normalization to termination.

The state machine called EXEC I, depicted on schematic 12740-60002 page 52, fully controls the sequence of events for addition, subtraction, fix and float. Figure III-11 combines the flow of these operations in one flowchart.

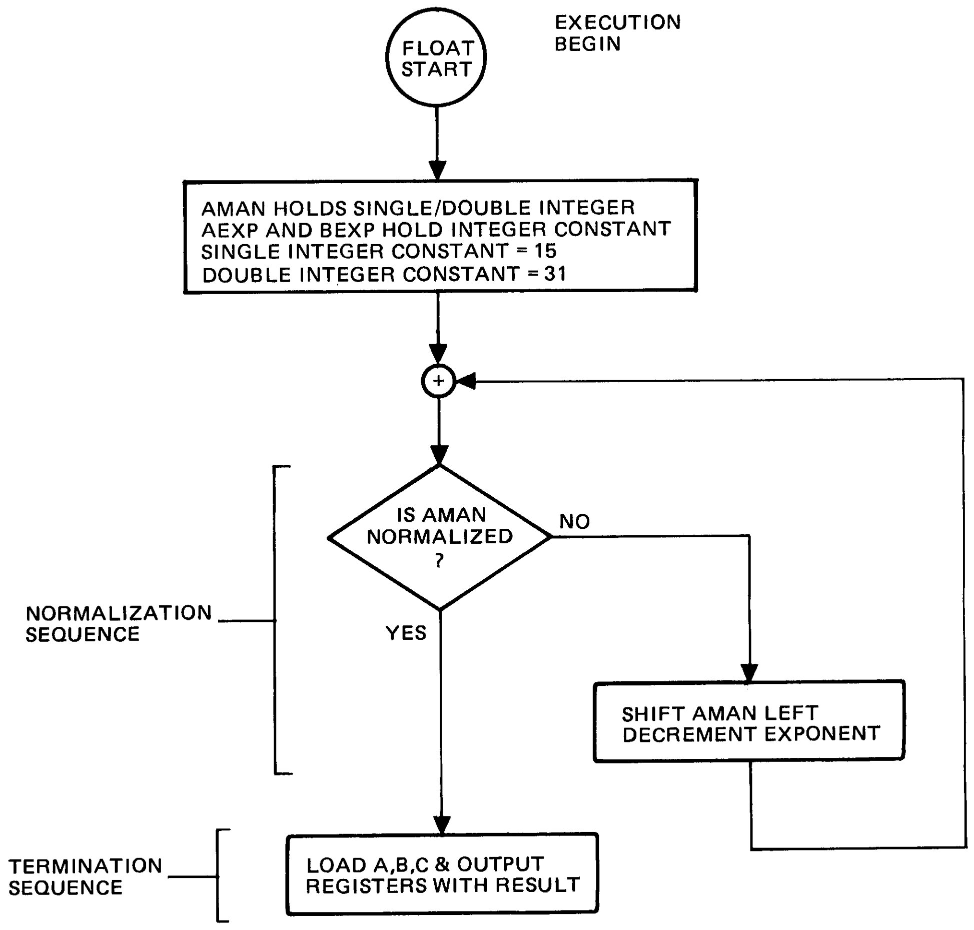

FLOAT START
EXECUTION BEGIN
AMAN HOLDS SINGLE/DOUBLE INTEGER
AEXP AND BEXP HOLD INTEGER CONSTANT
SINGLE INTEGER CONSTANT = 15
DOUBLE INTEGER CONSTANT = 31
+
IS AMAN NORMALIZED ?
NO
YES
NORMALIZATION SEQUENCE
SHIFT AMAN LEFT
DECREMENT EXPONENT
TERMINATION SEQUENCE
LOAD A,B,C & OUTPUT REGISTERS WITH RESULT

Figure III-10. Float From Single/Double Integer Flowchart

(HP 1000 M/E/F-SERIES ERD)

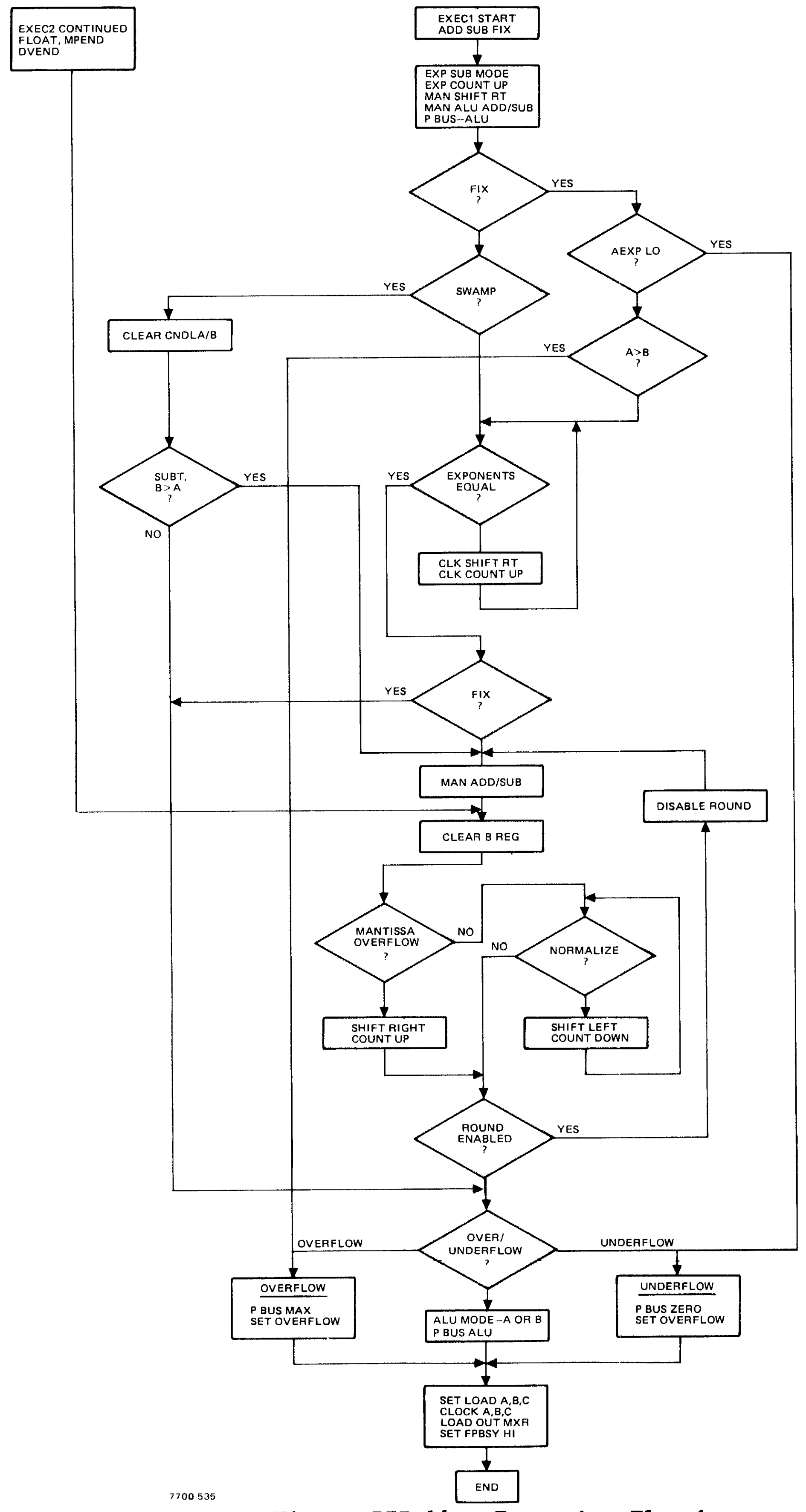


Figure III-11. Execution Flowchart

SCHEMATIC SUMMARY III-2 (Figure III-12)

I. Execution Begin to Termination for ADD/SUB/FIX

Mode control states prior to Execution Begin control state:
CRGOEN+ low
MNTSO high (add AMAN to BMAN)
ARGSO high and ABRGS1 high - AMAN is in load condition
CRGSO high and CRGS1 high - C-register is in load condition
EXPCNTUP+ high - count up
EXPLD- low
EXPALUSO- low, EXPALUS1- low, and EXPALUS2- low causes AEXP-BEXP
INREGEN+ high:

1. input register drives P-bus
2. A mantissa, B mantissa, AEXP, BEXP, C mantissa, registers all set to load condition
3. mantissa ALU set to add, exponent ALU set to subtract
4. exponent registers set to count up

II. Execution Begin (ADD, SUB, or FIX)

MALUEN+ high mantissa ALU drives P-bus
INREGEN+ input register enable low
FPCONSTANT- fix/float constant enable high
EXPALUEN- low exponent ALU drives P-bus
Thus, ALU's drive the P-bus

III. EXEC1 State Machine

```
SWPDELAY (1,2,3,4) - Swamp detect delay
         SWPDELAY(1):  ABRGS1 (A/B mantissa register S1) low
                              shift right mode
                       EXPLD- (exponent load) high
If subtract: MNTSO  (mantissa ALU mode SO) low, (AMAN-BMAN) on
                    P-bus
             SWPDELAY (2,3,4) provide delay

             SWPDELAY4 actions:
             clock AGTBZERO:(A>B or B=0)
              -----          --- ----    --------
If SWPDELAY4 (SWAMP-:ABZERO-):FIX(AGTB- OR EXPSIGN-):EXPEQUAL-
             then go to EQUALCTL(1)
If SWPDELAY4 :FIX:AGTB, then go to OVERFLOW
If SWPDELAY4 :FIX:EXPEQUAL, then go to ROUND DECISION
If SWPDELAY4 :EXPEQUAL:FIXOP:ABZERO-, then go to ALU CYCLE
If SWPDELAY4 :(SWAMP or ABZERO+):FIX, then go to SWAMP
If SWPDELAY4 :AGTBZERO:FPSUB, then set SWAMPSBGTA+
```

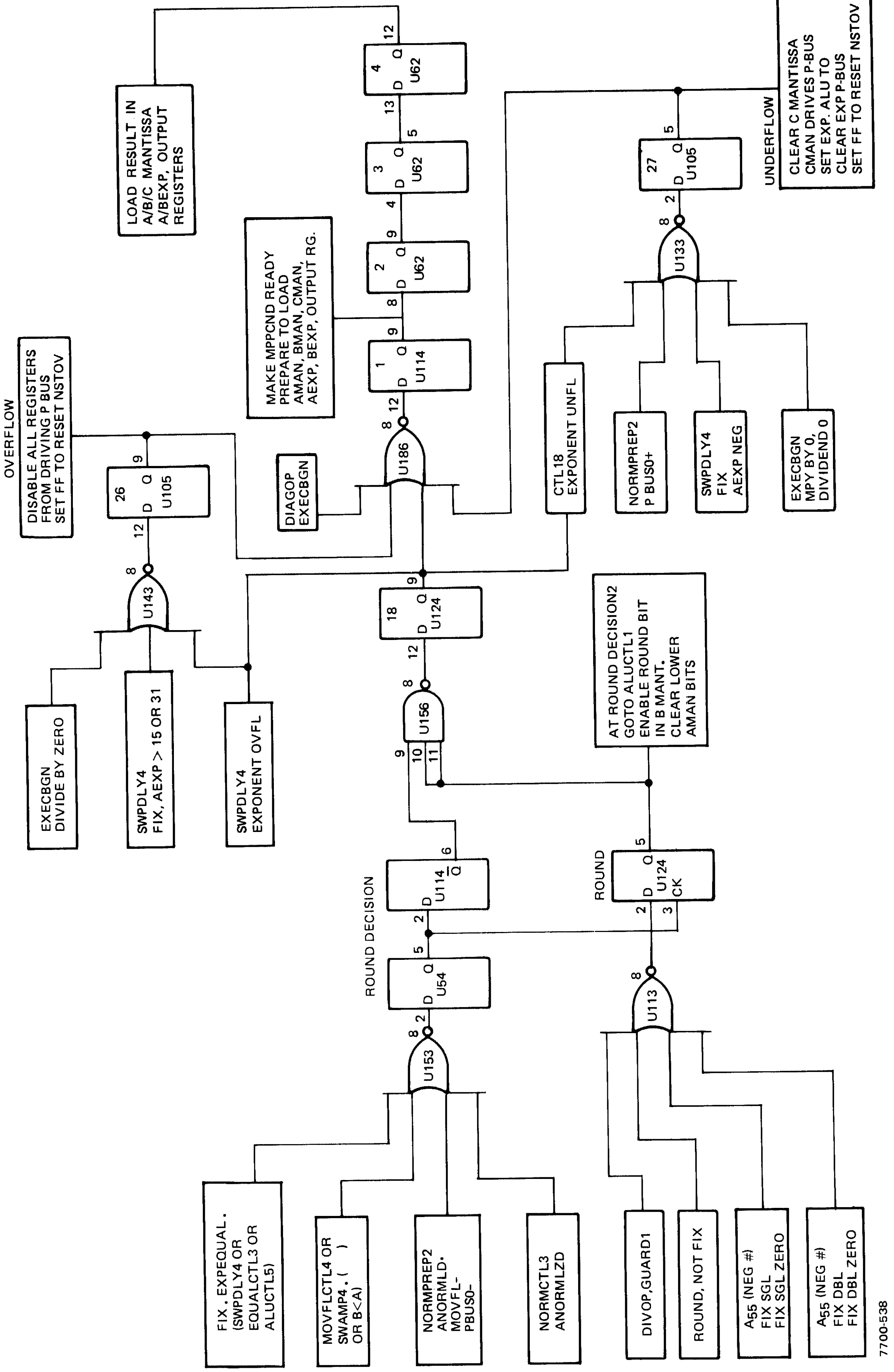
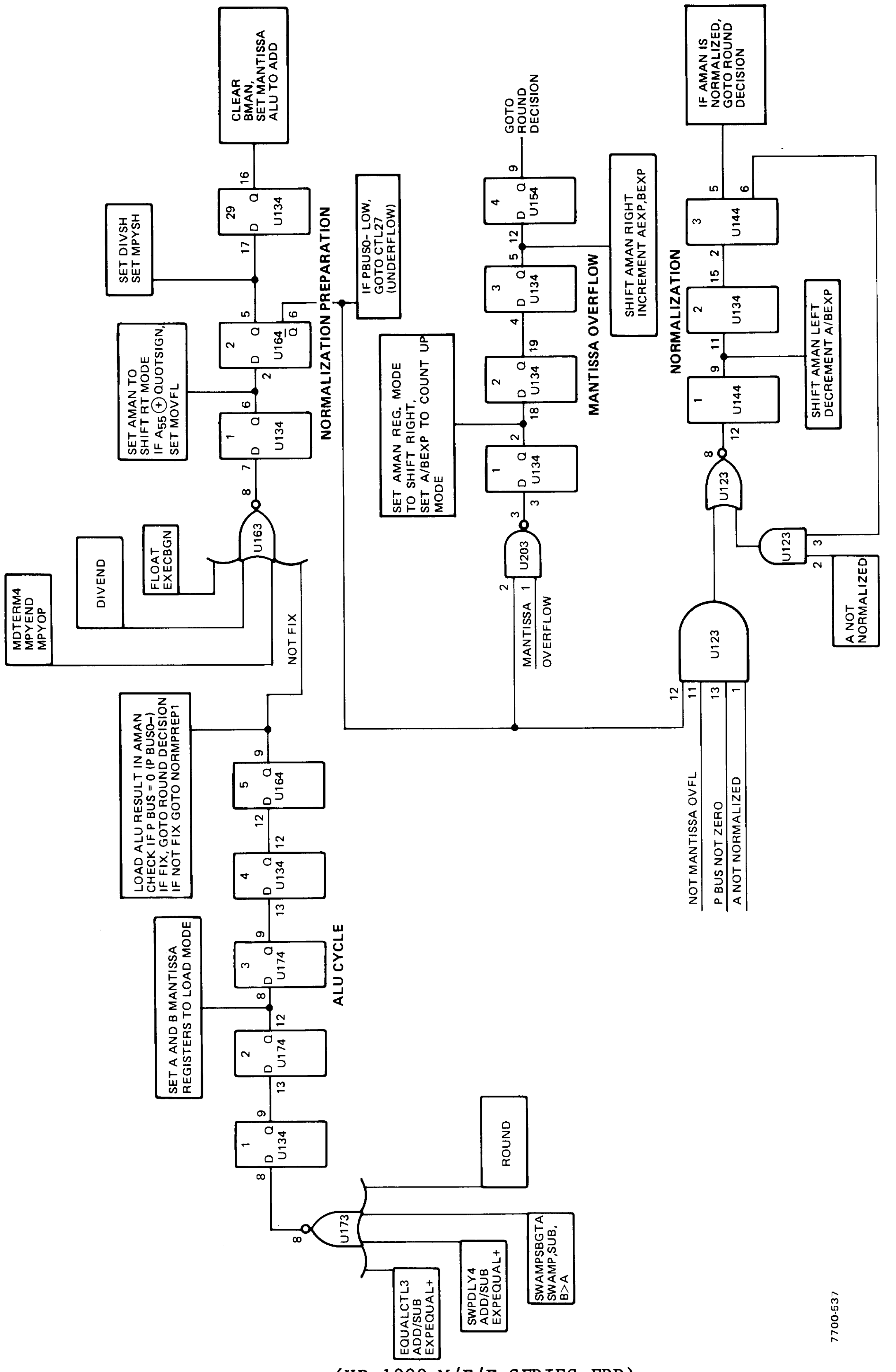

LOAD RESULT IN A/B/C MANTISSA A/BEXP, OUTPUT REGISTERS
MAKE MPPCND READY PREPARE TO LOAD AMAN, BMAN, CMAN, AEXP, BEXP, OUTPUT RG.
OVERFLOW
DISABLE ALL REGISTERS FROM DRIVING P BUS SET FF TO RESET NSTOV
DIAGOP EXECBGN
UNDERFLOW
CLEAR C MANTISSA CMAN DRIVES P-BUS SET EXP. ALU TO CLEAR EXP P-BUS SET FF TO RESET NSTOV
CTL18 EXPONENT UNFL
NORMPREP2 P BUS0+
SWPDLY4 FIX AEXP NEG
EXECBGN MPY BY 0, DIVIDEND 0
EXECBGN DIVIDE BY ZERO
SWPDLY4 FIX, AEXP > 15 OR 31
SWPDLY4 EXPONENT OVFL
AT ROUND DECISION2 GOTO ALUCTL1 ENABLE ROUND BIT IN B MANT. CLEAR LOWER AMAN BITS
ROUND DECISION
ROUND
FIX . EXPEQUAL . (SWPDLY4 OR EQUALCTL3 OR ALUCTL5)
MOVFLCTL4 OR SWAMP4 . ( ) OR B<A)
NORMPREP2 ANORMLD• MOVFL- PBUS0-
NORMCTL3 ANORMLZD
DIVOP,GUARD1
ROUND, NOT FIX
A55 (NEG #) FIX SGL FIX SGL ZERO
A55 (NEG #) FIX DBL FIX DBL ZERO
U62
U114
U186
U105
U133
U143
U124
U156
U54
U153
U113
D Q
CK
7700-538

Figure III-12 (Sheet 3 of 3). Summary of Control Board Schematics



CLEAR BMAN, SET MANTISSA ALU TO ADD
SET DIVSH SET MPYSH
SET AMAN TO SHIFT RT MODE IF A55 ⊕ QUOTSIGN, SET MOVFL
IF PBUS0- LOW, GOTO CTL27 (UNDERFLOW)
NORMALIZATION PREPARATION
GOTO ROUND DECISION
SHIFT AMAN RIGHT INCREMENT AEXP, BEXP
MANTISSA OVERFLOW
SET AMAN REG. MODE TO SHIFT RIGHT, SET A/BEXP TO COUNT UP MODE
IF AMAN IS NORMALIZED, GOTO ROUND DECISION
NORMALIZATION
SHIFT AMAN LEFT DECREMENT A/BEXP
A NOT NORMALIZED
MANTISSA OVERFLOW
NOT MANTISSA OVFL
P BUS NOT ZERO
A NOT NORMALIZED
MDTERM4 MPYEND MPYOP
DIVEND
FLOAT EXECBGN
NOT FIX
LOAD ALU RESULT IN AMAN CHECK IF P BUS = 0 (P BUS0–) IF FIX, GOTO ROUND DECISION IF NOT FIX GOTO NORMPREP1
SET A AND B MANTISSA REGISTERS TO LOAD MODE
ALU CYCLE
ROUND
EQUALCTL3 ADD/SUB EXPEQUAL+
SWPDLY4 ADD/SUB EXPEQUAL+
SWAMPSBGTA SWAMP,SUB, B>A
U134
U164
U174
U163
U203
U154
U144
U123
U173
7700-537

Figure III-12 (Sheet 2 of 3). Summary of Control Board Schematics

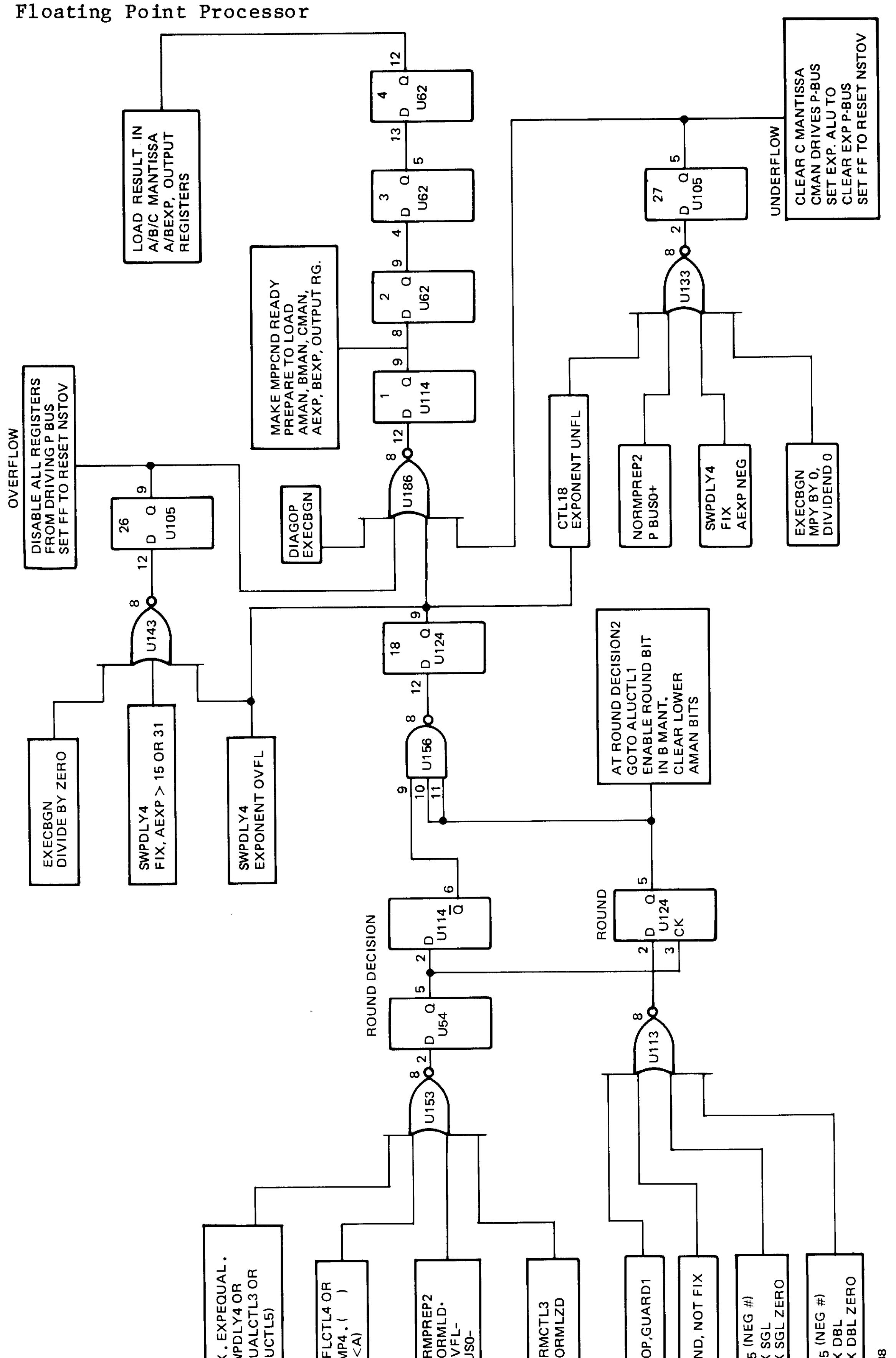

LOAD RESULT IN A/B/C MANTISSA A/BEXP, OUTPUT REGISTERS
U62
U114
MAKE MPPCND READY PREPARE TO LOAD AMAN, BMAN, CMAN, AEXP, BEXP, OUTPUT RG.
OVERFLOW
DISABLE ALL REGISTERS FROM DRIVING P BUS SET FF TO RESET NSTOV
U105
U186
DIAGOP EXECBGN
CLEAR C MANTISSA CMAN DRIVES P-BUS SET EXP. ALU TO CLEAR EXP P-BUS SET FF TO RESET NSTOV
UNDERFLOW
U133
CTL18 EXPONENT UNFL
NORMPREP2 P BUS0+
SWPDLY4 FIX AEXP NEG
EXECBGN MPY BY 0, DIVIDEND 0
U143
U124
EXECBGN DIVIDE BY ZERO
SWPDLY4 FIX, AEXP > 15 OR 31
SWPDLY4 EXPONENT OVFL
U156
AT ROUND DECISION2 GOTO ALUCTL1 ENABLE ROUND BIT IN B MANT. CLEAR LOWER AMAN BITS
ROUND DECISION
ROUND
U54
U113
U153
FIX . EXPEQUAL . (SWPDLY4 OR EQUALCTL3 OR ALUCTL5)
MOVFLCTL4 OR SWAMP4.( ) OR B<A)
NORMPREP2 ANORMLD· MOVFL- PBUS0-
NORMCTL3 ANORMLZD
DIVOP,GUARD1
ROUND, NOT FIX
A55 (NEG #) FIX SGL FIX SGL ZERO
A55 (NEG #) FIX DBL FIX DBL ZERO
7700-538

Figure III-12 (Sheet 3 of 3). Summary of Control Board Schematics

EQUALIZATION CYCLE (Figure III-13)

EQUALCTL1 ACTIONS:
If AGTBZERO- (A>B or B=0) then BRGCK, CKBEXP

If $\overline{\text{AGTBZERO-}}$, then ARGCK, CKAEXP, ROUNDCLOCK

EQUALCTL3 actions:
If FIXOP:EXPEQUAL, then go to ROUND DECISION
If FIXOP-:EXPEQUAL, then go to ALU CYCLE
If EXPEQUAL-, then go to EQUALCTL1

ALU CYCLE
ALUDELAY1 actions:
Set ABRGS1 (AMAN,BMAN register mode S1) high - prepare to load

ALUDELAY5 actions:
ARGCK load ALU result in AMAN
PBUSOCLK if P-bus = 0, then make PBUS0- low
If FIXOP, then go to ROUND DECISION
If FIXOP-, then go to NORMPREP(1)

Delay for Overflow, Zero Mantissa P-bus, A Normalized

NORMALIZATION PREPARATION
NORMPREP1 actions:
Set ABRGS1 (AMAN,BMAN register mode S1) high
Set ARGS0 (AMAN register S0) low - prepare to shift AMAN left
If DIVOP:(ARG55 XOR QUOTSIGN), then set MOVFL+ mantissa overflow at 32-D
NORMPREP2 actions:
Set DIVSH, MPYSH
Enter CLEARBCTL (clear BMAN, set MNTS0 high)
If MOVFL+ high, then go to MANTISSA OVERFLOW
If ANORMLZD+:PBUS0-:MOVFL-, then go to ROUND DECISION
If ANORMLZD-:PBUS0-:MOVFL-, then go to NORMALIZATION
If PBUS0 low, then go to UNDERFLOW

SWAMP (Figure III-14)
SWAMP1 actions:
Clear EXPALUS0, EXPALUS1, and EXPAULUS2 exponent ALU zeroes P-bus
Make EXPLD- (exponent register load) low

SWAMP4 actions:
If AGTBZERO+ (A>B or B=0), then CKBEXP (BEXP=0) and clear BMAN
If AGTBZERO- (B>A OR A=0), then CKAEXP (AEXP=0) and clear AMAN

```
If SWAMPSBGTA+ (SWAMP, SUBTRACT, B>A), then
        set EXPALUS0, S1, S2  high (AEXP IOR BEXP mode)
        go to MDEXPCTL4 (delay and exponent load)
        and go to ALU CYCLE (complement BMAN)
If SWAMSBGTA+ (not SUBTRACT or A>B),
           then go to round decision
```

IN ADD OR SUBTRACT, THE OPERANDS' EXPONENTS MUST BE EQUAL, BEFORE THEIR MANTISSAS CAN BE ADDED OR SUBTRACTED. IN THE EXPONENT EQUALIZATION PROCESS, THE SMALLER OPERAND'S EXPONENT IS INCREMENTED UNTIL IT EQUALS THE OTHER EXPONENT. WITH EACH INCREMENT, THE ASSOCIATED MANTISSA IS SHIFTED ONE PLACE TO THE RIGHT. IN FIX OPERATIONS, BEXP IS LOADED WITH A CONSTANT, AND THEN THE OPERAND, WHICH IS IN A, GOES THROUGH THE EQUALIZATION PROCESS.

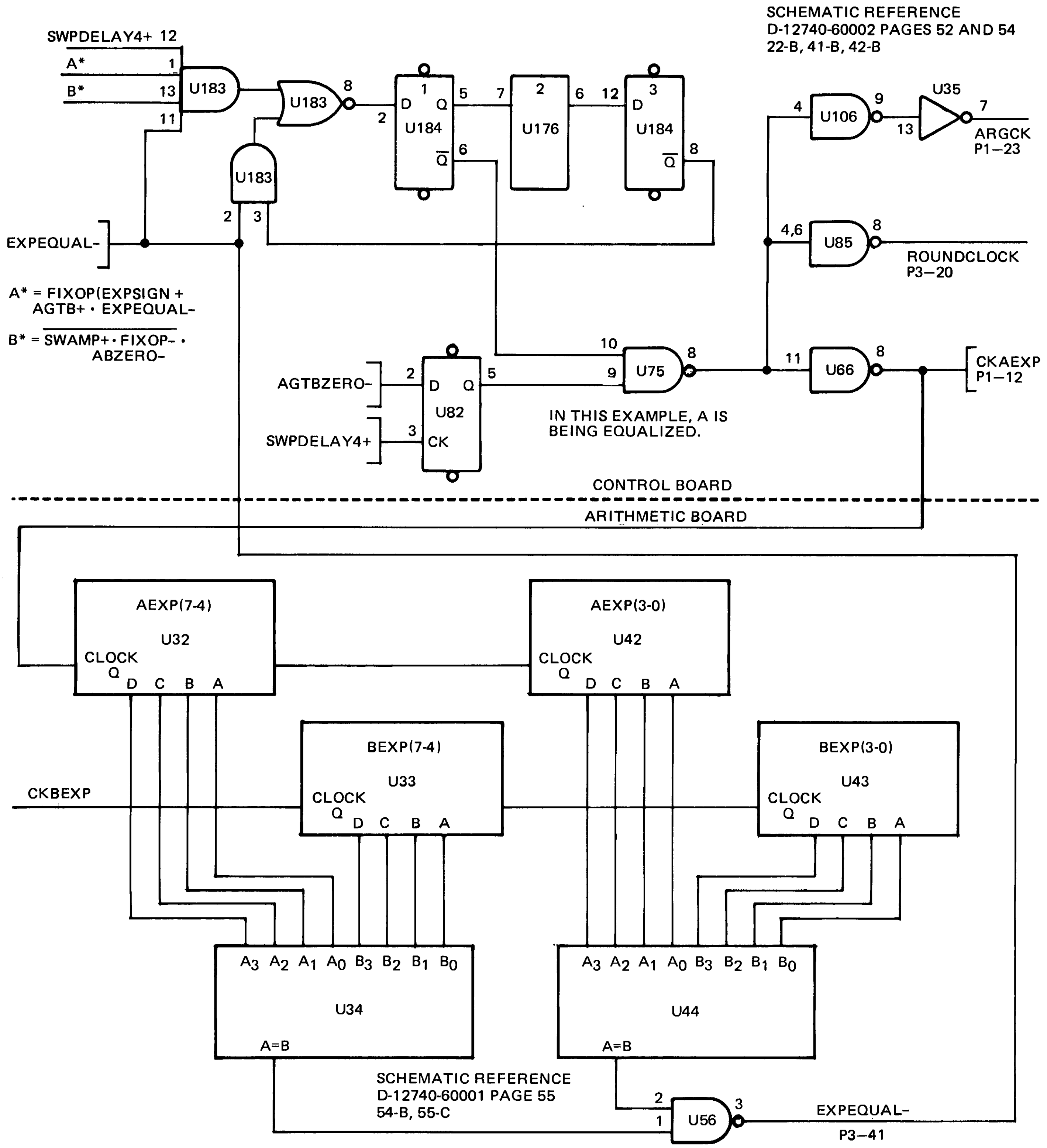


Figure III-13. Exponents Equal Detection Circuitry

(HP 1000 M/E/F-SERIES ERD)

THE SWAMP CONDITION EXISTS WHEN AEXP-BEXP > SWAMP CONSTANT.
IN ADDITION, THE RESULT IS THE LARGER OPERAND.
IN SUBTRACT, IF A IS GREATER THAN B, A IS THE RESULT, BUT IF B IS GREATER THAN A, −B IS THE RESULT.

OPERATION PRECISION	ACTIVE SIGNAL	SWAMP CONSTANT	F_{11}-F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0
A⩾B										
SINGLE (32 BIT)	2WDOP+	24	0	0	0	1	1	0	0	0
EXTENDED (48)	3WDOP+	40	0	0	1	0	1	0	0	0
DOUBLE (64)	4WDOP+	56	0	0	1	1	1	0	0	0
A<B										
SINGLE (32)	2WDOP+	-24	1	1	1	0	1	0	0	0
EXTENDED (48)	3WDOP+	-40	1	1	0	1	1	0	0	0
DOUBLE (64)	4WDOP+	-56	1	1	0	0	1	0	0	0
				F_7	$2WDOP \oplus F_{11}$	$3WDOP \oplus F_{11}$	TIE HIGH			

$$SWAMP+ = \bar{F}_{11} \cdot \{SWAMPCONSTANT > (AEXP-BEXP)\} + F_{11} \cdot \{SWAMPCONSTANT < (AEXP-BEXP)\}$$

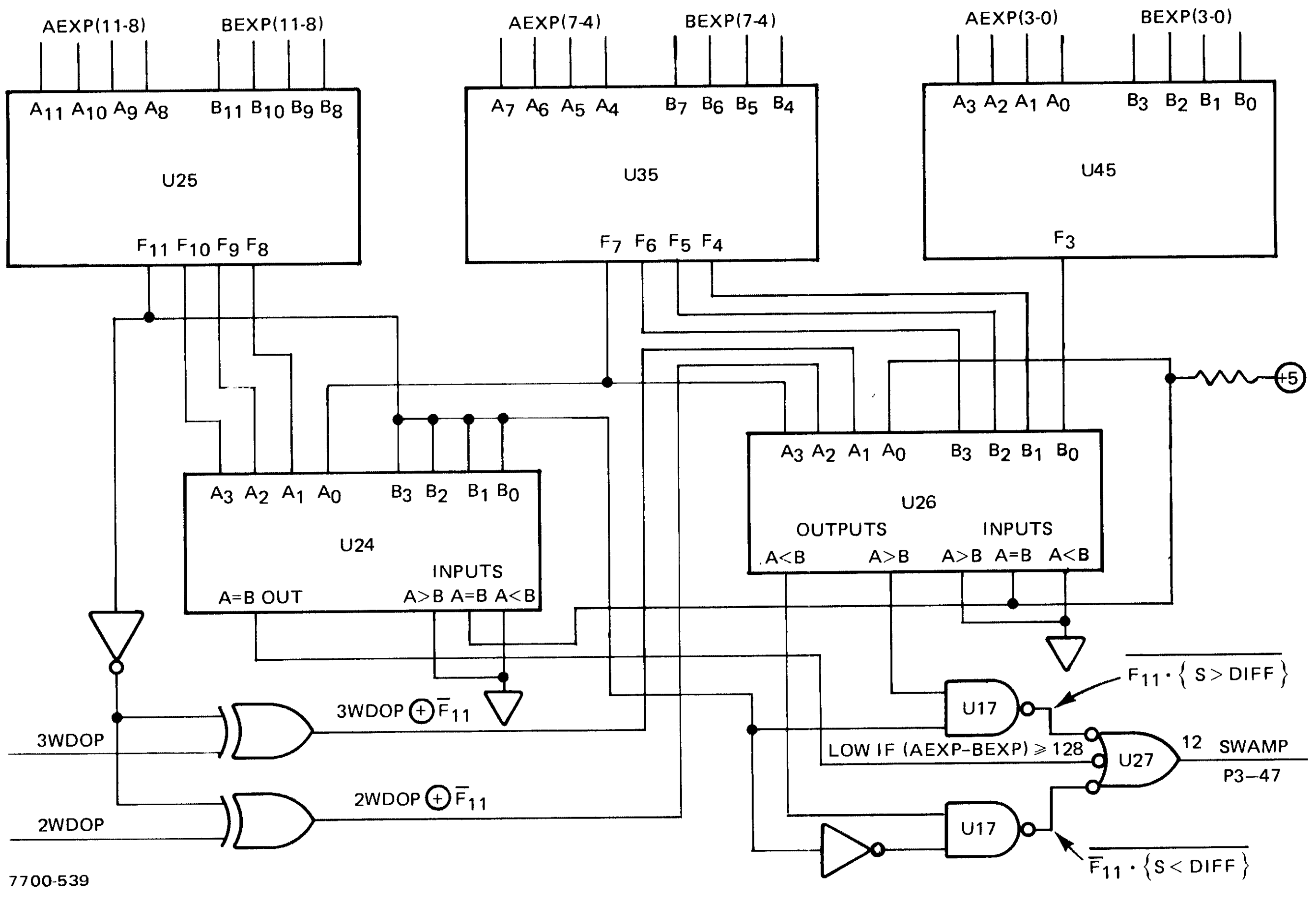


Figure III-14. Swamp Detection Circuits

(HP 1000 M/E/F-SERIES ERD)

MANTISSA OVERFLOW CYCLE (Figures III-15 and 16)

MOVFLCTL1 actions:
    Set ABRGS1 (AMAN/BMAN register mode S1) low
    Set ARGS0 (AMAN register mode S0) high - prepare to shift
            AMAN right
    Set EXPCNTUP+ high - prepare to increment exponent

MOVFLCTL3 actions:
    ARGCK - shift AMAN right
    CKAEXP - increment AEXP
    CKBEXP - increment BEXP
    ROUNDCLOCK shift lowest bit into round circuit

MOVFLCTL4 is delay
    Go to ROUND DECISION

NORMALIZE CYCLE (Figure III-17)

NORMCTL1 actions:
    ARGCK - shift AMAN left
    CKAEXP, CKBEXP - decrement exponents
    ROUNDCLOCK - shift round register left

NORMCTL3 actions:
    If ANORMLZD+ low, go to NORMALIZE cycle
    If ANORMLZD+ high, go to ROUND DECISION

ROUND DECISION CYCLE (Figure III-18)

    ROUND DECISION(1) - clock round control flip-flop

ROUND DECISION(2) actions:
    If ROUND- low, go to ALU CYCLE, increment AMAN
    If ROUND- high, go to exponent over/underflow decision

EXPONENT OVER/UNDERFLOW DECISION (Figure III-19)
    If EXPOVFL+ high, go to OVERFLOW
    If EXPUNFL+ high, go to UNDERFLOW
    If EXPOVUNFL- low, go to TERMINATION

EXPONENT OVERFLOW

    Set MALUEN+ (mantissa ALU output enable) low
    Set OVERFLOWB- (OVERFLOW buffer) low
            let P-bus float high (P-bus[55-1]=1, EXPSIGN=0)
    Set CPUOVFL- (CPU overflow latch) low
            go to TERMINATION

MANTISSA OVERFLOW CASES

1. DURING ADDITION, ALU MODE CONTROL S0 = HIGH
   a. BOTH OPERANDS POSITIVE, ALU RESULT NEGATIVE
      A55 LOW, B55 LOW, ALU55 HIGH

   b. BOTH OPERANDS NEGATIVE, ALU RESULT POSITIVE
      A55 HIGH, B55 HIGH, ALU55 LOW

      $\overline{(A55 \oplus B55)} * (A55 \oplus ALU55)$

2. SUBTRACTION, ALU MODE CONTROL S0 = LOW

   a. 1ST OPERAND POSITIVE, 2ND OPERAND NEGATIVE, ALU RESULT NEGATIVE
      A55 LOW, B55 HIGH, ALU55 HIGH

   b. 1ST OPERAND NEGATIVE, 2ND OPERAND POSITIVE, ALU RESULT POSITIVE
      A55 HIGH, B55 LOW, ALU55 LOW

      $(A55 \oplus B55) * (55 \oplus ALU55)$

$\therefore$ MANTISSA OVERFLOW = $[(A55 + B55) \oplus S0]* [(A55 \oplus B55\ ALU55)]$

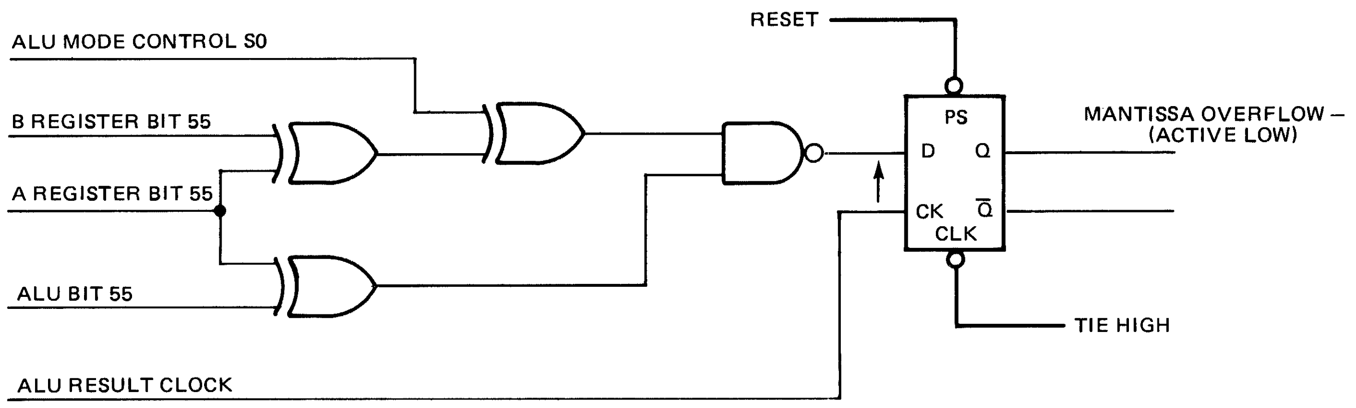


Figure III-15. Mantissa Overflow Cases

WHEN THE MANTISSA OVERFLOWS DURING AN ALU CYCLE, IT MUST BE CORRECTED IN THE MANTISSA OVERFLOW SEQUENCE. THIS SEQUENCE SHIFTS THE AMAN REGISTER TO THE RIGHT ONE PLACE DURING WHICH THE SIGN OF THE MANTISSA IS REVERSED. ALSO, THE EXPONENT WHICH IS HELD IN AEXP AND BEXP IS INCREMENTED.

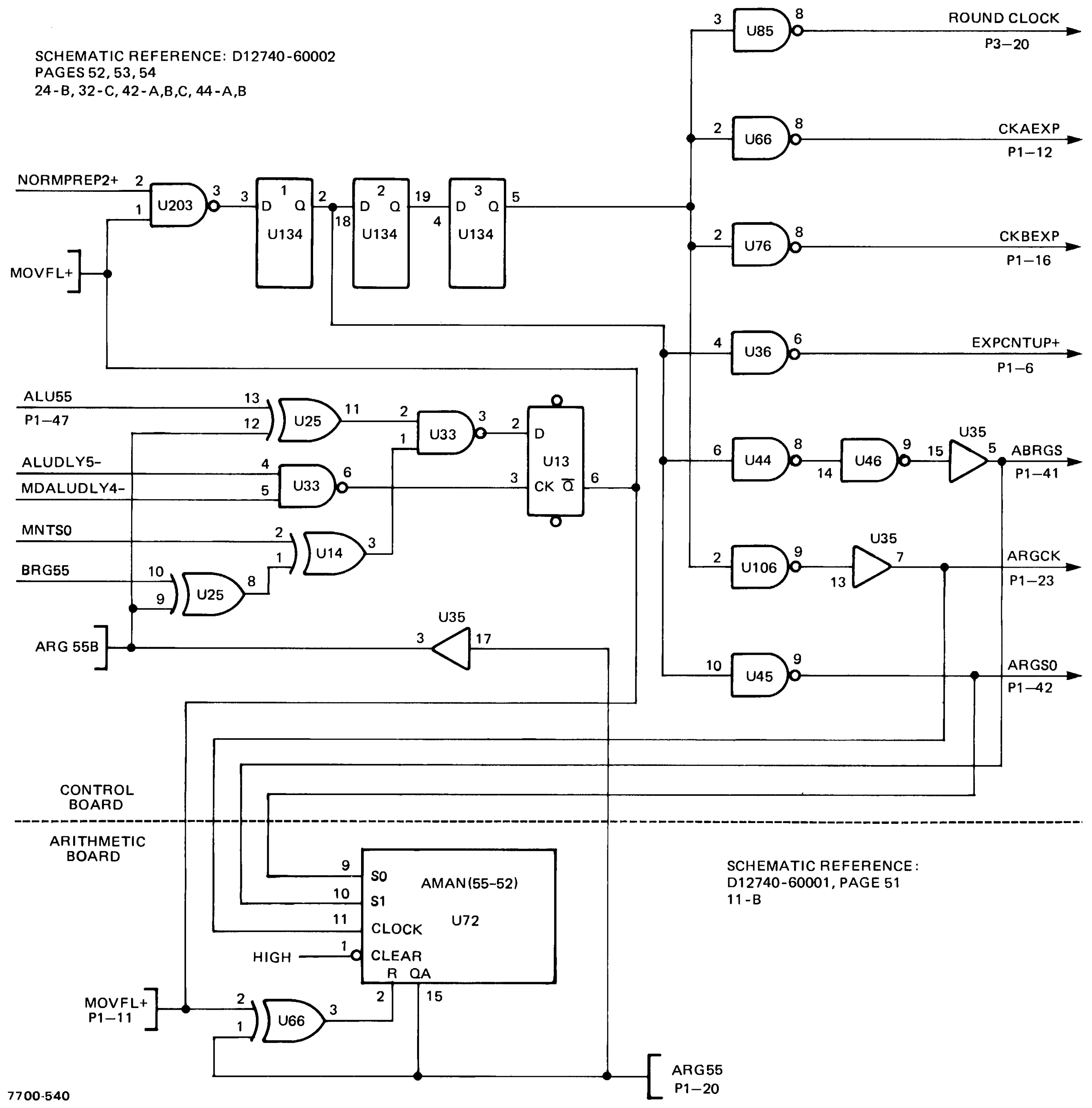


Figure III-16. Mantissa Overflow Circuits

(HP 1000 M/E/F-SERIES ERD)

IF A RESULTANT MANTISSA IS NOT NORMALIZED, IT MUST GO THROUGH THE NORMALIZATION SEQUENCE. THIS SEQUENCE SHIFTS THE A MANTISSA REGISTER LEFT UNTIL ARG55 ≠ ARG55. THE RESULTANT EXPONENT, WHICH IS IN BOTH AEXP AND BEXP, IS DECREMENTED WITH EACH SHIFT.

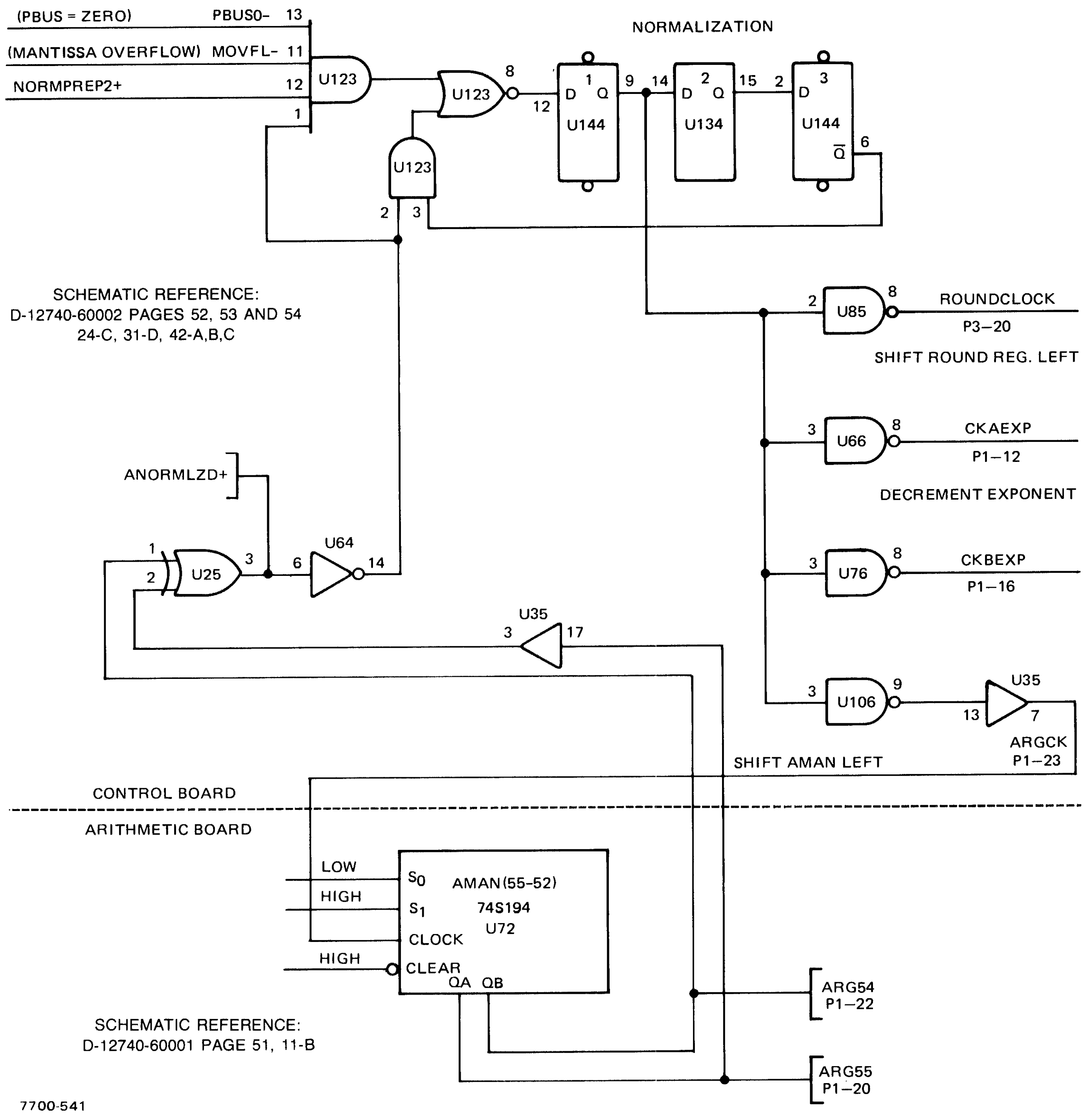


Figure III-17. Normalization Circuits

(HP 1000 M/E/F-SERIES ERD)

ROUND CIRCUITS USED IN ADD AND SUBTRACT.

THE ROUND REGISTER HOLDS 3 GUARD BITS AND SETS UP THE STICKY BIT. ALL SHIFT OPERATIONS ALSO SHIFT THIS REGISTER WHEN AMAN OR BMAN IS SHIFTED TO THE RIGHT, THE APPROPRIATE LSB IS SHIFTED INTO THE ROUND REGISTER. IN SUBTRACT OPERATIONS WHERE B IS BEING EQUALIZED, ONCE A "1" HAS BEEN PASSED TO THE ROUND REGISTER, THE BITS SHIFTED INTO THE ROUND REGISTER ARE COMPLEMENTED.

$$\text{ROUND} = \overline{\text{ARG55}} \cdot \text{GUARD1} + \text{ARG55} \cdot \text{GUARD1} \cdot (\text{GUARD2} + \text{GUARD3} + \text{STICKY})$$

$$= \text{GUARD1} \cdot (\overline{\text{ARG55}} + \text{GUARD2} + \text{GUARD3} + \text{STICKY})$$

$$= \text{GUARD1} \cdot (\overline{\text{ARG55} \cdot \overline{\text{GUARD2}} + \overline{\text{GUARD3}} \cdot \overline{\text{STICKY}}})$$

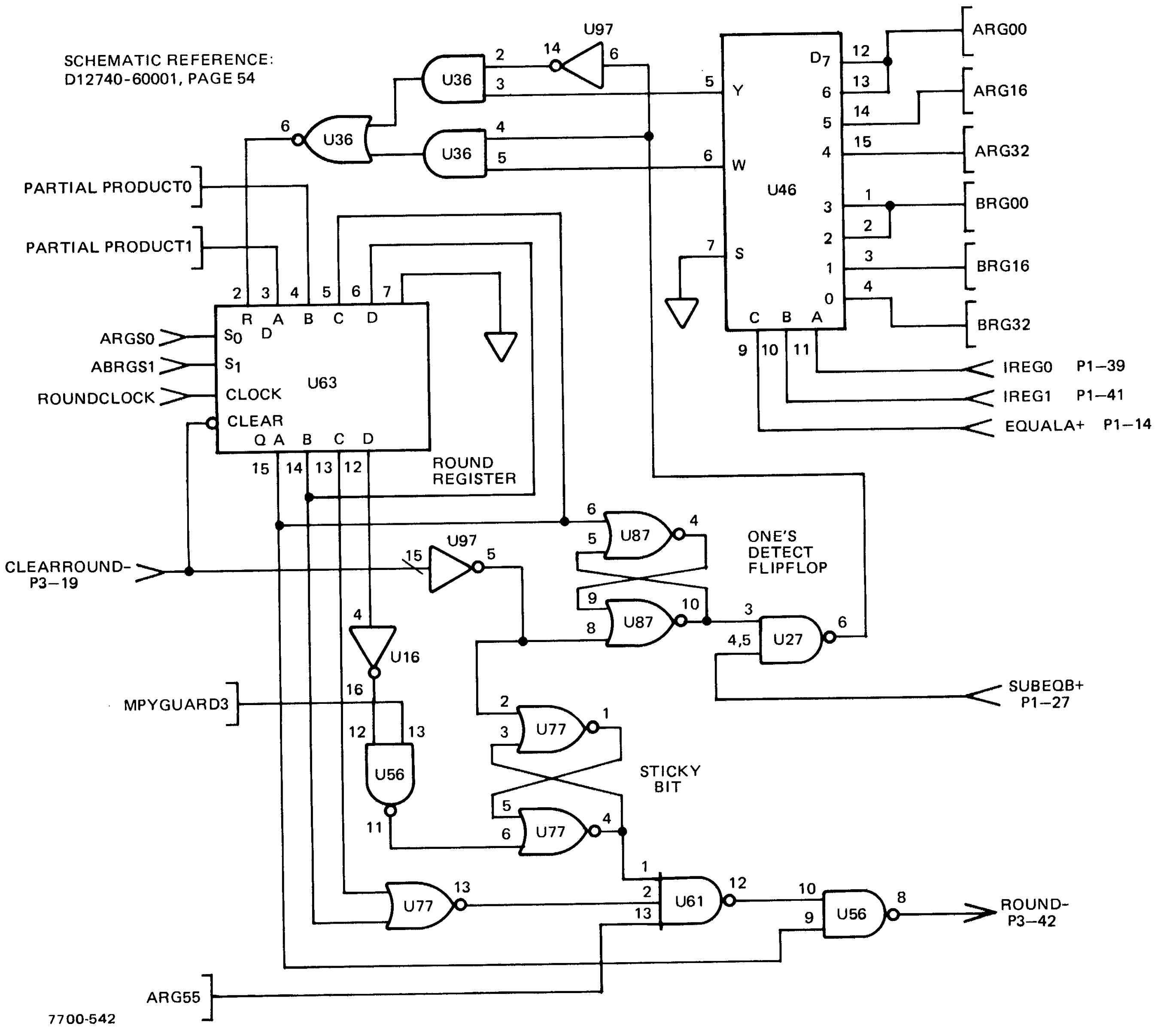


Figure III-18. Round Circuits

(HP 1000 M/E/F-SERIES ERD)

IN THE STANDARD 8 BIT EXPONENT CASE, WHERE INSTRUCTION REGISTER BIT 7 IS ZERO, THE EXPONENT IS OUT OF RANGE WHEN EXP(10-7) ≠ EXP(11). ON THE OTHER HAND, WHEN INSTRUCTION REGISTER BIT 7 IS ONE, THE EXPANDED EXPONENT IS OUT OF RANGE WHEN EXP(10) ≠ EXP(11), IN BOTH CASES, THE OUT OF RANGE EXPONENT OVERFLOWED IF EXP(11) IS ZERO, AND UNDER FLOWED IF EXP(11) IS ONE.

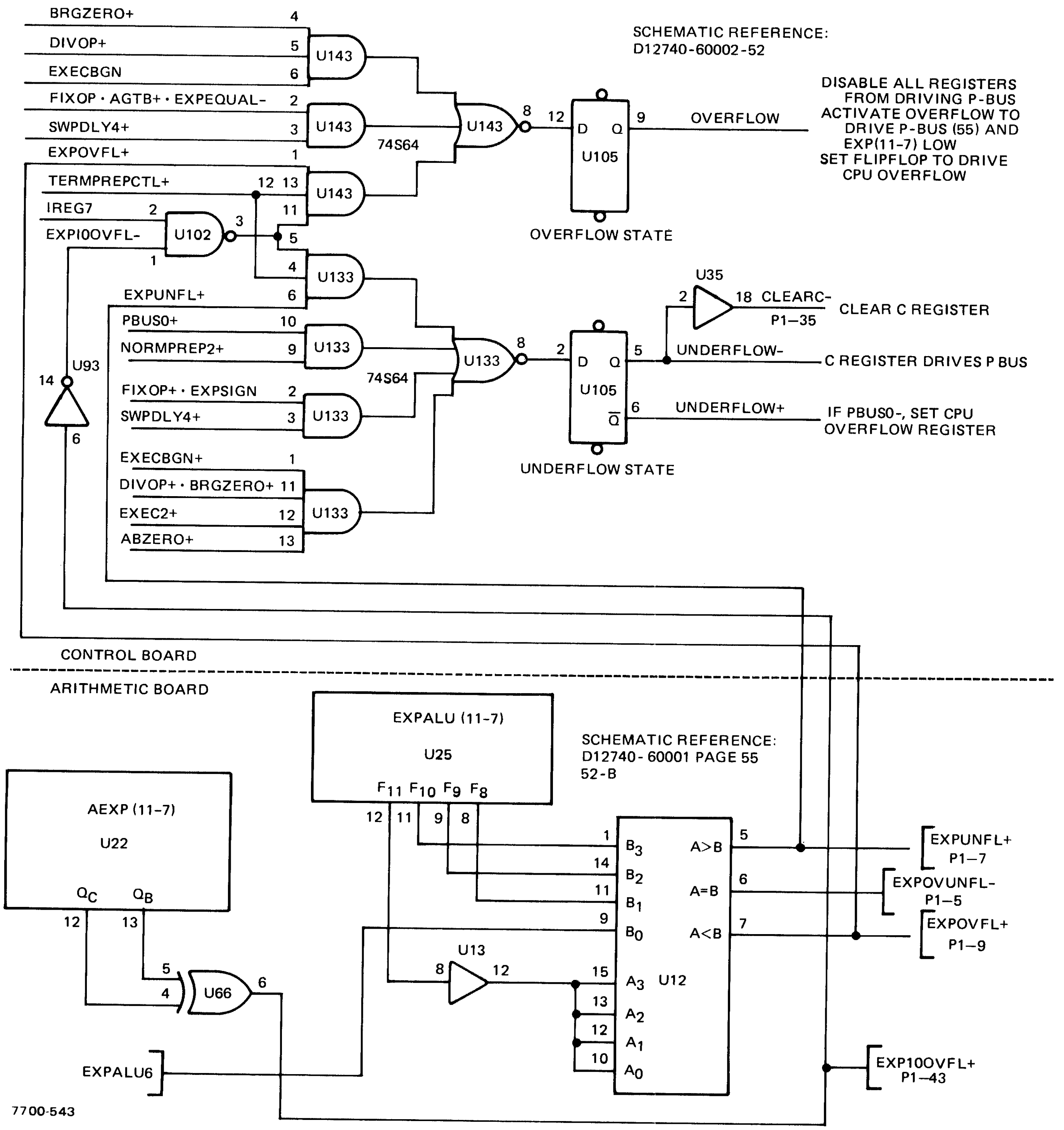


Figure III-19. Exponent Overflow/Underflow

(HP 1000 M/E/F-SERIES ERD)

EXPONENT UNDERFLOW

```
Set MALUEN+ (mantissa  ALU output enable) low
Set CRGEN+ (C register output enable)
            clear C register, exponent ALU S0, S1, S2
            P-bus = all zeroes
Set CPUOVFL- (CPU overflow latch) low
            go to TERMINATION
```

TERMINATION

TERMCTL1 actions:

```
Set ABRGS1, ARGS0 high - prepare to load AMAN and BMAN
Set CRGS0, CRGS1 high - prepare to load CMAN
Set EXPLD- low - prepare to load AEXP and BEXP
Reset CLEARA's - clear lower bits of A
Set MPPCND flip-flop low (FPP ready)
```

TERMCTL4 actions:

```
ARGCK - load result in AMAN
BRGCK - load result in BMAN
CRGCK - load result in CMAN
CKAEXP - load result in AEXP
CKBEXP - load result in BEXP
TERMLOAD - load result in OUTPUT register
```

7.0 OVERVIEW OF EXEC2 OPERATIONS - MULTIPLICATION AND DIVISION

The second group of FPP operations are multiplication and division. Since the multiplication and division process are sequences of shift or arithmetic cycles, they share the same control state machine, called EXEC2, which is depicted on schematics 12740-60002 page 53. Their operations follow the standard loading sequence, meaning that the A and B registers are loaded according to instruction register bits (3,2). At the completion of the loading sequence control passes directly from the execution begin control state to the multiplication/division initialization state at schematics 12740-60002-53-32-B. Control remains in the EXEC2 state machine until multiplier/quotient counter signals that the multiplier scan has completed or that a sufficient number of quotient bits have been formed. From the EXEC2 state machine control passes to the normalization preparation states of the EXEC1 state machine. In the EXEC1 section a product or quotient may be normalized, adjusted for mantissa overflow or rounded. As with all FPP operations, if the exponent is over or under range, the overflow or underflow constant results. The final result is loaded into all registers during the termination sequence.

The previous paragraph summarizes the flow from the loading sequence through to the termination sequence. The following paragraphs are devoted to the operations within the EXEC2 state machine. Refer to Schematic summaries III-3 and III-4.

8.0 IMPLEMENTATION OF THE MULTIPLICATION PROCESS

The limit of the multiplication operation takes place with the EXEC2 phases of operation. It is in the EXEC2 section that the multiplication process multiplies the operand's mantissas and sums their exponents. While the mantissas undergo multiplication, the EXEC2 multiply/divide exponent control sequence sets the exponent ALUs to the A plus B mode and loads the resulting sum into AEXP and BEXP. Note that the FPP multiply algorithm dictates that (AEXP + BEXP +1) should be the product exponent. The carry input to the least significant ALU bit is high to form the +1 of the exponent sum. Note that this carry input is also high during division while the ALUs are in the subtract mode, so that two's complement subtraction is performed. Since this carry input is used only in these two cases, it is tied high through a pull-up resistor.

During multiplication, control flows from the execution begin state to the multiplication/division initialization state at control board schematic page -53-31-A. At this point the multiplicand occupies the B register and the multiplier mantissa resides in CMAN. AMAN is cleared, in order to initialize the partial product to zero. Multiplication is a process of shift and arithmetic cycles whose sequence depends on bits of the multiplier which is scanned from the least significant to the most significant bit. The decision on which type cycle, shift or arithmetic, to perform next depends on a history bit and the two bits currently in the two least significant bit positions of the multiplier. The first decision is formed during the initialization state. Also, at this state the multiplier is shifted right one place to prepare for the second decision. As the multiplier is one position ahead of the partial product, the next cycle decision can always be performed during the current cycle. Figure III-20 shows the decision circuits and Figure III-21 displays the actions at each state in the EXEC2 group. The following paragraphs discuss only the events that require additional explanation in facilitating the understanding of the hardware design.

THESE CIRCUITS CONTROL THE SEQUENCE OF ALU AND SHIFT CYCLES THAT MAKE UP MULTIPLICATION. THE DECISION TO DO AN ALU OR SHIFT CYCLE IS BASED ON THE CURRENT TWO LEAST SIGNIFICANT BITS OF THE MULTIPLIER AND A HISTORY BIT. THE C REGISTER HOLDS THE MULTIPLIER, WHILE THE PRODUCT IS DEVELOPED IN THE A REGISTER.

ALGORITHM TRUTH TABLE

C_1	C_0	H	NEXT H	OPERATION
0	0	0	0	SHIFT
0	0	1	0	ADD
0	1	0	0	ADD
0	1	1	1	SHIFT
1	0	0	0	SHIFT
1	0	1	1	SUBTRACT
1	1	0	1	SUBTRACT
1	1	1	1	SHIFT

C_0 IS THE BIT CURRENTLY IN THE LSB POSITION OF THE MULTIPLIER

C_1 IS THE BIT JUST LEFT OF THE LSB IN THE MULTIPLIER

SHIFT OPERATION $= HC_0 + \overline{HC_0} = H \oplus C_0$

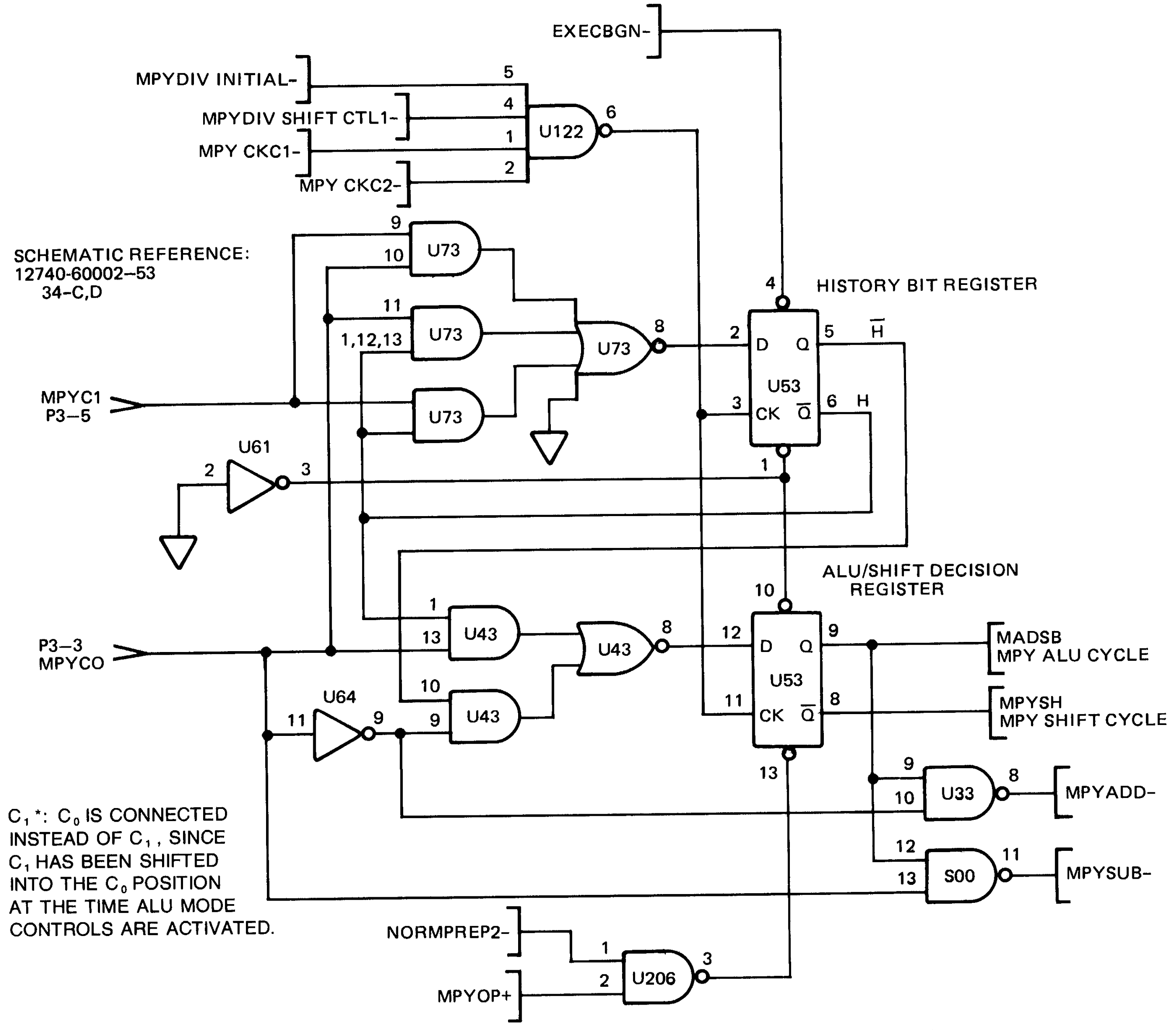
ALU OPERATION $= \overline{H} \oplus C_0$

ADD $=$ ALU $\cdot \overline{C_1^*}$

SUBTRACT $=$ ALU $\cdot C_1^*$

NEXT H $= C_1 C_0 + C_0 H + C_1 H$

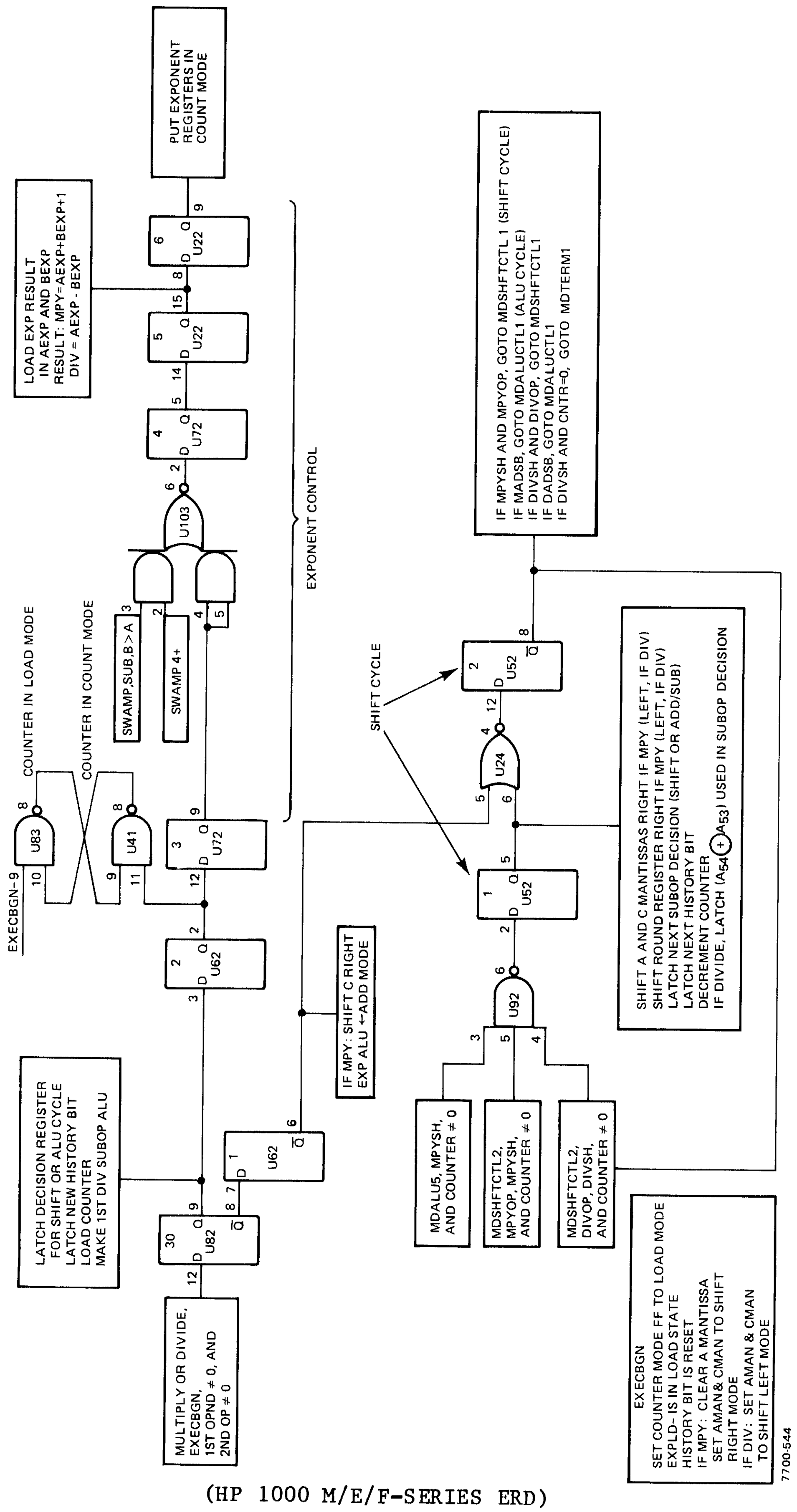
TEXT REFERENCE: THE LOGIC OF COMPUTER ARITHMETIC BY IVAN FLORES CH. 10.



C_1*: C_0 IS CONNECTED INSTEAD OF C_1, SINCE C_1 HAS BEEN SHIFTED INTO THE C_0 POSITION AT THE TIME ALU MODE CONTROLS ARE ACTIVATED.

Figure III-20. Multiplication Decision Circuit

(HP 1000 M/E/F-SERIES ERD)



PUT EXPONENT REGISTERS IN COUNT MODE
LOAD EXP RESULT IN AEXP AND BEXP RESULT: MPY=AEXP+BEXP+1 DIV = AEXP - BEXP
COUNTER IN LOAD MODE
COUNTER IN COUNT MODE
EXECBGN-9
U83
U41
SWAMP,SUB,B > A
SWAMP 4+
U103
EXPONENT CONTROL
U72
U22
U62
LATCH DECISION REGISTER FOR SHIFT OR ALU CYCLE LATCH NEW HISTORY BIT LOAD COUNTER MAKE 1ST DIV SUBOP ALU
IF MPY: SHIFT C RIGHT EXP ALU ←ADD MODE
U82
MULTIPLY OR DIVIDE, EXECBGN, 1ST OPND ≠ 0, AND 2ND OP ≠ 0
IF MPYSH AND MPYOP, GOTO MDSHFTCTL 1 (SHIFT CYCLE)
IF MADSB, GOTO MDALUCTL1 (ALU CYCLE)
IF DIVSH AND DIVOP, GOTO MDSHFTCTL1
IF DADSB, GOTO MDALUCTL1
IF DIVSH AND CNTR=0, GOTO MDTERM1
SHIFT CYCLE
U52
U24
U92
SHIFT A AND C MANTISSAS RIGHT IF MPY (LEFT, IF DIV)
SHIFT ROUND REGISTER RIGHT IF MPY (LEFT, IF DIV)
LATCH NEXT SUBOP DECISION (SHIFT OR ADD/SUB)
LATCH NEXT HISTORY BIT
DECREMENT COUNTER
IF DIVIDE, LATCH ($A_{54} \oplus A_{53}$) USED IN SUBOP DECISION
MDALU5, MPYSH, AND COUNTER ≠ 0
MDSHFTCTL2, MPYOP, MPYSH, AND COUNTER ≠ 0
MDSHFTCTL2, DIVOP, DIVSH, AND COUNTER ≠ 0
EXECBGN
SET COUNTER MODE FF TO LOAD MODE
EXPLD- IS IN LOAD STATE
HISTORY BIT IS RESET
IF MPY: CLEAR A MANTISSA
SET AMAN & CMAN TO SHIFT RIGHT MODE
IF DIV: SET AMAN & CMAN TO SHIFT LEFT MODE
7700-544

Figure III-21 (Sheet 1 of 2). Multiply/Divide Control States Schematic (12740-60002)

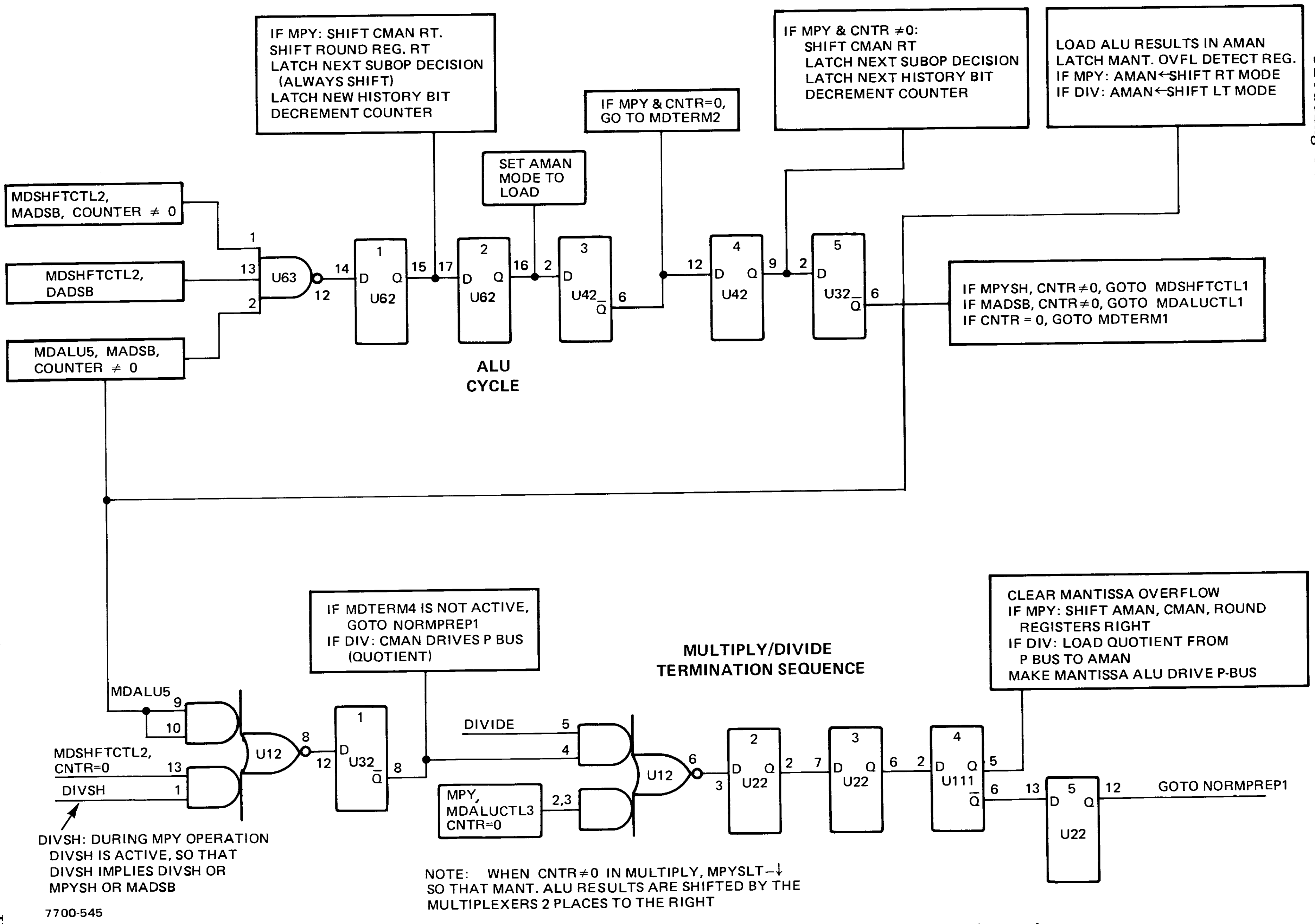


Figure III-21 (Sheet 2 of 2). Multiply/Divide Control States Schematic (12740-60002)

The multiplication process is a sequence of shift and arithmetic cycles. A shift cycle consists of only two control states, so that consecutive shifts can occur only 50 nanoseconds apart. In order to prepare for these short shift cycles AMAN is set to the shift right mode at execution begin and at the end of ALU cycles. Since AMAN already is in the shift right mode, the shift cycle merely clocks this right shift via the signal MDSHIFTCTL-. This signal also decrements the counter.

Arithmetic cycles consist of five control states, since it takes up to 125 nanoseconds to complete a 56 bit addition or subtraction. Arithmetic cycles combine an ALU operation and a shift operation. Besides forming a new partial product through the ALUs, the arithmetic cycle shifts the partial product to the right twice through multiplexers, shifts the multiplier twice and decrements the counter twice. The multiplier shifts and counter decrements are accomplished by the signal MPYCKC1-, which is active at the first arithmetic cycle state, and the signal MPYCKC2- which is active during the third state. As AMAN is always in the right shift mode during shift cycles, the arithmetic cycle signal MDALUCTL2- must switch AMAN to the load mode. During the fifth state the partial product is loaded into AMAN, and AMAN is switched back to the right shift mode. The shift and arithmetic cycles are shown in great detail in Figure III-21.

The multiply/divide counter at 12740-60002-53-36-C signals the end of the product forming process when the entire multiplier has been scanned. This counter is decremented each time the multiplier is shifted. It is loaded during the initialization state with the binary values of 0010 0111, 0011 0111 or 0100 0111 depending on whether a 24, 40 or 56 bit multiplier is used. As an example examine the 24-bit case; the upper four bits counter is decremented to 0001 after eight counts and is decremented to 0000 after sixteen more counts. At the 0000 0000 point CNTZERO+ and CNTZERO- become active and prevent control from entering the shift or arithmetic cycle. Instead, control passes to the multiply/divide termination sequence. Also, CNTZERO causes the mantissa ALU multiplexers to switch mode in order to select not the twice shifted ALU results, but the ALU results directly.

The product forming process always terminates with an arithmetic cycle, since the multiplier sign bit differs from the most significant mantissa bit at this final arithmetic cycle. The counter will equal 2 or 1. If it equals 1, the product must not be shifted twice through the ALU multiplexers. When the uppermost three bits of the multiplier are 0.10 or 1.01, the final arithmetic cycle starts with the counter equal to two. The arithmetic cycle performs the usual multiplicand addition or subtraction and shifts the partial product and multiplier twice to the right. The second shift (MPYCKC2-) will zero the counter. At the end of this cycle, control passes from MDALUCTL5 to the multiply/divide termination sequence. The first state of this sequence activates the signal MPYEND+. MYPEND+ causes control to pass from the EXEC2 state machine to the EXEC1 state machine at the normalization preparation sequence. From this point the product may be normalized and checked for rounding and exponent overflow or underflow.

The second multiplication termination case concerns the other set of multipliers whose most significant bits are 0.11 or 1.00. In this case, the next to the last cycle is a shift cycle which occurs when the counter equals 2. Since the final cycle is an arithmetic cycle where the counter equals one, the partial product should be shifted once only not twice as in other ALU cycles. The signal MPYCKC1- at the first state of the arithmetic cycle zeros the counter. The CNTZERO signal changes the ALU multiplexers select input to pass the unshifted version of the partial product to the P-bus. At this point the P-bus holds a partial product which should be shifted once to the right. Note from the schematic page 53-34-B that from the third arithmetic state control passes to both the fourth arithmetic state, and since the counter is zero, also passes to the second state of the multiply/divide termination sequence. The arithmetic cycle state loads the partial product into AMAN (MPYCKC) and the EXEC2 termination state shifts the partial product to the right (MDTERM4). Since MDTERM4- shifts AMAN, control state MDTERM5 is used for delay before control passes to the EXEC1 group. In this case, MPYEND+ is active at the same time MDTERM4- is active, which prevents control from passing to the normalization preparation states until the fifth state of the termination sequence when DIVEND+ is active (see U163-2,3,4,5,6 at page 52-21-D).

Occasionally, mantissa overflow occurs during the partial product forming process. The mantissa overflow detection circuits on page 53-32-D handles the overflow in two ways. First of all, if it occurs during an arithmetic-double shift cycle, PBUS56 loads the proper sign bit into the partial product. On the other hand, in the -1 times -1 case, mantissa overflow occurs in the final arithmetic cycle where only one shift takes place. In this case the signal MOVFL+ causes the proper sign bit to be shifted into AMAN(55). The -1 times -1 case is the only case where the product coming out of EXEC2 is already normalized.

During the product formation process, rounding information is set up in two ways. During shift cycles partial product bits are routed from the appropriate least significant bit position of AMAN through multiplexers to the right shift input of the round register. On the other hand, during arithmetic cycles, the appropriate two least significant bits of the ALUs are loaded into the first two guard bits of the round register. These circuits are summarized in Figure III-22.

DURING MULTIPLY ALU RESULTS ARE SHIFTED TWO PLACES TO THE RIGHT. THUS, IN ORDER TO MAINTAIN PROPER GUARD BITS, THE TWO LEAST SIGNIFICANT BITS OF ALU RESULTS ARE MULTIPLEXED INTO GUARD1 AND GUARD2 AT THE SAME TIME SHIFTED ALU RESULTS ARE LOADED INTO AMAN.

IN DIVISION, THE QUOTIENT IS ROUNDED UP IF GUARD1 IS "1". THE LAST QUOTIENT BIT FORMED, WHICH IS THE BIT TO THE RIGHT OF THE LSB, IS LOADED INTO GUARD1 WHEN THE QUOTIENT IS TRANSFERRED TO THE A REGISTER.

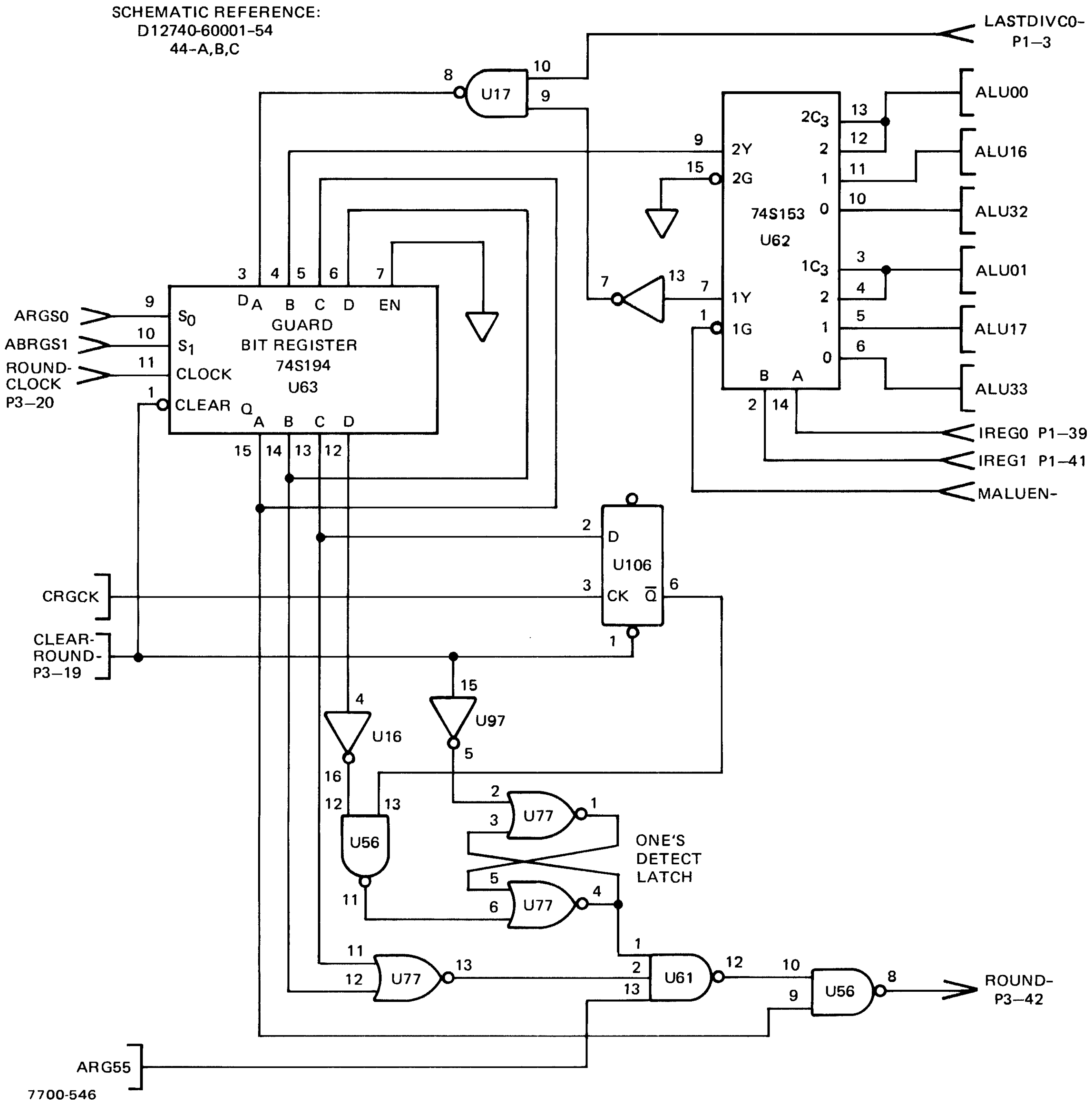


Figure III-22. Round Circuits Devoted to Multiplication and Division

(HP 1000 M/E/F-SERIES ERD)

SCHEMATIC SUMMARY III-3

MULTIPLY - EXPLANATION OF EXEC2 STATE MACHINE

Schematic Reference: D-12740-60002 page 53

Execution begin control state status:

Clear A - A-register is zero
ARGS0 high, ARGS1 low - A-register is in shift right mode
CRGS0 high, CRGS1 low - C-register is in shift right mode
MPY CNTR load FF is in load state
History register is reset
EXPLD- Exponent registers are in load state

Multiplication initialization control state actions:

Clock history bit register - 1st multiplier decision
Clock shift/ALU decision register - 1st multiplier decision
Enter exponent add and load sequence
Load multiplier counter
Enter MDEXPCTL1

MDEXPCLT1 control state actions:
MPYCKCINIT: clock C-reg; C is one bit ahead of A
Make EXPALUS0 high : exponent ALU is in add mode
Enter MDSHFTCL2: Note 1st cycle decision has been made.

EXPONENT CONTROL
MDEXPCTL2 control state actions:
Make counter load FF in count enable state

MDEXPCTL5:
CKAEXP - Load exponent ALU results into AEXP
CKBEXP - Load exponent ALU results into BEXP

MDEXPCTL6:
Make EXPL0- high; exponents are in count mode

If MPYSH, enter MDSHIFTCTL1 (shift cycle)
If MADSB, enter MDALUCTL1 (ALU shift cycle)

MULTIPLY - SHIFT CYCLE

MDSHIFTCTL1:
Shift A & C mantissas right
Shift round register right
Clock next cycle decision (shift or add/sub)
Clock new history bit
Decrement counter

MDSHIFTCTL2:

If MPYSH, enter MDSHIFTCTL1
If MADSB, enter MDALUCTL1

MULTIPLY - ALU CYCLE

MDALUCTL1:

Activate MPYCKC1- Shift C mantissa right
Clock next SUBOP decision - note decision is always to shift
Clock next history bit
Decrement counter

MDALUCTL2:

Make ARGS0 and ABRGS1 high - prepare AMAN to load

MDALUCTL3:

If not CNTZERO- high, then shift C mantissa right via MPYCLKC2
If CNTZERO+ high, then go to MDTERM2
Clock next cycle decision
Clock next history bit
Decrement counter

MDALUCTL4:

Load ALU results into AMAN
Clock mantissa overflow register
Make ABRGS1 low - put AMAN in shift right mode via MPYSHIFTRT-

MDALUCTL5:

If CNTZERO- high and MPYSH, then enter MDSHIFTCTL1
If CNTZERO- high and MADSB, then enter MDALUCTL1
If CNTZERO- low, enter MDTERM1

MULTIPLY - TERMINATION OF

MDTERM1:

If MDTERM4 is not also active, go to NORMPREP1

MDTERM4:

Preset mantissa overflow register
Shift AMAN right
Shift round register right

MDTERM5:

End of multiply or divide, go to NORMPREP1

Counter = zero:

MPYSLT- goes high = Make MPLXR's on ALU board pass data from ALU straight thru to P-bus

End of multiply, most significant bits of multiplier are 0.11 or 1.00:

At last cycle counter = 1 at MDALUCTL1.

MDALUCTL1 clock decrement - counter goes to 0. Control passes into both MDALUCTL3 and MDTERMCM2. Since counter = 0, MPYSLT- is high and causes ALU results to pass straight thru multiplexer. At MDALUCTL4 ALU results are loaded into AMAN, and AMAN goes to the shift right mode. MDTERM4 clocks the required right shift to AMAN. If there is a mantissa overflow at MDALUCTL4, the proper sign bit gets shifted into AMAN. Since multiply takes care of its own mantissa overflows, it should never enter MOVFLCTL1. For this reason, MDTERM4 clears mantissa overflow.

9.0 IMPLEMENTATION OF THE DIVISION PROCESS (Figure III-23)

Since division is similar to multiplication in that is a process of arithmetic and shift cycles, it also is executed in the EXEC2 state machine. At the execution begin point, the A register holds the dividend and the B register holds the divisor. During the division process the divisor is repeatedly added to or subtracted from the quantity in AMAN. The arithmetic results, which which become the new partial remainder, are loaded into AMAN. The multiply/divide initialization state loads the counter with the proper value to form a 24, 40 or 56 bit quotient. The initialization state also latches the value of the sign of the end quotient in U25, the quotient sign register. The quotient sign is the Exclusive-OR function of AMAN (55) and BMAN (55).

From the initialization state control passes to the exponent sequence. The exponent sequence computes the difference between the exponents, AEXP-BEXP, and loads this value into both AEXP and BEXP.

IN DIVISION, THE QUOTIENT IS FORMED THROUGH SEQUENCES OF ALU AND SHIFT CYCLES WHICH ARE CONTROLLED BY THE CIRCUITS BELOW. NOTE THAT THE NEXT CYCLE DECISION CANNOT BE DETERMINED UNTIL THE PARTIAL REMAINDER OF THE CURRENT CYCLE IS FULLY FORMED.

SCHEMATIC REFERENCE: 12740-60002-53, 32-C,D

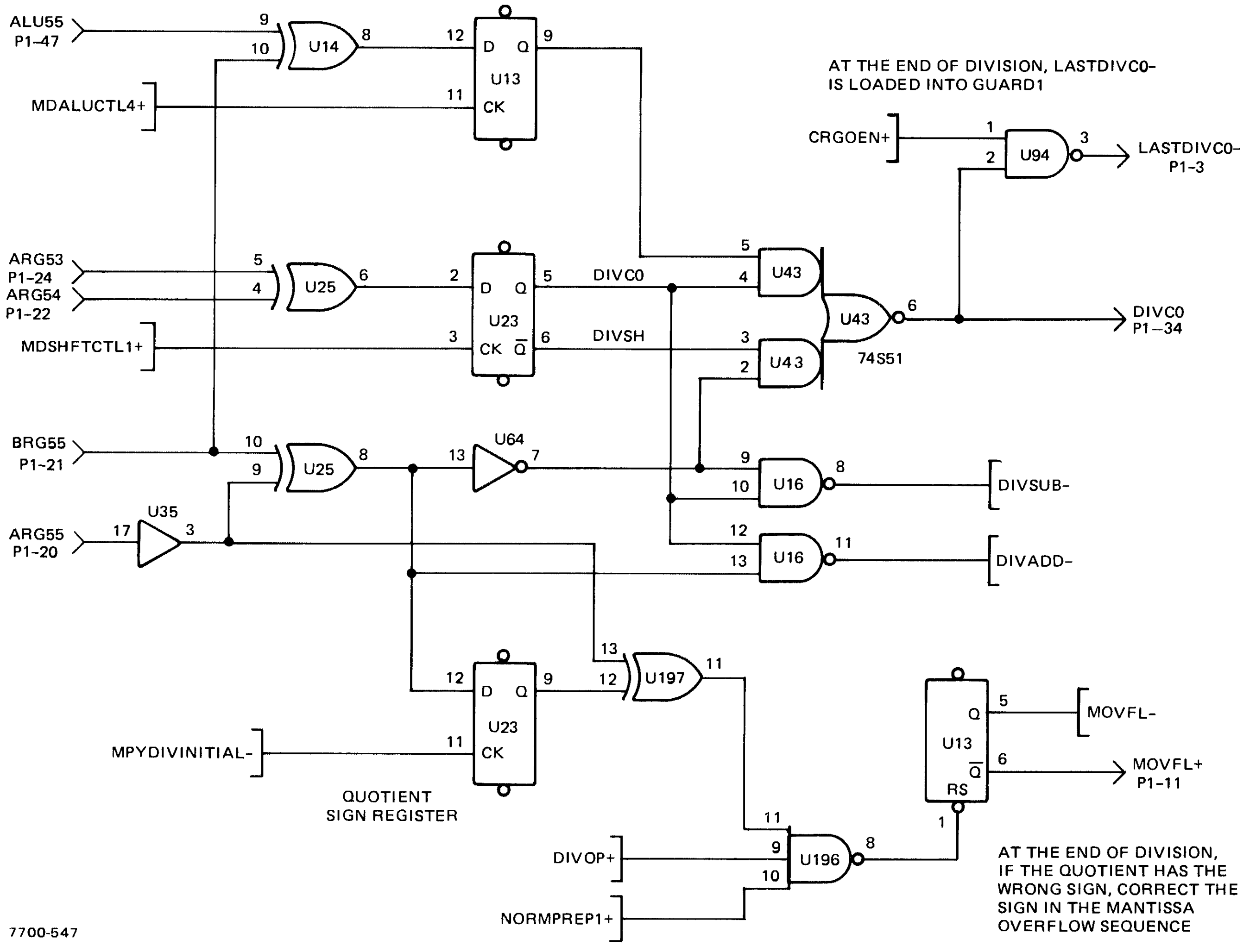


Figure III-23. Division Sequence Decision Circuits

(HP 1000 M/E/F-SERIES ERD)

The decision rules for entering an ALU cycle or a shift cycle are much simpler for division than for multiplication. In summary, perform a shift cycle until the partial remainder is normalized, then do an ALU cycle. At the initialization state the dividend is already normalized, since it is a valid FPP operand, and the first division cycle executed is an arithmetic cycle. At the initialization state the arithmetic/shift cycle decision register points to the arithmetic cycle, since its reset is controlled by the NOR function of DIVOP- and NORMPREP2+. Thus, from the initialization state control passes to the arithmetic cycle.

During this first arithmetic cycle the divisor is added to or subtracted from the dividend. The arithmetic cycle results, which become the current partial remainder, are loaded into AMAN. Thus, each arithmetic cycle wipes out the previous partial F OFFnder, which by the way was the original dividend in the first cycle. The quotient bit in arithmetic cycles in the Exclusive-NOR function of ALU55 and AMAN(55). Since the quotient bit depends on the sign of the ALU results, the quotient bit is latched at the same time (MDALUCTL4) the new fully formed partial remainder is loaded into AMAN. Every arithmetic cycle is followed by a shift cycle which shifts the new quotient bit into CMAN and shifts the new partial product left. The arithmetic cycles include a shift cycle, and require 175 nanoseconds.

Another function that the shift cycle following the arithmetic cycle performs is to latch the decision of what cycle to execute next. The MDSHIFTCTL1 state which shifts the partial remainder in AMAN left, also latches the Exclusive-OR function of AMAN(54) and AMAN(53) into the next decision register. If AMAN(54) equals AMAN(53) then a shift cycle is executed next. Otherwise, the next cycle is an arithmetic cycle. The function of shift cycles is to normalize the partial remainder. In shift cycles, the quotient bit is merely the Exclusive-NOR function of the signs of the partial remainder and divisor which are always accessible. Thus, the new quotient bit is shifted into CMAN at the same time the partial product is shifted left. Therefore, a shift cycle shifts in a new quotient bit, shifts the partial product, latches the decision as to the next cycle type, and decrements the counter, all within 50 manoseconds.

During the division process, AMAN is either in the load mode or shift left mode, and CMAN is always in the shift left mode. Quotient bits are shifted into the least significant bit of CMAN, which is CMAN(32), CMAN(16) and CMAN(0) for single, extended and double precision operations, respectively. The multiplication/ division counter is loaded with the proper value to form a 24, 40 or 56 quotient. Note that the first quotient bit is entered in the least significant bit position and then is shifted left all the way to the sign bit position. The all zero condition of the counter forces the division process to end.

However, before the signal CNTZERO has time to shut down the division process, one extra quotient bit is formed. This extra bit called LASTDIVC0- is used in division's rounding decision. It is loaded into the first guard bit at state four of the multiply/divide termination sequence.

When the counter is zeros, the signal CNTZERO becomes active. As in multiplication, control then passes to the multiptication/ division termination sequence. This sequence routes the quotient from CMAN to AMAN. For instance, DIVTERM1 causes CMAN to drive the quotient on the P-bus. MDTERM4 loads the quotient into AMAN, Now that the quotient resides in AMAN, it can be subject to sign correction, mantissa overflow, and rounding. Note that at DIVEND control passes from the EXEC2 termination sequence to the normalization preparation sequence in EXEC1.

The normalization preparation sequence examines the sign of the quotient mantissa, AMAN(55), which is formed during the first arithmetic cycle of division. If the magnitude of the divisor mantissa is less than the dividend mantissa, the first arithmetic operation successfully reduces the dividend. An indication of successful operation is if the partial remainder has the same sign as the dividend. If the divisor and original dividend have the same sign, the successful quotient bit is a one, and unsucessful quotient bits are zero. Conversely, the successful quotient bit for divisors and dividends of different signs is a zero. This first quotient bit ends up in the sign position. If the first cycle is successful the quotient will not have the proper sign. The correction to the quotient sign is handled as a case of mantissa overflow. The sign of the quotient is checked during NORMPREP1, and if it is wrong, the mantissa overflow detection register is reset to activate MOVFL+. See schematics 12740-60002 page 53-32-D and Figure III-23 for these circuits. Thus, in the mantissa overflow sequence the proper sign is shifted into the mantissa quotient, and the exponent is incremented. Also, the bit from the LSB of the quotient is shifted into the first guard bit of the round register.

After the normalization preparation sequence the quotient goes through the mantissa overflow sequence if it has the wrong sign, or it may go through normalization. There is only one case in division where the quotient has to be normalized: mantissa 1/2 divided by -1 resulting with -1/2 or in binary, 1.1000. Remember that in the multiply/divide termination sequence one extra quotient bit was loaded into the first guard bit. In this case, where the quotient is normalized (shifted) left, the extra quotient bit should be shifted into the least significant bit position of AMAN. However, since the extra quotient bit in this case is zero, and since the bits to the right of the LSB position of AMAN are cleared during execution begin, zero is shifted into the LSB of the quotient. Thus, in the case of 1/2 divided by -1, the result of -1/2 is properly normalized to -1.

Once the quotient mantissa has the proper sign and is normalized, it may be rounded. Division has its own rounding rules. Namely, if the first guard bit of the quotient is a one, then the quotient is rounded. This first guard bit receives the extra quotient bit developed at the multiply/division termination sequence. Also, during the mantissa overflow sequence the LSB of AMAN may be shifted into the first guard bit. The rounding decision is based on the first guard bit contents at the time of the round decision states. As in other floating point operations, rounding may cause mantissa overflow which would have to be corrected in the mantissa overflow sequence.

After the rounding process, the quotient exponent is checked to see if it is in range. If it is not in range, the final quotient is the overflow or underflow constant. In any case, the final quotient is loaded into the A, B, C and output registers.

TABLE III-3. SUMMARY OF DIVISION

DIVISION - LOAD SEQUENCE:
Schematic Reference 12740-60002, page 53

The first operand is the dividend which is loaded into A mantissa and exponent. During the division process A holds the partial remainder. The second operand is the divisor and is loaded into B mantissa and exponent. The contents of B mantissa do not change during the division process. The C register will hold the developing quotient.

Execution begin control state:

Set A & C mantissa registers to shift left mode

MDY/DIV initialization:
Load MPY/DIV counter
Set ALU/SHIFT decision register to ALU (DADSB high)
Load quotient sign register with [A(55) XOR B(55)]

Exponent sequence: Load (AEXP-BEXP) into both exponent registers

Division decision equations: Text reference; The Logic Of Computer Arithmetic by Ivan Flores, chapter 13.

1. Decision to go to shift cycle or ALU cycle
   If A(55) is not equal to A(54), do ALU cycle, DADSB activated

   If A(55) is not equal to A(54), do shift cycle, DIVSH activated

   Since a shift operation accompanies every cycle, the FPP bases its decision on (A(54) XOR A(53)) before the shift.

2. Decision to add or subtract divisor from partial product during ALU cycle.

   (A(55) XOR B(55)):
   If one (A(55) is not equal to B(55)), then ADD, DIVADD- activated
   If zero (A(55)=B(55)), then SUBTRACT, DIVSUB- activated

3. Quotient bit determination, DIVCO

   Next quotient bit = $\overline{(\text{ALU}(55) \text{ XOR } \text{B}(55))}$ during ALU operation
   = (A(55) XOR B(55)) during shift operation

4. Carry out division until one extra quotient bit is formed (held in LASTDIVCO), but is not shifted into C.

MDTERMINATION SEQUENCE:

Route the quotient from the C register to the A mantissa register. Load LASTDIVCO into the first guard bit of the round register.

If the sign of A does not equal the quotient sign, then go through mantissa overflow sequence.

If the first guard bit is a one, then go through the rounding sequence. After rounding, check for mantissa overflow.

If the exponents overflowed or underflowed, go through that sequence.

TERMINATION SEQUENCE:

Load result in all registers. Set ready/busy FF to ready.

SCHEMATIC SUMMARY III-4

DIVISION - EXPLANATION OF EXEC2 STATE MACHINE
Schematic reference 12740-60002 page 53

PWRST- : Exponent ALU is in subtract mode.

Execution begin control state actions:

Set ARGSO low, ABRGS1 high, CRGSO low, CRGS1 high - A & C mantissa registers are in shift left mode
MPY/DIV counter load flip-flop (U41-8,9,10,11; U83-8,9,10) in load state
Exponent register in load state (EXPLD- is low)

MPY/DIV Initialization state:

Set DADSB/DIVSH decision register (U23-1,2,3,4,5) to DADSB (ALU cycle)
Clock quotient sign register (U23,10,11,12,13) to equal A(55) XOR B(55)
Clock loading of MPY/DIV counter (U11,U21)
Enter both MDEXPCTL1 and MDEXPCTL2
After MDEXPCTL1, pass to ALU cycle, since first divide cycle is always ALU

MDEXPCTL2:

MPY/DIV counter load flip-flop (U41-8,9,10,11; U83-8,9.10) in count state

MDEXPCTL5:

CKAEXP, CKBEXP - clock exponent difference into AEXP, BEXP

MDEXPCTL6:
    Set exponent register to count mode (EXPLD- low)

MDSHIFTCTL2-:
    When coming from MDEXPCTL1, the first divide decision is to go thru ALU cycle, so that this state acts as delay. Otherwise, this state forms the decision point for entry into an ALU subcycle or a shift subcycle.

DIVISION - SHIFT CYCLE

MDSHIFTCTL1:
    CRGCK- shift CMAN left while shifting in new quotient bit.
    ARGCK- shift AMAN (which holds partial remainder) left
    ROUNDCLOCK- shift round register left (meaningless in divide, since round register should = 0 in divide)
    Clock division decision register (U23-1,2,3,4,5,6) to activate DADSB (ALU subcycle) or DIVSH (shift cycle)
    If DADSB is active:
        MNTSO high (mantissa ALU add), if A(55) is not equal to B(55)
        MNTSO low (mantissa ALU sub), if A(55)=B(55)
        Decrement MPY/DIV counter (U11,U21)

MDSHIFTCTL2:
    If DIVOP:DIVSH:CNTZERO-, enter shift cycle
    If DADSB, enter ALU cycle
    If DIVSH:CNTZERO+, enter divide termination

DIVISION - ALU CYCLE

MDALUCTL1:
    ARGSO and ABRGS1 high, set AMAN to load mode

MDALUCTL4:
    ARGCK: Clock ALU result into AMAN
    Clock mantissa overflow register (U13-1,2,3,4,5,6)
    DIVSHIFTLT-: ARGSO low - set AMAN to shift left mode
    Clock quotient bit register

MDALUCTL5:
    If CNTZERO-:MPYSH (in divide, MPYSH always active) enter shift cycle
    If CNTZERO+, enter MPY/DIV termination sequence

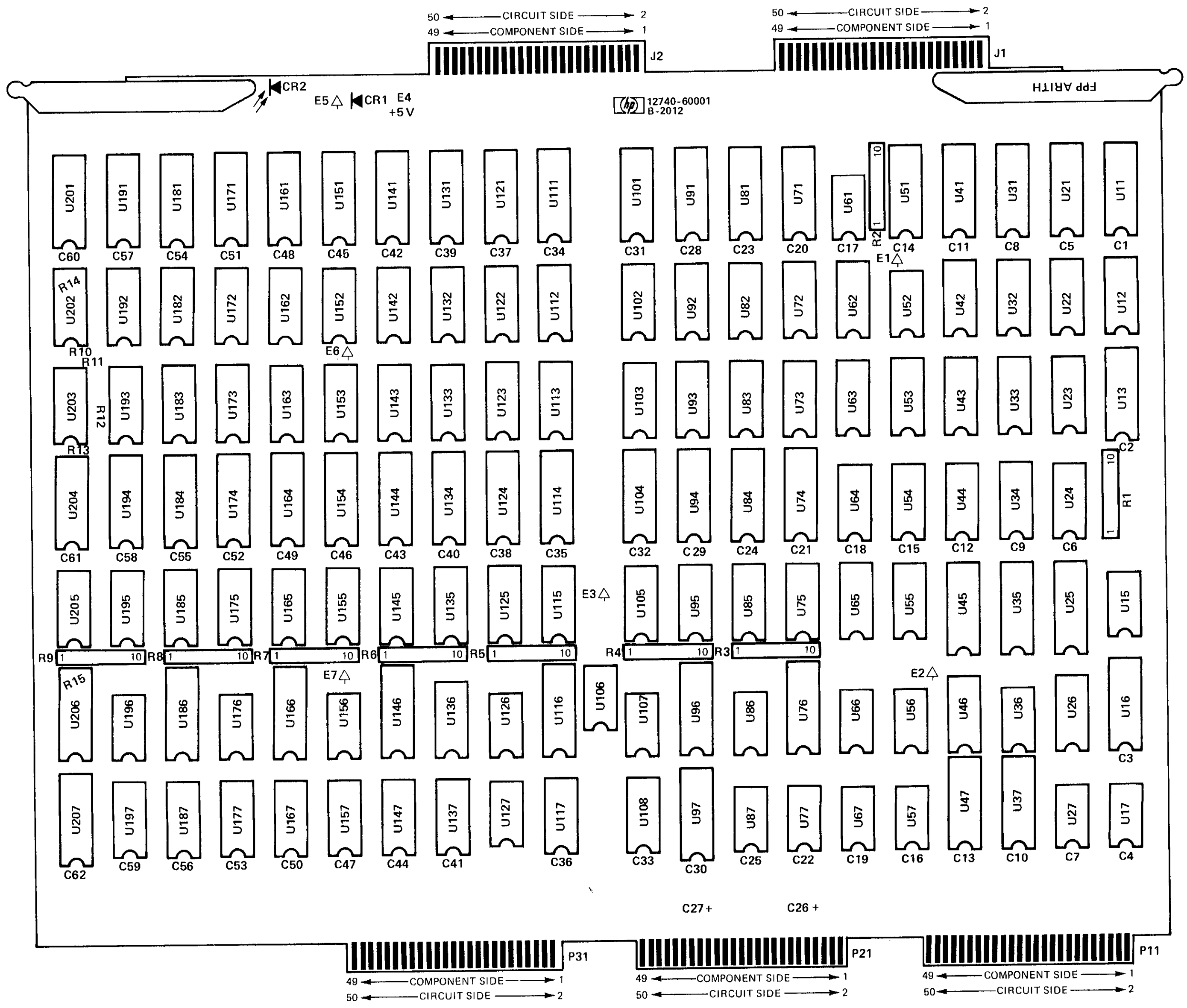
DIVIDE TERMINATION SEQUENCE

MDTERM1:
DIVTERM1-: MALUEN+ low, CRGOEN+ high - CMAN register drives P-bus

CRGOEN+ enables LASTDIVCO (last quotient bit)
CRGOEN- lowers CRGSO (CMAN is not in load mode)
ARGSO and ABRGS1 high - AMAN & round register are in load mode

MDTERM4:
ARGCK: load P-bus data (quotient) in AMAN register
ROUNDCLOCK: load LASTDIVCO (last quotient bit) in round register
CRGOEN+: CMAN does not drive P-bus
MALUEN+: mantissa ALU drives P-bus
MIDTERM4- used on Schematic page 52 at U163-4 in multiply, meaningless in divide

MDTERM2: go to normalization preparation sequence



FPP Arithmetic Card Assembly 12740-60001

FPP Arithmetic Card Assembly Parts List (12740-60001) Sht. 1 of 3

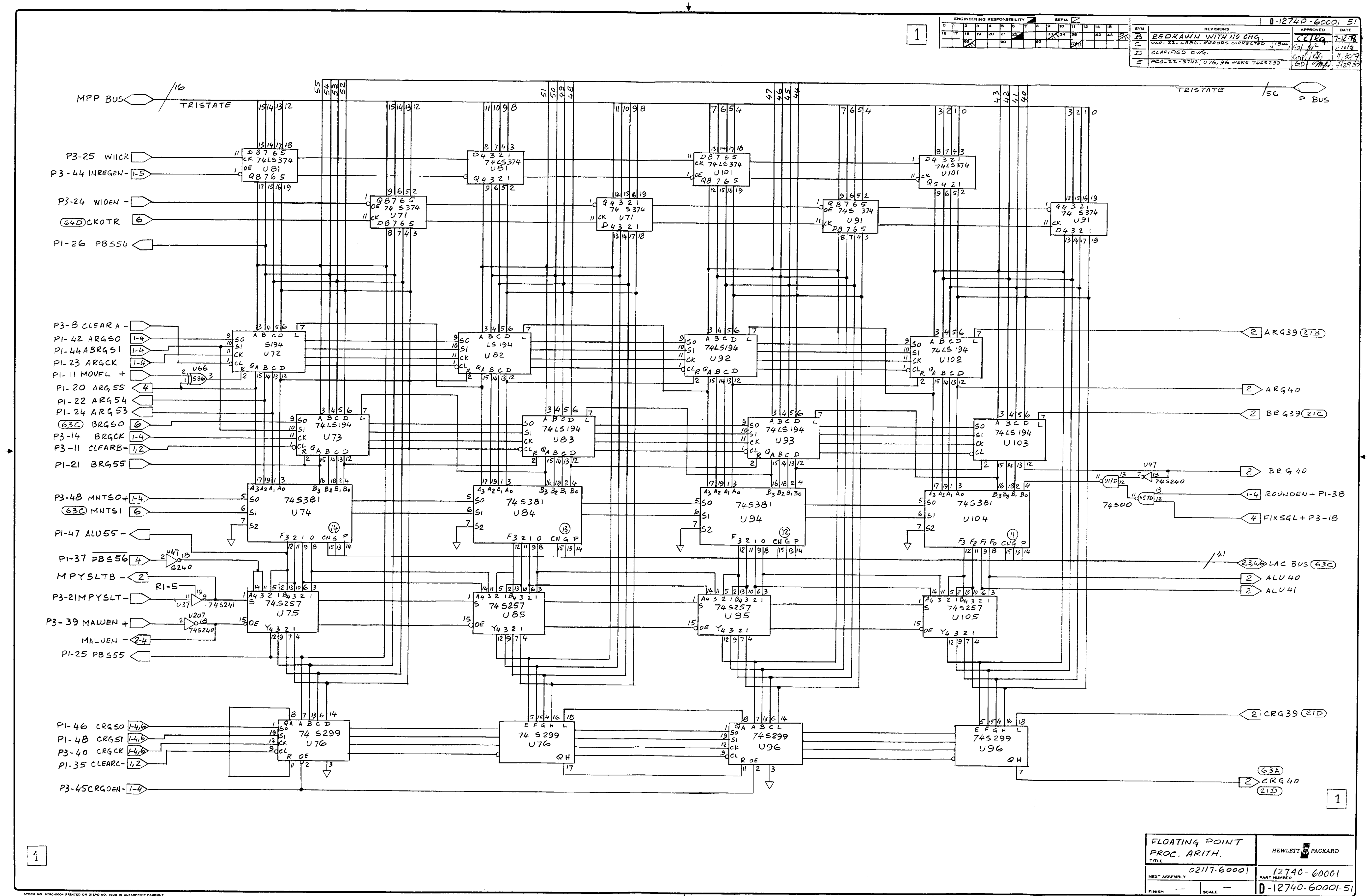
ITEM NO.	REFERENCE DESIGNATOR (FIRST SIX)	PART DESCRIPTION	PARENT OPTION	PART NUMBER	COMP. OPTION	LOC	QUANTITY PER
01	C26,27	CAP 120UF 10%		0180-2145		U	2
00	E1-7	TERM-STUD SGL		0360-1682		U	7
		SCR TAP 4-40X.31		0624-0077		U	4
01	R15	RES 1.0K 5% .25		0683-1025		U	1
01	R12,14	RES 147 1%.125		0698-3438		D	2
		PIN GRV .062X.25		1480-0116		U	2
00	R1-9	NTWK RES 9X1K		1810-0275		U	9
00	U15	IC SN7417N		1820-0618		U	1
01	U17,56,57,67	IC SN74S00N		1820-0681		U	4
01	U27,61	IC SN74S10N		1820-0685		U	2
01	U106	IC SN74S74N		1820-0693		U	1
00	U66	IC SN74S86N		1820-0694		U	1
01 02	U62,108,147,157,167 U187,197	IC SN74S153N		1820-0998		U	7
01	U136,137	IC SN74S133N		1820-1130		U	2
00	U36	IC SN74S51N		1820-1158		U	1
01 03	U86,107,126,156,176 196	IC SN74S260N		1820-1275		U	6
		IC 74LS194		1820-1276		U	27

FPP Arithmetic Card Assembly Parts List (12740-60001) Sht. 2 of 3

ITEM NO.	REFERENCE DESIGNATOR (FIRST SIX)	PART DESCRIPTION	PARENT OPTION	PART NUMBER	COMP. OPTION	LOC	QUANTITY PER
				1820-1276			
01	U73,82	,83,92,93,					
03	102,1	03,112,113,122					
05	123,1	32,133,142					
07	143,1	52,153,162,163					
09	172,1	73,182,183					
11	192,1	93,202,203					
		IC SN74S257N		1820-1301		U	14
01	U75,85	,95,105,115,					
03	125,1	35,145,155,165					
05	175,1	85,195,205					
		IC SN74S194N		1820-1304		U	4
01	U63,72	,117,177					
		IC SN74S182N		1820-1305		U	5
01	U53,54	,55,64,65					
		IC SN74S151N		1820-1319		U	1
00	U46						
		IC SN74S85N		1820-1321		U	5
01	U12,24	,26,34,44					
		IC SN74S02N		1820-1322		U	3
00	U52,77	,87					
		IC SN74S30N		1820-1323		U	1
00	U127						
		IC SN74S169N		1820-1455		U	6
01	U22,23	,32,33,42,43					
		SN74S299N		1820-1457		U	7
01	U76,96	,116,146,166,					
03	186,2	06					
		IC SN74S381N		1820-1458		U	17
01	U25,35	,45,74,84,94					
03	104,1	14,124,134,144					
05	154,1	64,174,184,194					
07	204						
		ICSN74S241N		1820-1624		U	3
01	U13,37	,51					
		IC SN74S240N		1820-1633		U	4
01	U16,47	,97,207					
		IC SN74S374N		1820-1677		U	10
01	U21,31	,41,71,91,111,					
03	131,15	1,171,191					
		IC SN74LS374PC		1820-1997		U	8

FPP Arithmetic Card Assembly Parts List (12740-60001) Sht. 3 of 3

ITEM NO.	REFERENCE DESIGNATOR (FIRST SIX)	PART DESCRIPTION	PARENT OPTION	PART NUMBER	COMP. OPTION	LOC	QUANTITY PER
01 03	U11,81 161,181	,101,121,141, 1,201		1820-1997			
01	CR1	DIODE SILICONE		1901-0463		U	1
00	CR2	LED-V SEN		1990-0581		U	1
		EXTRACTOR PC		5040-6009		W	2
		BRACE-PC BOARD		5040-6058		W	1
		ASSY-COMP SEG		12740-64001		A	1
		BOARD-ETCHED		12740-80001		W	1
01	TEST	DTS70 TEST ADAPT FIXTURE		ET13472		1	0
01	TEST	CABLE ASSEMBLY FIXTURE		ET13472-6002		1	0



SYM	REVISIONS	APPROVED	DATE
B	REDRAWN WITH NO CHG.	[illegible]	7-12-78
C	PCO-22-4886-ERRORS CORRECTED (1844)	[illegible]	[illegible]
D	CLARIFIED DWG.	[illegible]	11/30/79
E	PCO-22-3742; U76,96 WERE 74LS299	[illegible]	[illegible]

D-12740-60001-51

FLOATING POINT PROC. ARITH.

TITLE

HEWLETT PACKARD

NEXT ASSEMBLY 02117-60001

PART NUMBER 12740-60001

FINISH — SCALE —

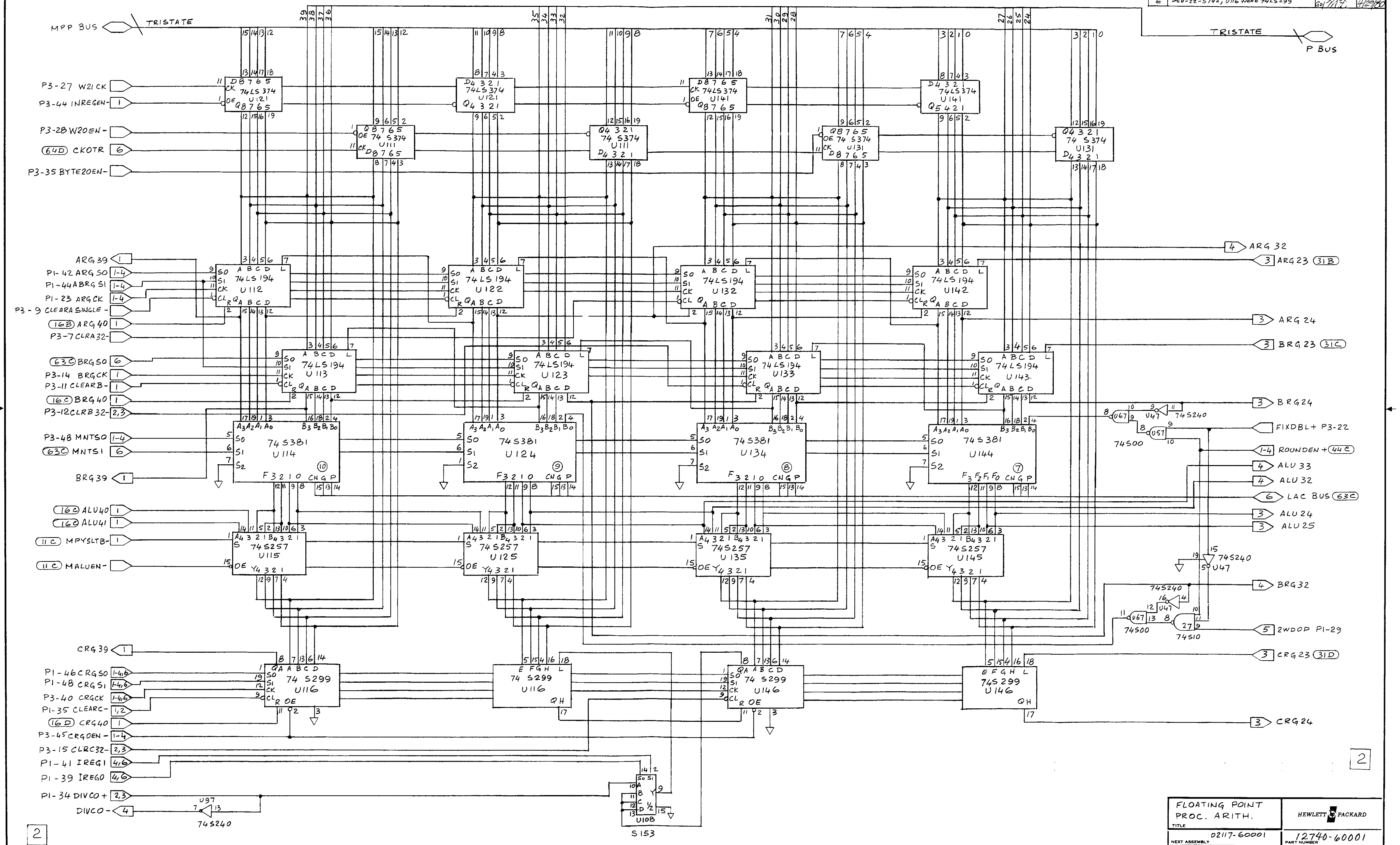
D-12740-60001-51

SHEET 1 OF 1

STOCK NO. 9280-0004 PRINTED ON DIEPO NO. 1020-10 CLEARPRINT FADEOUT

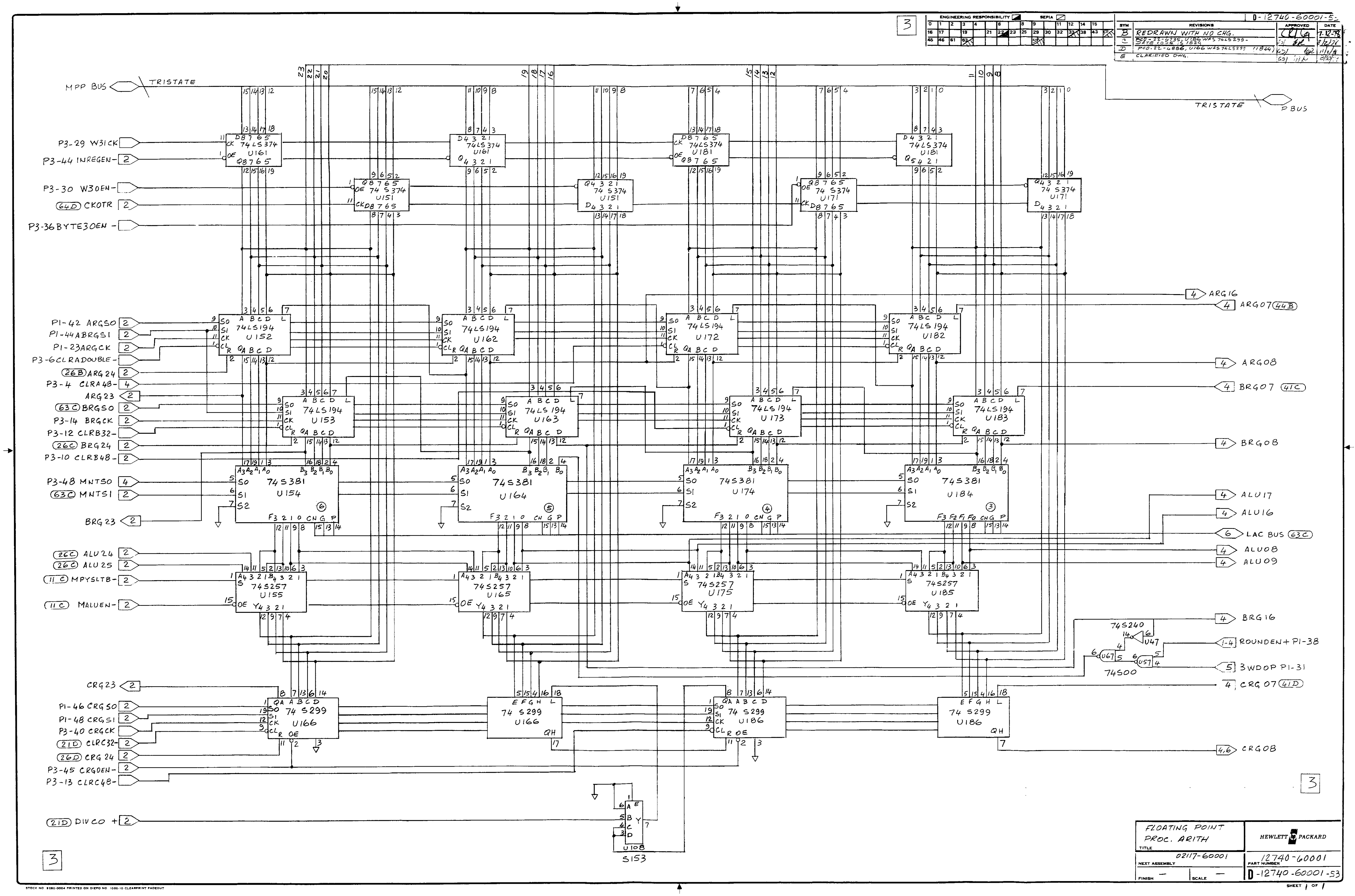
SYM	REVISIONS	APPROVED	DATE
B	REDRAWN WITH NO CHG.	CR	7-12-74
C	PCO-22-4886-ERRORS CORRECTED (1844)	[illegible]	1-2-75
D	CLARIFIED DWG.	[illegible]	8/30/77
E	PCO-22-5742; U116 WERE 74LS299	[illegible]	4/29/80

D-12740-60001



FLOATING POINT PROC. ARITH.
TITLE
NEXT ASSEMBLY 02117-60001
HEWLETT PACKARD
12740-60001
PART NUMBER
FINISH — SCALE —
D-12740-60001-52
SHEET 1 OF 1

STOCK NO. 9280-0004 PRINTED ON DIEPO NO. 1020-10 CLEARPRINT FADEOUT



3

ENGINEERING RESPONSIBILITY ◩ SEPIA ◸

0	1	2	3	4		6		8	9		11	12	14	15	
16	17		19		21	22	23	25	29	30	32	33	38	43	50
45	46	61	63						50						

D-12740-60001-5

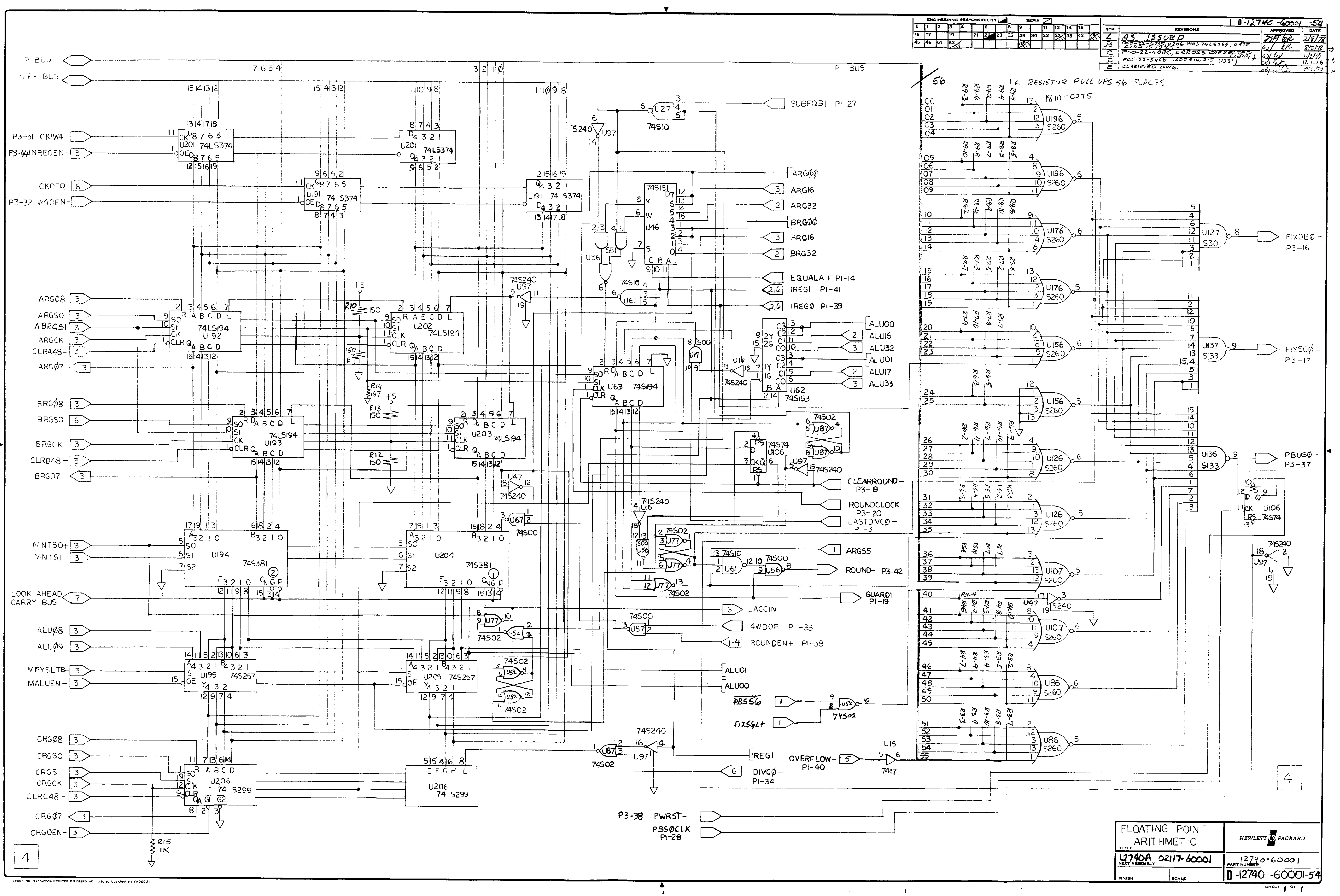
SYM	REVISIONS	APPROVED	DATE
B	REDRAWN WITH NO CHG.	[illegible]	7-12-78
C	PCO-22-4735, U186 WAS 74LS299 - DATE CODE IS 1829	[illegible]	8/2/78
D	PCO-22-4886, U166 WAS 74LS299 (1844)	[illegible]	11/2/78
E	CLARIFIED DWG.	[illegible]	8/30/[illegible]

3

3

FLOATING POINT PROC. ARITH TITLE	HEWLETT PACKARD
02117-60001 NEXT ASSEMBLY	12740-60001 PART NUMBER
FINISH — SCALE —	D-12740-60001-53

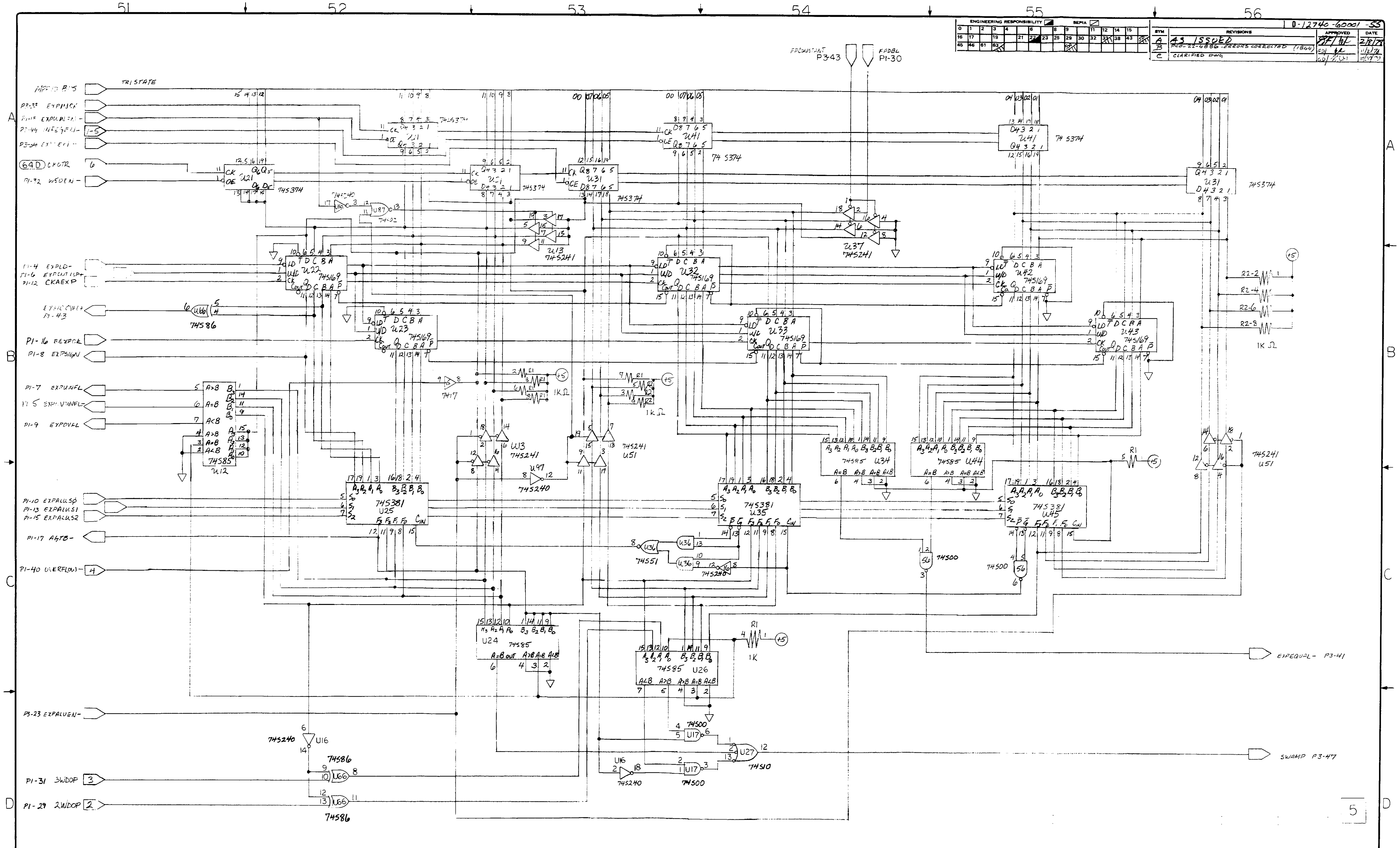
SHEET 1 OF 1



SYM	REVISIONS	APPROVED	DATE
A	AS ISSUED	[illegible]	2/8/78
B	PCO-22-4735, U206 WAS 74LS299, DATE CODE IS 1829-	[illegible]	8/2/78
C	PCO-22-4886, ERRORS CORRECTED (1844)	[illegible]	11/7/78
D	PCO-22-5408 .ADD R14, R15 (1931)	[illegible]	12-1-78
E	CLARIFIED DWG.	[illegible]	[illegible]

D-12740-60001 -54

FLOATING POINT ARITHMETIC
TITLE
12740A 02117-60001
NEXT ASSEMBLY
HEWLETT PACKARD
12740-60001
PART NUMBER
D-12740-60001-54
FINISH
SCALE
SHEET 1 OF 1



SYM	REVISIONS	APPROVED	DATE
A	AS ISSUED	[illegible]	2/8/78
B	PCO-22-4886 - ERRORS CORRECTED (1844)	[illegible]	[illegible]
C	CLARIFIED DWG	[illegible]	[illegible]

D-12740-60001-55

EXPONENT
TITLE ARITHMETIC

HEWLETT PACKARD

12740A 02117-60001
NEXT ASSEMBLY

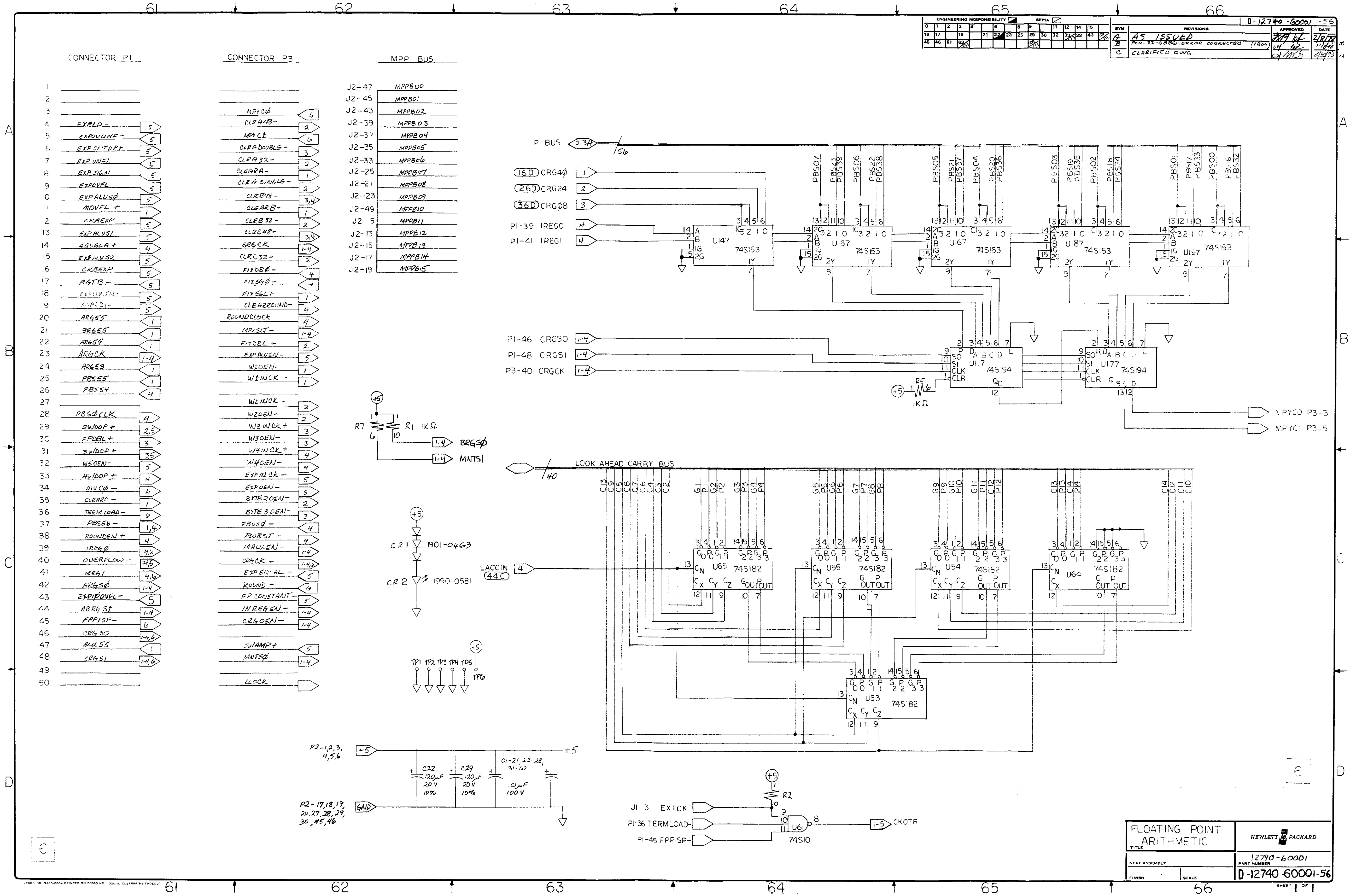
12740-60001
PART NUMBER

FINISH
SCALE

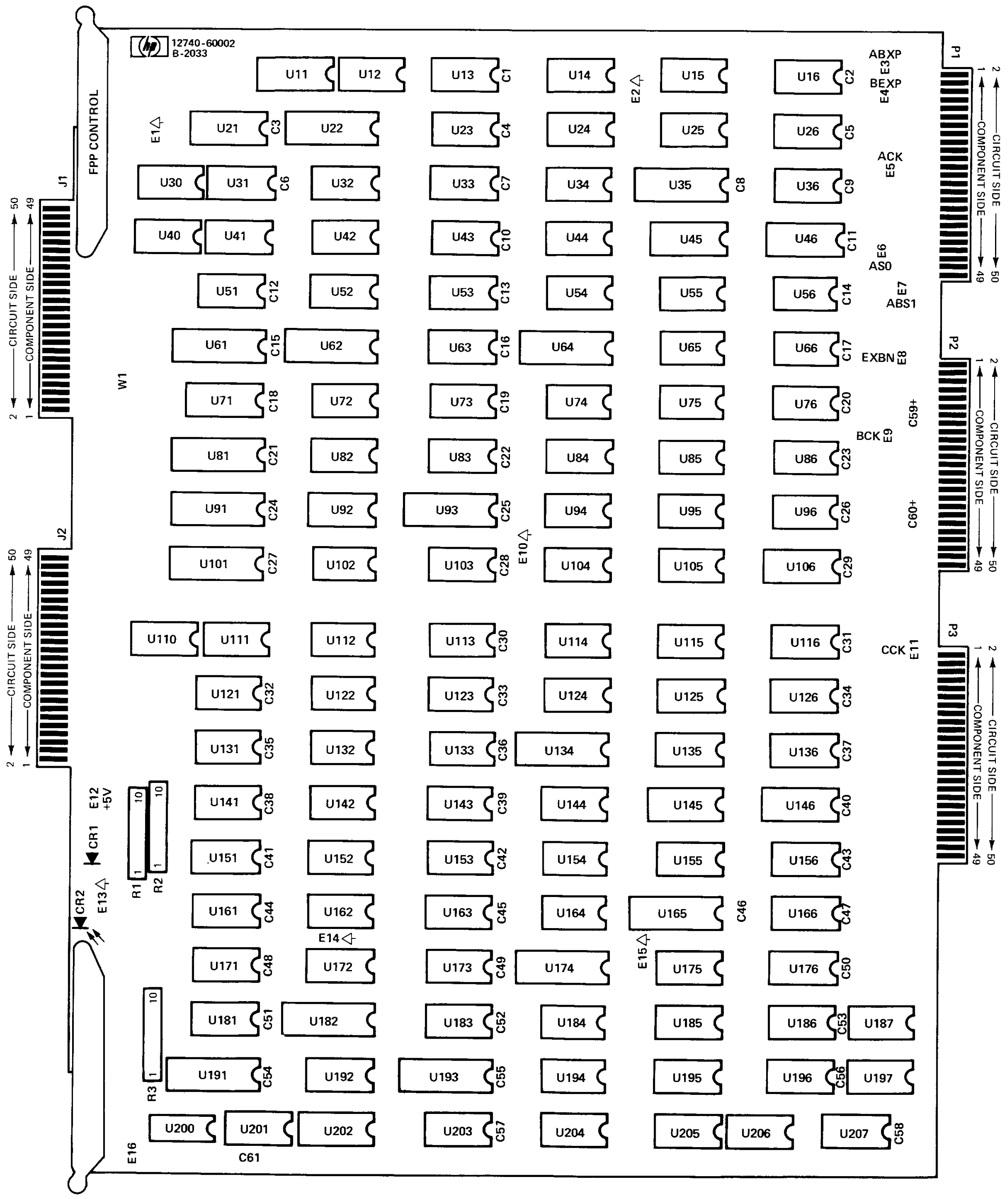
D-12740-60001-55

SHEET 1 OF 1

5



CONNECTOR P1
CONNECTOR P3
MPP BUS
P BUS
LOOK AHEAD CARRY BUS
U117 74S194
U177 74S194
U147 74S153
U157 74S153
U167 74S153
U187 74S153
U197 74S153
U65 74S182
U55 74S182
U54 74S182
U64 74S182
U53 74S182
U61 74S10
CR1 1901-0463
CR2 1990-0581
R1 1KΩ
R7
R5 1KΩ
R2
BRGSØ
MNTS1
LACCIN
MPYCØ P3-3
MPYC1 P3-5
J1-3 EXTCK
P1-36 TERMLOAD
P1-45 FPPISP
CKOTR
P2-1,2,3,4,5,6
P2-17,18,19,20,27,28,29,30,45,46
GND
C22 120μF 20V 10%
C29 120μF 20V 10%
C1-21,23-28,31-62 .01μF 100V
TP1 TP2 TP3 TP4 TP5 TP6
FLOATING POINT ARITHMETIC
HEWLETT PACKARD
12740-60001
D-12740-60001-56
A AS ISSUED
B PCO-22-4886 ERROR CORRECTED
C CLARIFIED DWG.


FPP Control Card Assembly - 12740-60002

FPP Control Card Assembly Parts List (12740-60002) Sht. 1 of 3)

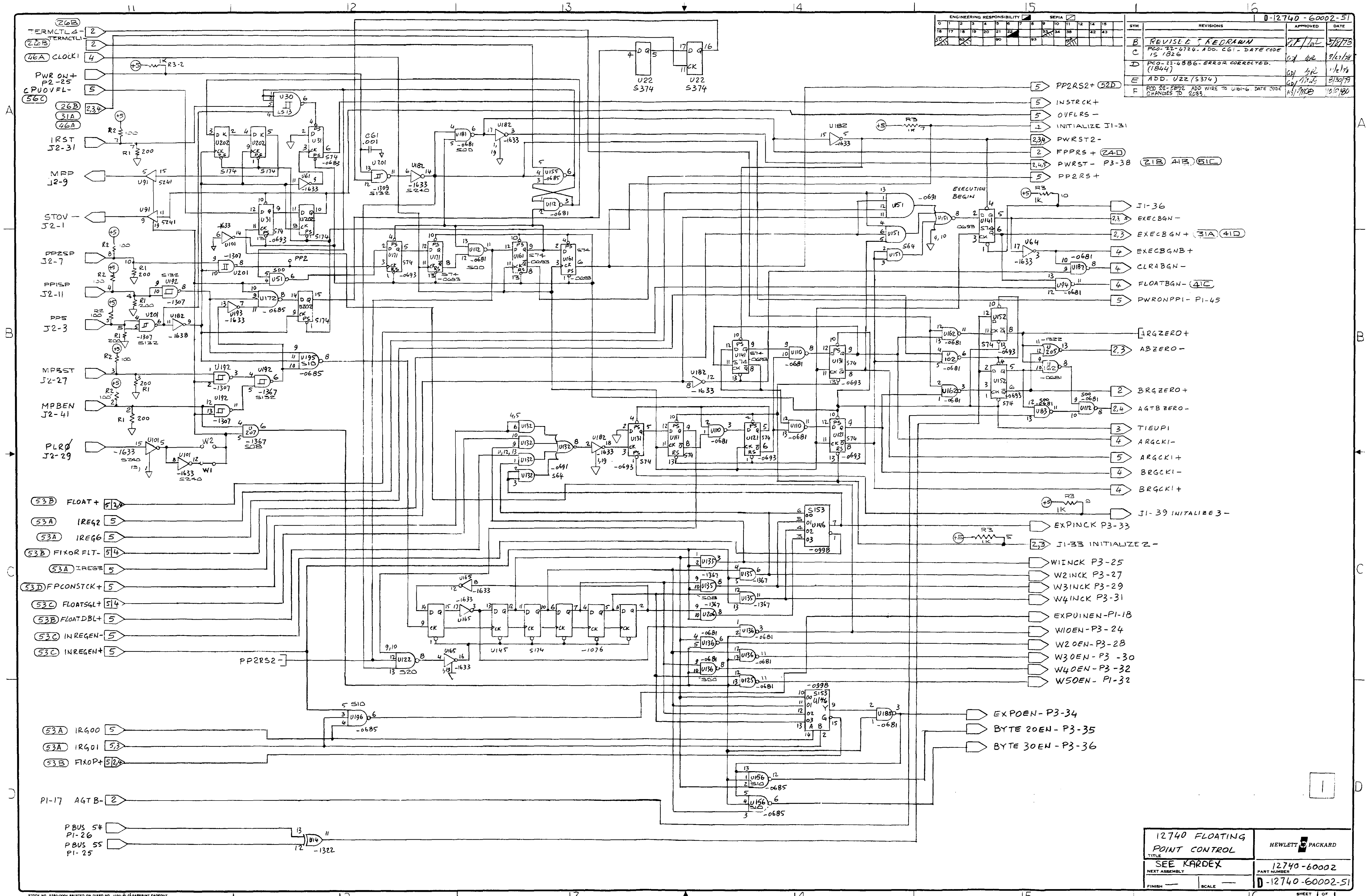
ITEM NO.	REFERENCE DESIGNATOR (FIRST SIX)	PART DESCRIPTION	PARENT OPTION	PART NUMBER	COMP. OPTION	LOC	QUANTITY PER
01	C1-58	CAP .01UF		0160-2055		U	58
01	C61	CAPACITOR .001MF		0160-3448		U	1
01	C59,60	CAP 120UF 10%		0180-2145		U	2
00	E1-16	TERM-STUD SGL		0360-1682		U	16
		SCR TAP 4-40X.31		0624-0077		U	4
		PIN GRV .062X.25		1480-0116		U	2
01	R1	NTWK RES 9X200		1810-0271		U	1
00	R3,	NTWK RES 9X1K		1810-0275		U	1
01	R2	NETWORK-RES SIP		1810-0386		U	1
01	U200	OSCILLATOR 40MHZ		1813-0119		U	1
		IC SN74S00N		1820-0681		U	20
01	U16,33	,51,54,55,74					
03	U75,83	,94,102,110,					
05	U112,1	25,136,162,166					
07	U181,1	85,187,203					
		IC SN74S10N		1820-0685		U	13
01	U15,36	,41,56,63,92,					
03	U95,15	5,156,172,					
05	U176,1	95,196,					
00	U84	IC SN74S11N		1820-0686		U	1
		IC SN74S20N		1820-0688		U	4
01	U26,34	,122,142,					
00	U86	IC SN74S40N		1820-0690		U	1
		IC SN74S64N		1820-0691		U	12

FPP Control Card Assembly Parts List (12740-60002) Sht. 2 of 3

ITEM NO	REFERENCE DESIGNATOR (FIRST SIX)	PART DESCRIPTION	PARENT OPTION	PART NUMBER	COMP. OPTION	LOC	QUANTITY PER
				1820-0691			
01	U73,11	3,123,132,133					
03	143,1	51,153,163,173					
05	U183,1	86					
		IC SN74S74N		1820-0693		U	25
01	U13,23	,31,32,42,					
03	52,53,	72,82,105,111,					
05	114,12	1,124,131,141,					
07	144,15	2,154,161,164,					
09	171,18	4,194,204					
		IC SN74S86N		1820-0694		U	3
01	U14,25	,197					
		IC SN74S153N		1820-0998		U	1
01	U146,						
		IC SN74S174N		1820-1076		U	2
01	U145,2	02					
		IC SN74S133N		1820-1130		U	3
01	U45,46	,106					
		IC SN74S51N		1820-1158		U	4
01	U12,43	,103,115					
		IC SN74S138N		1820-1240		U	1
01	U71						
		IC SN74S132N		1820-1307		U	2
01	U192,2	01					
		IC SN74S02N		1820-1322		U	4
01	24,40,	104,205					
		IC SN74S30N		1820-1323		U	5
01	U44,66	,76,85,96					
		IC SN74S08N		1820-1367		U	7
01	U65,11	6,126,135,					
03	U175,2	06,207,					
		IC SN74LS13N		1820-1415		U	1
01	U30						
		IC SN74S169N		1820-1455		U	2
01	U11,21	,					
		IC SN74273N		1820-1461		U	1
01	U81						
		ICSN74S241N		1820-1624		U	3
01	U35,91	,191					

FPP Control Card Assembly Parts List (12740-60002) Sht. 3 of 3

ITEM NO.	REFERENCE DESIGNATOR FIRST SIX	PART DESCRIPTION	PARENT OPTION	PART NUMBER	COMP. OPTION	LOC	QUANTITY PER
		IC SN74S240N		1820-1633		U	7
01	U61,64	,93,101,165,					
03	182,1	93					
		IC SN74S374N		1820-1677		U	4
01	U22,62	,134,174					
		DIODE SILICONE		1901-0463		U	1
01	CR1						
		LED-V SEN		1990-0581		U	1
00	CR2						
		WIRE JUMPERS		8159-0005		D	1
01	W1						
		EXTRACTOR PC		5040-6009		W	2
		BRACE-PC BOARD		5040-6058		W	1
		PC BD-ETCHED		12740-80002		W	1
		DTS70 TEST ADAPT		ET13472		1	0
01	TEST	FIXTURE					
		CABLE ASSEMBLY		ET13472-6002		1	0
01	TEST	FIXTURE					



SYM	REVISIONS	APPROVED	DATE
B	REVISE & REDRAWN	[illegible]	5/8/78
C	PCO-22-4724- ADD. C61 - DATE CODE IS 1826	[illegible]	7/27/78
D	PCO-22-4886- ERROR CORRECTED. (1844)	GD/ [illegible]	11/2/78
E	ADD. U22 (S374)	GD/ [illegible]	3/30/79
F	PCD 22-5892 ADD WIRE TO U161-6. DATE CODE CHANGES TO 2033.	AY/ [illegible]	10/9/80

D-12740-60002-51

TITLE: 12740 FLOATING POINT CONTROL

HEWLETT PACKARD

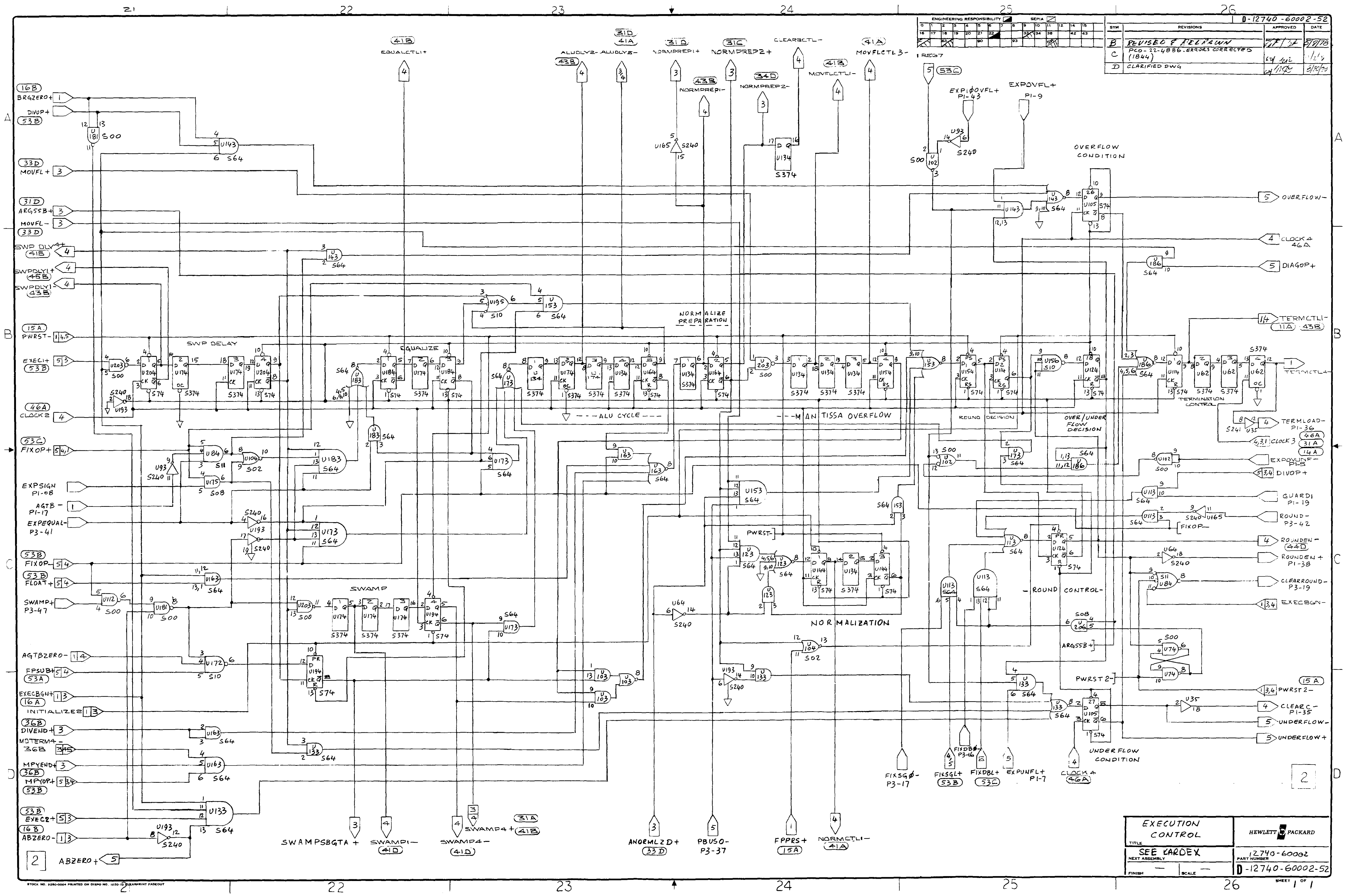
NEXT ASSEMBLY: SEE KARDEX

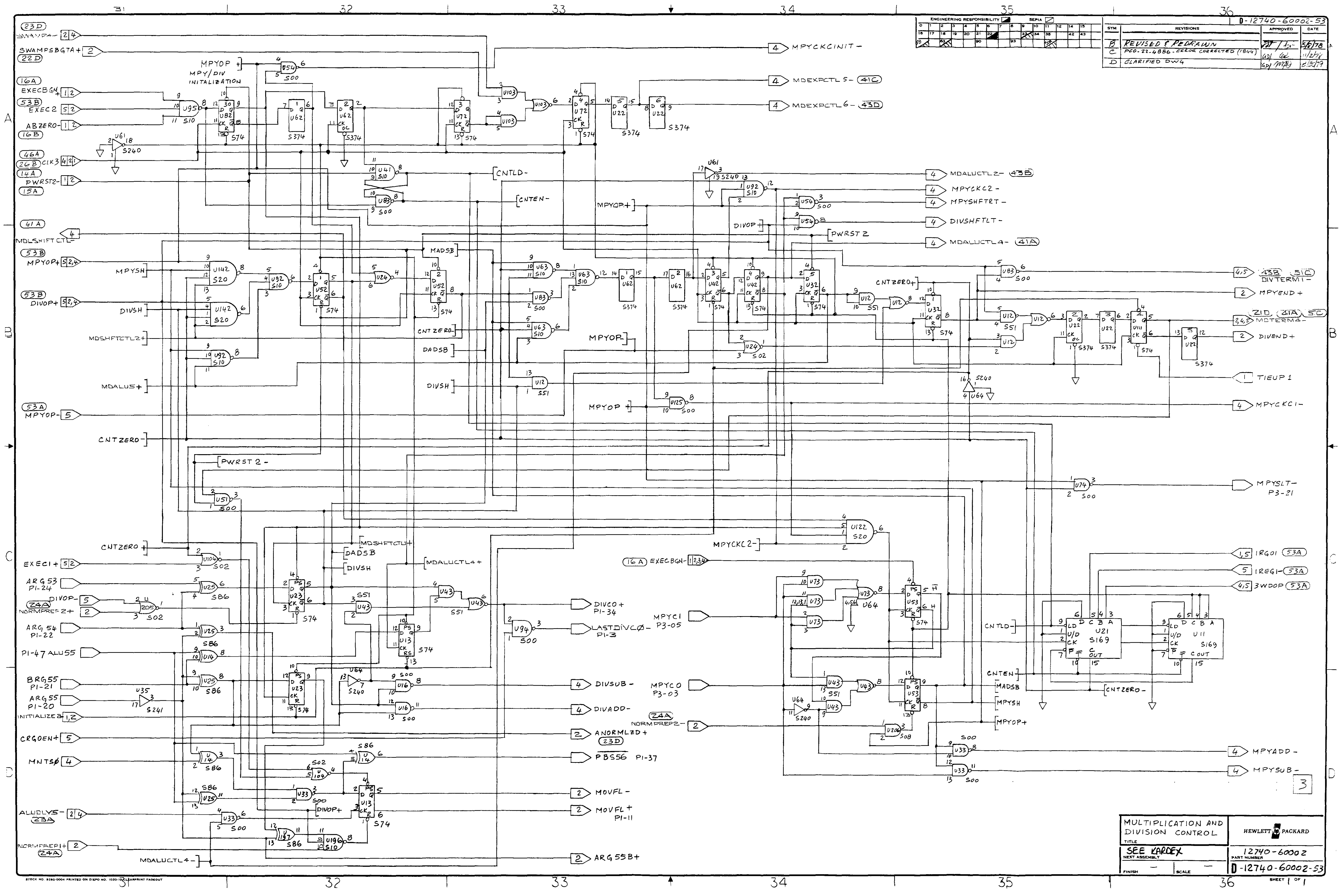
PART NUMBER: 12740-60002

FINISH — SCALE —

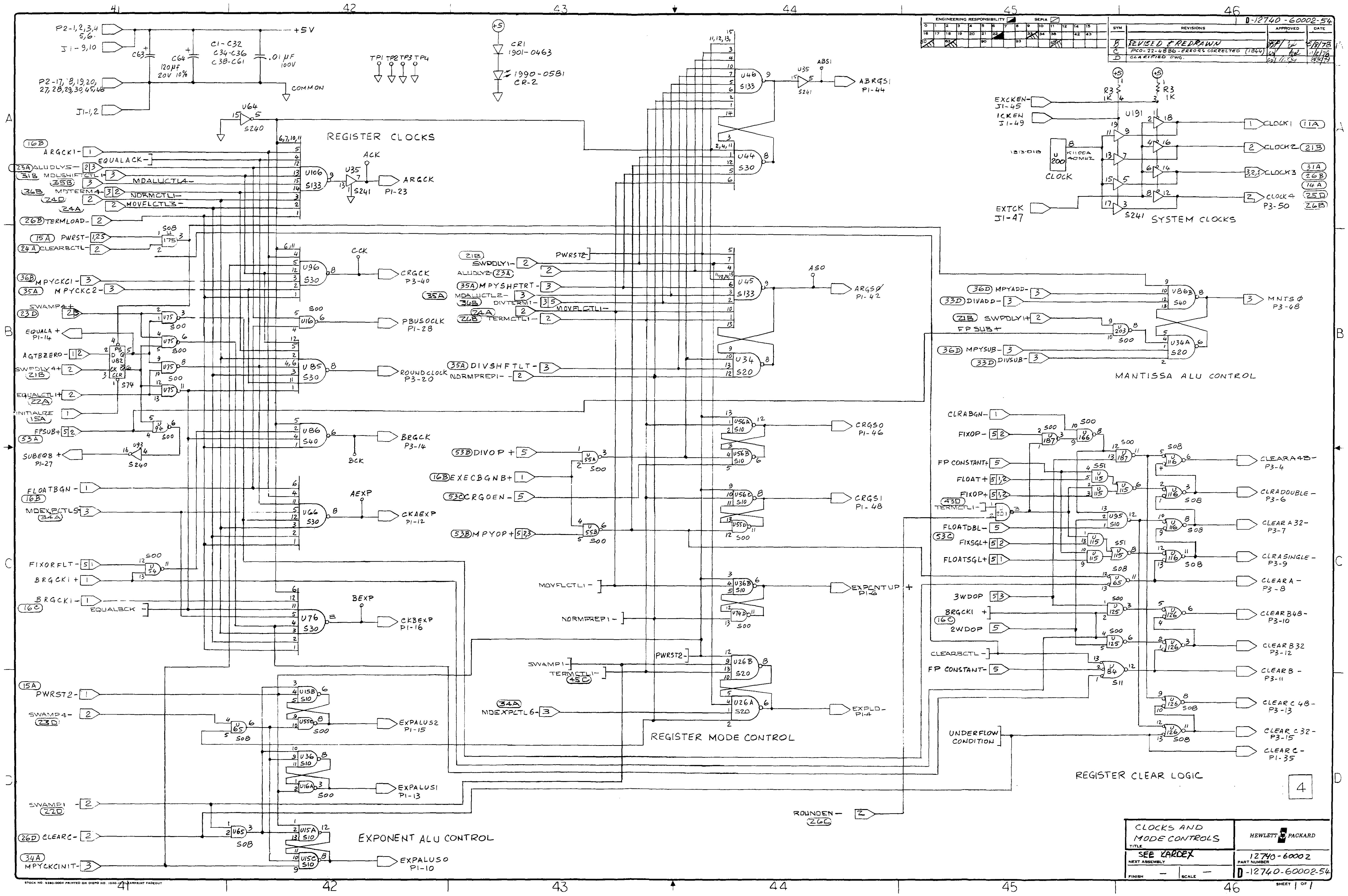
D-12740-60002-51

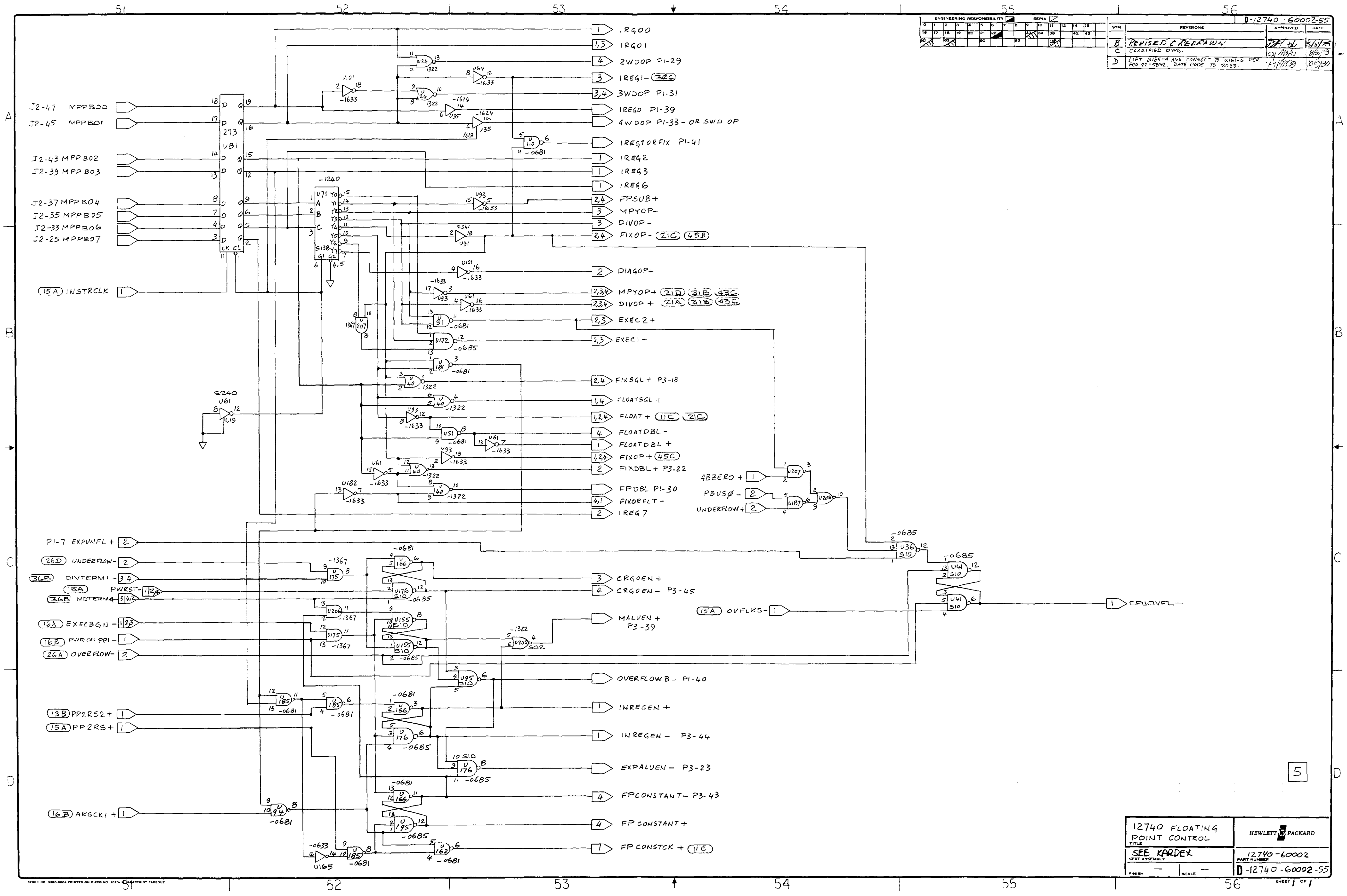
SHEET 1 OF 1


D-12740-60002-52
SYM | REVISIONS | APPROVED | DATE
B | REVISED & REDRAWN | | 5/8/78
C | PCO-22-4886. ERRORS CORRECTED (1844)
D | CLARIFIED DWG | | 8/30/79
ENGINEERING RESPONSIBILITY
SEPIA
BRGZERO+
DIVOP+
MOVFL+
ARGSSB+
MOVFL-
SWP DLY
SWPDLY1+
SWPDLY1-
PWRST-
EXECI+
CLOCK2
FIXOP+
EXPSIGN P1-08
AGTB- P1-17
EXPEQUAL- P3-41
FIXOP-
FLOAT+
SWAMP+ P3-47
AGTBZERO-
FPSUB+
EXECBGN+
INITIALIZE
DIVEND+
MPTERM4-
MPYEND+
MPYOP+
EXEC2+
ABZERO-
ABZERO+
SWP DELAY
EQUALIZE
NORMALIZE PREPARATION
ALU CYCLE
MANTISSA OVERFLOW
ROUND DECISION
OVER/UNDER FLOW DECISION
TERMINATION CONTROL
OVERFLOW CONDITION
SWAMP
NORMALIZATION
ROUND CONTROL
UNDERFLOW CONDITION
EQUALCTL1+
ALUDLY2-
ALUDLY5-
NORMPREP1+
NORMPREP2+
CLEARBCTL-
MOVFLCTL3-
NORMPREP1-
NORMPREP2-
MOVFLCTL1-
1 REG7
EXP1ØOVFL+ P1-43
EXPOVFL+ P1-9
OVERFLOW-
CLOCK4
DIAGOP+
TERMCTL1-
TERMCTL4-
TERMLOAD- P1-36
CLOCK3
EXPOVUNF- P1-5
DIVOP+
GUARD1 P1-19
ROUND- P3-42
ROUNDEN-
ROUNDEN+ P1-38
CLEARROUND- P3-19
EXECBGN-
PWRST2-
CLEARC- P1-35
UNDERFLOW-
UNDERFLOW+
FIXSGØ- P3-17
FIXSGL+
FIXDBØ- P3-16
FIXDBL+
EXPUNFL+ P1-7
CLOCK4
SWAMPSBGTA+
SWAMP1-
SWAMP4-
SWAMP4+
ANORML2D+
PBUSO- P3-37
FPPRS+
NORMCTL1-
EXECUTION CONTROL
HEWLETT PACKARD
SEE CARDEX
NEXT ASSEMBLY
12740-60002
PART NUMBER
D-12740-60002-52
SHEET 1 OF 1



D-12740-60002-53
REVISED & REDRAWN
PCO-22-4886 ERROR CORRECTED (1844)
CLARIFIED DWG
SWAMPSBGTA+
EXECBGN
EXEC2
ABZERO-
CLK3
PWRST2-
MDLSHIFTCTL-
MPYOP+
DIVOP+
MPYOP-
CNTZERO
EXEC1+
ARG53 P1-24
DIVOP-
NORMPREP2+
ARG54 P1-22
P1-47 ALU55
BRG55 P1-21
ARG55 P1-20
INITIALIZE
CRGOEN+
MNTSØ
ALUDLYS-
NORMPREP1+
MPYOP MPY/DIV INITIALIZATION
CNTLD-
CNTEN-
MADSB
PWRST2-
MDSHFTCTL2+
MDALU5+
DADSB
DIVSH
MPYCKCINIT-
MDEXPCTL5-
MDEXPCTL6-
MDALUCTL2-
MPYCKC2-
MPYSHFTRT-
DIVSHFTLT-
MDALUCTL4-
DIVTERM1-
MPYEND+
MDTERM4-
DIVEND+
TIEUP1
MPYCKC1-
MPYSLT- P3-21
EXECBGN-
MPYC1 P3-05
MPYC0 P3-03
NORMPREP2-
IRG01
IREG1-
3WDOP
CNTLD
CNTEN
CNTZERO-
MPYADD-
MPYSUB-
DIVCO+ P1-34
LASTDIVCØ- P1-3
DIVSUB-
DIVADD-
ANORMLZD+
PBS56 P1-37
MOVFL-
MOVFL+ P1-11
ARG55B+
MDALUCTL4-
MULTIPLICATION AND DIVISION CONTROL
HEWLETT PACKARD
SEE KARDEX
12740-60002
D-12740-60002-53
SHEET 1 OF 1


REGISTER CLOCKS
SYSTEM CLOCKS
MANTISSA ALU CONTROL
REGISTER MODE CONTROL
REGISTER CLEAR LOGIC
EXPONENT ALU CONTROL
+5V
COMMON
C1-C32 C34-C36 C38-C61
.01 μF 100V
C64 120μF 20V 10%
C63
CR1 1901-0463
1990-0581 CR-2
TP1 TP2 TP3 TP4
ARGCK P1-23
CRGCK P3-40
PBUSOCLK P1-28
ROUNDCLOCK P3-20
BRGCK P3-14
CKAEXP P1-12
CKBEXP P1-16
EXPALUS2 P1-15
EXPALUS1 P1-13
EXPALUS0 P1-10
ABRGS1 P1-44
ARGS0 P1-42
CRGS0 P1-46
CRGS1 P1-48
EXPCNTUP+ P1-6
EXPLD- P1-4
MNTS0 P3-48
CLOCK1
CLOCK2
CLOCK3
CLOCK4 P3-50
CLOCK
1813-0118
EXTCK J1-47
EXCKEN- J1-45
ICKEN J1-49
CLEARA48- P3-4
CLRADOUBLE- P3-6
CLEARA32- P3-7
CLRASINGLE- P3-9
CLEARA- P3-8
CLEARB48- P3-10
CLEARB32 P3-12
CLEARB- P3-11
CLEARC48- P3-13
CLEARC32- P3-15
CLEARC- P1-35
UNDERFLOW CONDITION
ROUNDEN-
4
CLOCKS AND MODE CONTROLS
SEE KARDEX
12740-60002
D-12740-60002-54
HEWLETT PACKARD
SHEET 1 OF 1
REVISIONS
B REVISED & REDRAWN
C PCO-22-4886-ERRORS CORRECTED (1844)
D CLARIFIED DWG.



J2-47 MPPB00
J2-45 MPPB01
J2-43 MPPB02
J2-39 MPPB03
J2-37 MPPB04
J2-35 MPPB05
J2-33 MPPB06
J2-25 MPPB07
15A INSTRCLK
IRG00
IRG01
2WDOP P1-29
IREG1- 36C
3WDOP P1-31
IREGO P1-39
4WDOP P1-33- OR 5WD OP
IREG1ORFIX P1-41
IREG2
IREG3
IREG6
FPSUB+
MPYOP-
DIVOP-
FIXOP- 21C 45B
DIAGOP+
MPYOP+ 21D 31B 43C
DIVOP + 21A 31B 43C
EXEC2+
EXEC1+
FIXSGL+ P3-18
FLOATSGL +
FLOAT+ 11C 21C
FLOATDBL -
FLOATDBL +
FIXOP+ 45C
FIXDBL+ P3-22
FPDBL P1-30
FIXORFLT-
IREG7
ABZERO +
PBUSØ -
UNDERFLOW+
P1-7 EXPUNFL +
26D UNDERFLOW-
36B DIVTERM1 -
15A PWRST-
36B MDTERM-
16A EXECBGN -
16B PWR ON PP1 -
26A OVERFLOW-
13B PP2RS2+
15A PP2RS+
16B ARGCK1 +
CRG0EN +
CRG0EN - P3-45
MALUEN + P3-39
15A OVFLRS-
CPUOVFL-
OVERFLOWB - P1-40
INREGEN +
INREGEN - P3-44
EXPALUEN - P3-23
FPCONSTANT- P3-43
FP CONSTANT +
FP CONSTCK + 11C
REVISIONS
B REVISED & REDRAWN
C CLARIFIED DWG.
D LIFT U185-4 AND CONNECT TO U161-6 PER PCO 22-5892. DATE CODE TO 2033.
D-12740-60002-55
12740 FLOATING POINT CONTROL
SEE KARDEX
HEWLETT PACKARD
12740-60002
D-12740-60002-55
SHEET 1 OF 1
5

SIGNAL NAMES AND DESCRIPTIONS	APPENDIX A

A.0 CONTROL BOARD 12740-60002

SIGNAL	DESCRIPTION
2WDOP P1-29	32 bit single precision operation
3WDOP P1-31	48 bit extended precision operation
4WDOP P1-33	4 or 5 word operation
ABRGS1 P1-44	A and B mantissa register mode control S1
ABZERO+	A or B is equal to zero or is not normalized
ABZERO-	A or B is equal to zero or is not normalized
AEXPICK P1-12	load operand into AEXP from input register
AGTB- P1-17	A is greater than B
AGTBZERO-	A is greater than B or A equals zero
ALU55 P1-47	bit 55 of the ALU outputs
ALUDLY2-	Delay state 2 in ALU cycle
ALUDLY5-	Delay state 5 of ALU cycle
ANORMLZD+	A mantissa is normalized (AMAN(55) not = to AMAN(54))
ARG53 P1-24	A mantissa register bit 53
ARG54 P1-22	A mantissa register bit 54
ARG55B+ P1-20	A mantissa register bit 55, buffered
ARGCK P1-23	A mantissa register clock
ARGCKI+	Load operand into A mantissa register from input register
ARGCKI-	Load operand into A mantissa register from input register
ARGS0 P1-42	A mantissa register mode control S0
BEXPCK P1-16	B exponent register clock
BRG55 P1-21	B mantissa register bit 55

BRGCK	B mantissa register clock
P3-14	
BRGCKI+	Load operand into B-regester from input register
BRGCKI-	Load operand into B-register from input register
BRGZERO+	B mantissa is equal to zero or is unnormalized
BYTE2OEN-	enable output register bits 31-24
BYTE3OEN-	enable output register bits 15-8
CLEARA-	Clear A mantissa register bits 55-0
P3-8	
CLEARA32-	Clear A mantissa register bits 31-0
P3-7	
CLEARA48-	Clear A mantissa register bits 15-0
P3-4	
CLEARB-	Clear B mantissa register bits 55-0
P3-11	
CLEARB32-	Clear B mantissa register bits 31-0
P3-12	
CLEARB48-	Clear B mantissa register bits 15-0
p3-10	
CLEARBCTL-	Clear B mantissa register control state
CLEARC-	Clear C register bits 55-0
P1-35	
CLEARC32-	Clear C register bits 31-0
P3-15	
CLEARC48-	Clear C register bits 15-0
P3-13	
CLEARROUND-	Clear round circuits
P3-19	
CLOCK1,2,3,4	Buffered internal 40MHz clock
P3-50	
CLRABGN-	Clear excess A mantissa bits at execution begin
CLRADOUBLE-	Clear A mantissa register bits 23-0
P3-6	
CLRASINGLE-	Clear A mantissa register bits 39-0
P3-9	
CPUOVFL-	CPU overflow - set the CPU overflow flipflop
CRGCK	C register clock
P3-40	
CRGOEN+	C register output enable
CRGOEN-	C register output enable
P3-45	
CRGS0	C register mode control S0
P1-46	
CRGS1	C register mode control S1
P1-48	
DIAGOP+	Diagnostic operation - used in pretest
DIVADD-	Add divisor to partial remainder in ALU cycle
DIVCO+	Current quotient bit inverted
DIVEND+	End of division, go to normalization preparation
DIVOP+	Division is being performed
DIVOP-	Division is being performed

DIVSHFTLT-	Shift partial remainder and quotient left
DIVSUB-	Subtract divisor from partial remainder in ALU cycle
DIVTERM1-	Division termination sequence control state 1
EQUALA+	Equalize A - A is less than B
P1-14	
EQUALCTL1+	Equalization control state 1
EXEC1+	Execution group 1 = Add, Subtract, Fix
EXEC2+	Execution group 2 = Multiply, Divide
EXECBGN+	Execution begin, operands are loaded
EXECBGN-	Execution begin, operands are loaded
EXP10OVFL-	Exponent is beyond 10 bit range
P1-43	
EXPALUEN-	Exponent ALU enable; exponent ALU drives P-bus
P3-23	
ExPALUS0	Exponent ALU mode control S0
P1-10	
EXPALUS1	Exponent ALU mode control S1
P1-13	
EXPALUS2	Exponent ALU mode control S2
P1-15	
EXPCNTUP+	Exponent registers count up/down mode control
P1-6	
EXPEQUAL-	A exponent equals B exponent
P3-41	
EXPINCK	Load exponent from MPP bus to input register
P3-33	
EXPLD-	Exponent register parallel load mode control
P1-4	
EXPOEN-	Enable exponent output register bits 7-0
P3-34	
EXPOVFL+	Exponent has overflowed
P1-9	
EXPOVUNFL-	Exponent is not in 8 bit range
P1-5	
EXPSIGN	Sign of output of exponent ALU
P1-8	
EXPUINEN-	Upper exponent (bits 11-8) input register enable
P1-18	
EXPUNFL+	Exponent has underflowed
P1-7	
FIXDB0+	If high, P-bus bits 23-0 equal zero
FIXDB0-	If low, P-bus bits 23-0 equal zero
P3-16	
FIXDBL+	Fix to double integer operation is active
P3-22	
FIXOP+	If high, fix to integer is in operation
FIXOP-	If low, fix to integer is in operation
FIXORFLT-	Fix or float is in operation
FIXSG0-	P bus bits 39-0 equal zero
P3-17	
FIXSGL+	Fix to single integer is in operation

(HP 1000 M/E/F-SERIES ERD)

P3-18	
FLOAT+	If high, float is in operation
FLOAT-	If low, float is in operation
FLOATDBL+	Float from double integer is in operation
FLOATDBL-	Float from double integer is in operation
FLOATSGL+	Float from single integer is in operation
FPCONSTANTEN+	Fix/float constant enable on to P-bus
FPCONSTANTEN-	Fix/float constant enable on to P-bus
FPCONSTCK	Clock the load of the fix/float constant
FPDBL+	Fix or float double integer is in operation
P1-30	
FPPRS+	Reset FPP
FPSUB+	Subtract is in operation
GUARD1	First guard bit of round operation
P1-19	
INITIALIZE	Connector J1 external initialization signal
J1-31	
INITIALIZE2	Connector J1 external initialization signal
J1-33	
INITIALIZE3	Connector J1 external initialization signal
J1-39	
INREGEN+	Input register output enable
INREGEN-	Input register output enable
P3-44	
INSTRCLK	Load MPP data into instruction register clock
IREG0	Instruction register bit 0 buffered
P1-39	
IREG1	Instruction register bit 1
IREG1ORFIX	Instruction register bit 1 set or fix operation
P1-41	
IREG2	Instruction register bit 1
IREG3	Instruction register bit 2
IREG6	Instruction register bit 6
IREG7	Instruction register bit 7
IRG00	Instruction register bit 0
IRG01	Instruction register bit 1
IRST	Instruction register store - MPP control signal
J2-31	
LASTDIVC0	Last quotient bit developed
MALUEN-	Mantissa ALU output enable
P3-39	
MDALUCTL2-	Multiply/divide ALU cycle control state 2
MDALUCTL4-	Multiply/divide ALU cycle control state 4
MDEXPCTL4-	Multiply/divide exponent sequence control state 4
MDEXPCTL5-	Multiply/divide exponent sequence control state 5
MDSHFTCTL-	Multiply/divide shift cycle control state 1
MDTERN4-	Multiply/Divide termination sequence control state 4
MNTS0	Mantissa ALU mode control S0
P3-48	
MOVFL+	Mantissa has overflowed
P1-11	

MOVFL-	Mantissa has overflowed
MOVFLCTL1-	Mantissa overflow sequence control state 1
MOVFLCTL3-	Mantissa overflow sequence control state 3
MPBEN J2-41	MPP enable control signal, send data to CPU
MPBST J2-27	MPP store control signal, store MPP data
MPPBnn J1	MPP data bus bit nn
MPPCNDX J2-9	MPP conditional signal, FPP ready/busy signal
MPYADD-	Add multiplicand to partial product in ALU cycle
MPYC0 P3-3	Current least significant bit of multiplier
MPYC1 P3-5	Current next to least significant bit of multiplier
MPYCKC1-	First clock to shift C in multiply's ALU cycle
MPYCKC2-	Second clock to shift C in multiply's ALU cycle
MPCKCINIT-	Multiply's clock initial operation decision
MPYEND+	End of multipliation, go to normalization preparation
MPYOP+	Multiplication is in operation
MPYSHFTRT-	Shift multiplier and partial product right
MPYSLT-	Shift ALU output twice to the right through multiplexer
MPYSUB-	Subtract multiplicand from partial product in ALU cycle
NORMCTLI-	Normalization sequence control state 1
NORMPREP1+	Normalization preparation sequence control state 1
NORMPREP1-	Normalization preparation sequence control state 1
NORMPREP2+	Normalization preparation sequence control state 2
NORMPREP2-	Normalization preparation sequence control state 2
OVERFLOW+	Overflow control state
OVERFLOW- P1-40	Overflow control state
OVFLRS-	Reset FPP's CPU overflow flipflop
PBUS0- P3-37	P-bus bits 55-0 are zero
PBUS0CLK P1-28	Clock P-bus equal to zero register
PBUS54 P1-26	P-bus bit 54
PBUS55 P1-25	P-bus bit 55
PBUS56- P1-37	P-bus bit 56, inverted
PLR0- J2-29	CPU latch register bit 0, inverted
PP1SP J2-11	MPP special control signal 1; qualifies MPBEN, MPBST
PP2RS	PP2SP received, prepare for FPP operation
PP2SP	MPP special control signal 2; initates operation

J2-7	
PP5	CPU clock phase 5 timing signal
J2-3	
PWRON+	Power supply voltage is up
P2-25	
PWRONPP1-	Power on or PP1SP reset
PWRST-	Power on, PP1SP reset or PP2SP reset
P3-38	
PWRST2-	Power on, PP1SP reset or PP2SP reset
ROUND-	Result should be rounded
P3-42	
ROUNDCLOCK	Round register clock
P3-20	
ROUNDEN+	If high, enable rounding ciruits
P1-38	
ROUNDEN-	If low, enable rounding circuits
STOV-	MPP control signal to set CPU overflow flip-flop
J2-1	
SUBEQB+	Subtract and equalize B is in operation
P1-27	
SWAMP+	Swamp condition-exponents are too far apart to be equalized
P3-37	
SWAMP1-	Swamp sequence control state 1
SWAMP4+	Swamp sequence control state 4
SWAMP4-	Swamp sequence control state 4
SWAMPSBGTA+	Swamp condition and subtract and B is greater than A
SWPDLY1+	Check exponents for swamp condition control state 1
SWPDLY1-	Check exponents for swamp condition control state 1
SWPDLY4+	Check exponents for swamp condition control state 4
TERMTCTL1-	Termination sequence control state 1
TERMCTL4-	Termination sequence control state 4
TERMLOAD-	Termination sequence control load all registers clock
P1-36	
TIEUPI	Tie up to +5 volts
UNDERFLOW+	If high, underflow control state
UNDERFLOW-	If low, underflow control state
W1INCK	Clock input register bits 55-40
P3-25	
W1OEN	Enable output register bits 55-40
P3-24	
W2INCK	Clock input register bits 39-24
P3-27	
W2OEN-	Enable output register bits 39-32
P3-28	
W3INCK	Clock input register bits 23-8
P3-29	
W4OEN-	Enable output register bits 23-16
P3-30	
W4INCK	Clock input register bits 7-0
P3-31	

W4OEN- P3-32	Enable output register bits 7-0
W5OEN- P1-32	Enable exponent output register bits 15-8

A.1 ALU BOARD 12740-60001

SIGNAL	DESCRIPTION
2WDOP P1-29	32 bit single precision operation
3WDOP P1-31	48 bit extended precision operation
4WDOP P1-33	4 word or 5 word operation
ABRGS1 P1-44	A and B mantissa registers mode control S1
AEXPCK P1-12	A exponent register clock
AGTB- P1-17	If low, A is greater than B
ALUnn	ALU output bit nn
ALU55 P1-47	ALU output bit 55
ARGnn	A mantissa register bit nn
ARGCK P1-23	A mantissa register clock
ARGS0 P1-42	A mantissa register mode control S0
BEXPCK P1-16	B exponent register clock
BRGnn	B mantissa register bit nn
BRGCK P3-14	B mantissa register clock
BRGS0	B mantissa register mode control S0
BYTE2OEN- P3-35	Enable output register bits 31-24
BYTE3OEN- P3-36	Enable output register bits 15-8
CKOTR	Output register clock
CLEARA- P3-8	Clear A mantissa bits 55-0
CLEARA32- P3-7	Clear A mantissa bits 31-0
CLEARA48- P3-4	Clear A mantissa bits 15-0
CLEARB- P3-11	Clear B mantissa bits 55-0

CLEAB32 P3-12	Clear B mantissa bits 31-0
CLEARB48- P3-10	Clear B mantissa bits 15-0
CLEARC- P1-35	Clear C register bits 55-0
CLEARC32- P3-15	Clear C register bits 31-0
CLEARC48- P3-13	Clear C register bits 15-0
CLEARROUND- P3-19	Clear round circuits
CLRADOUBLE- P3-6	Clear A mantissa register bits 7-0
CLRASINGLE- P3-9	Clear A mantissa register bits 23-0
CRGnn	C register bit nn
CRGCK P3-40	C register clock
CRGOEN- P3-45	C register output enable
CRGS0 P1-46	C register mode control S0
CRGS1 P1-48	C register mode control S1
DIVC0- P1-34	Current quotient bit inverted
EQUALA+ P1-14	Equalize A
EXPI0OVFL+ P1-43	Exponent is beyond 10 bit range
EXPALUEN- P3-23	Enable exponent ALU to drive P bus
EXPALUS0 P1-10	Exponent ALU mode control S0
EXPALUS1 P1-13	Exponent ALU mode control S1
EXPALUS2 P1-15	Exponent ALU mode control S2
EXPCNTUP+ P1-6	Exponent register count up/down mode control
EXPEQUAL- P3-41	A exponent equals B exponent
EXPINCK P-33	Clock exponent input register
EXPLD- P1-4	Exponent register load mode control
EXPOEN-	Enable output of exponent register bits 7-0
EXPOVFL+ P1-9	Exponent has overflowed
EXPOVUNFL-	Exponent is beyond 8 bit range

P1-5	
EXPSIGN	Sign of exponent ALU results
P1-8	
EXPUINEN-	Enable upper exponent input register bits 11-7
P1-18	
EXPUNFL+	Exponent has underflowed
P1-7	
EXTCK	External clock to FPP
J1-3	
FIXDB0-	P bus bits 23-0 are zero
P3-16	
FIXDBL+	Fix to double integer is in operation
P3-22	
FIXSG0-	P bus bits 39-0 are zero
P3-17	
FIXSGL+	Fix to single integer is in operation
P3-18	
FPCONSTANTEN-	Enable fix/float constant on exponent P-bus
P3-43	
FPDBL+	Fix/float double integer is in operation
P1-30	
GUARD1	First guard bit of round register
P1-19	
INREGEN-	Enable input register to drive P-bus
P3-44	
IREG0	Instruction register bit 0
P1-39	
IREG1	Instruction register bit 1
P1-41	
LACnn	Look ahead carry bus nn
LACCIN	Look ahead carry circuits carry input
MALUEN+	Enable mantissa ALU to drive P-bus
MALUEN-	Enable mantissa ALU to drive P-bus
P3-39	
MNTS0	Mantissa ALU mode control S0
P3-48	
MNTS1	Mantissa ALU mode control S1
MPPBIOnn	MPP I/O bus bit nn
J2	
MPYC0	Current least significant bit of multiplier
P3-3	
MPYC1	Current next to least significant bit of multiplier
P3-5	
MPYSLT-	Shift ALU output twice to the right through multiplexer
P3-21	
MPYSLTB-	Shift ALU output twice to the right buffered
OVERFLOW-	Overflow control state
P1-40	
PBUS0	P bus bits 55-0 are equal to zero
P3-37	

PBUSOCLK P1-28	P-bus zero detect register clock
PBUSnn	P-bus bit nn
PBUS56- P1-37	P-bus bit 56 inverted
PWRONPP1- P1-45	Power on or PP1SP reset to FPP
PWRST- P3-38	PP2SP reset power on or PP1SP reset to FPP
ROUND- P3-42	Result should be rounded
ROUNDCLOCK P3-20	Round register clock
ROUNDEN+ P1-38	Enable round circuits
SUBEQB+ P1-27	Subtract and equalize B
SWAMP+ P3-37	Swamp condition-exponents are too far apart to be equalized
TERMLOAD- P1-36	Termination sequence load all registers clock
W1INCK P3-25	Input register bits 55-40 clock
W1OEN- P3-24	Enable output register bits 55-40
W2INCK P3-27	Clock output register bits 39-24
W2OEN- P3-28	Enable output register bits 39-32
W3INCK P3-29	Clock output register bits 23-8
W3OEN- P3-30	Enable output register bits 23-16
W4INCK P3-31	Clock input register bits 7-0
W4OEN- P3-32	Enable output register bits 7-0
W5OEN- P1-32	Enable exponent output register bits 15-8

```
+-----------------------------------------------------------+------------------+
|                                                           |                  |
|   FPP/MPP SIGNAL LINES                                    |   APPENDIX  B    |
|                                                           |                  |
+-----------------------------------------------------------+------------------+
```

FPP CONNECTOR J2
ALL EVEN PINS OF J2 ARE GROUNDED.

```
1  NSTOV
3  PP5   - P5 OF THE CPU CYCLE
5  MPPIO 11
7  PP2SP - INITIATE FPP OPERATION
9  MMCND - FPP BUSY
11 PP1SP - FPP RESET AND QUALIFIER
13 MPPIO 12
15 MPPIO 13
17 MPPIO 14
19 MPPIO 15
21 MPPIO  8
23 MPPIO  9
25 MPPIO  7
27 MPPIO - STORE DATA
29 PLR0  - L-REGISTER 0, FPP ADDRESS
31 PIRST - INSTRUCTION REGISTER STORE
33 MPPIO  6
35 MPPIO  5
37 MPPIO  4
39 MPPIO  3
41 MPBEN - READ DATA
43 MPPIO  2
45 MPPIO  1
47 MPPIO  0
49 MPPIO 10
```

The port accomodates two external processors which are individually addressed by latch register zero high or low.

```
+-----------------------------------------------------+------------------+
|                                                     |                  |
|  CURRENT REQUIRED                                   |  APPENDIX  C     |
|                                                     |                  |
+-----------------------------------------------------+------------------+
```

The current required by the FPP control and arithmetic boards is:

		TYP	RMS	MAX
ARITHMETIC	@5V	7.5A	9.75A	11.0A
CONTROL	@5V	3.6A	5.16A	6.0A