EK-FP730-TD-001

VAX-11/730 FP730 Floating-Point Accelerator Technical Description

Prepared by Educational Services
of
Digital Equipment Corporation

First Edition, May 1982

The material in this manual is for informational purposes and is subject to change without notice.

Digital Equipment Corporation assumes no responsibility for any errors which may appear in this manual.

Printed in U.S.A.

This document was set on DIGITAL's DECset-8000 computerized typesetting system.

5/82-14

CONTENTS

APPENDIX A PROGRAMMED ARRAY LOGIC DEVICES (PALs)

APPENDIX B GLOSSARY

FIGURES

TABLES

Table No. **Title Page**

CHAPTER 1
INTRODUCTION

1.1 GENERAL

The FPA-11/730 floating-point accelerator (FPA) is a hardware option that performs all floating-point arithmetic operations and converts data between integer and floating-point formats. Floating-point representation permits a greater range of number values than is possible with a 32-bit integer. The FPA option accelerates execution of most floating-point instructions and a few integer instructions. Without the FPA the floating-point instructions are executed by central processor unit (CPU) microcode, with little hardware help. The FPA operates on single, double, grand, and huge data formats or types.

Functionally, the FPA is an integral part of CPU. It operates using the same address modes and the same memory management facilities as the CPU. Floating-point processor instructions can reference the CPU's general registers or any location in memory.

1.2 RELATED DOCUMENTATION

Table 1-1 lists all related documentation.

Table 1-1 Related Hardware Manuals

Title	Comments
VAX-11/730 Central Processor Technical Description	In microfiche library
VAX-11 Architecture Handbook	Available in hard copy*

*This document can be ordered from:

Digital Equipment Corporation
444 Whitney Street
Northboro, MA 01532
Attention: Communication Services (NR2/M15)
Customer Services Section

For information concerning microfiche libraries, contact:

Digital Equipment Corporation
Micropublishing Group, PK3-2/T12
129 Parker Street
Maynard, MA 01754

1.3 PHYSICAL DESCRIPTION

The FPA-11/730 consists of a standard hex module, containing mostly Schottky TTL logic. There are no calibration adjustments, switches or controls.

1.4 FUNCTIONAL DESCRIPTION

The FPA-11/730 FPA is a hardware option available on the VAX-11/730 computer system. It can perform floating-point addition, subtraction, multiplication, and division instructions.

The FPA, functioning in conjunction with the CPU, speeds the execution of floating-point arithmetic instructions. FPA operations overlap CPU operations, allowing the CPU to proceed with other tasks relating to the floating-point instruction, such as destination address calculation, while the FPA completes the instruction. The CPU cannot overlap another instruction; it must wait for the FPA to complete the floating-point instruction. This overlap helps to speed program execution.

The FPA also speeds the execution of some integer arithmetic instructions. Operation of the FPA is transparent to macro level software and main machine microcode.

The FPA can operate on a wide range of numbers. A floating-point number between 1.5 $\times$ 10^{-39} and 3.4 $\times$ 10^{38} can be represented. A single-precision number is accurate to 7-decimal digits, and a double-precision number to 16-decimal digits. The range of a grand operand is 8.9 $\times$ 10^{+307} to 1.11 $\times$ 10^{308}. The range of a huge operand is 5.94 $\times$ 10^{4931} to 8.40 $\times$ 10^{-4933}. The FPA can operate on 32-bit signed integers from $-2,147,483,648$ to 2,147,483,647, inclusive.

As a functional extension of the CPU, the FPA does not access memory data. The CPU must calculate a memory address, access the address, and then transmit the data to the FPA. The CPU is also responsible for fetching and storing the FPA results. The FPA performs only the required floating-point or integer operation on the properly formatted operands transmitted to it.

Basically, the FPA (Figure 1-1) consists of data path logic that processes operands, and a control store that generates data processing control signals. The data path logic consists of 20 4-bit 2901 bit slices (microprocessors).

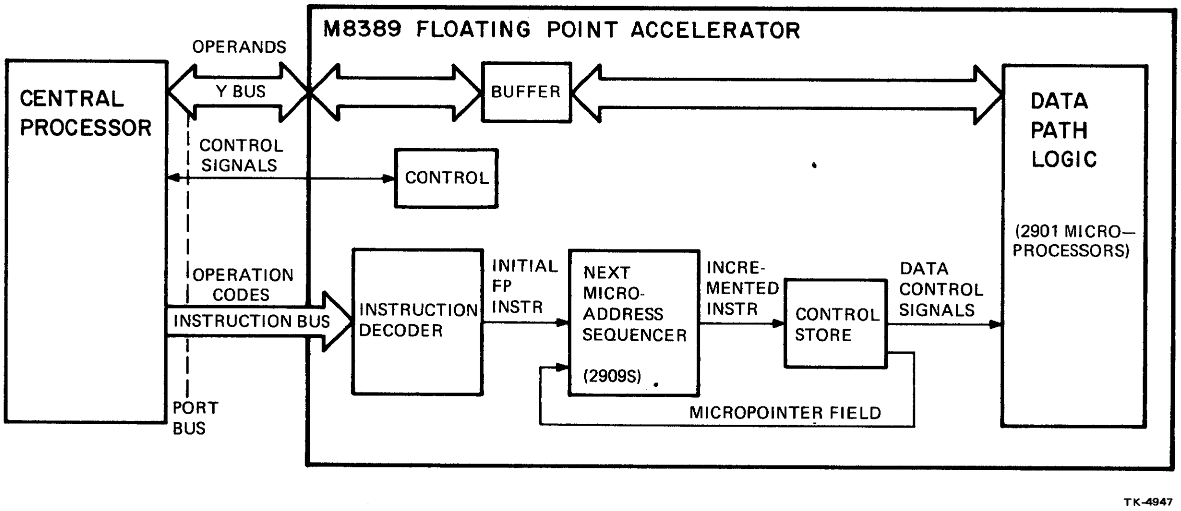


Figure 1-1 FPA-11/730

Initially, the CPU sends the FPA an operation code that is decoded into a starting microaddress. An FPA sequencer converts the instruction into an address for a control store PROM where data path logic control signals are generated. This sets up the data path logic to receive the first data input via the Y-Bus.

The CPU then sends the FPA packed, normalized, floating-point data, including a sign bit, in the form of 32-bit operands. These are buffered, and applied to the data path logic. The data path logic breaks the number (operand) into parts (unpacks it) and performs operations required to carry out the instruction on each part. Once the arithmetic result is achieved, the data path logic normalizes and packs the results in accordance with control signals in the control store. The result is then buffered and returned to the CPU in 32-bit segments via the Y-Bus.

As the FPA performs calculations, a micropointer field in the FPA control store points to the next microaddress to be executed. This address is then latched in the 2909 microsequencer, which alters the latched base microaddress by ORing selected status signals into it. The result is the next microaddress for control store.

1.5 DIAGNOSTIC FEATURES

FPA diagnostics include a force/read function whereby the CPU can force an address into the FPA control store or read the next address the microsequencer will apply to the control store. Diagnostics check operation of the instruction decoding circuit, microsequencer, control store, and data path logic. Two parity bits are used to perform error checks on the control store. If a parity error occurs, the FPA traps to a parity error routine.

1.6 FLOATING-POINT NUMBERS AND ARITHMETIC

1.6.1 Integers

All data within a computer system can be represented in integer form. The numbers that can be represented in a 32-bit machine range in magnitude from 00000000_{16} to $FFFFFFFF_{16}$ (or from 0_{10} to 4,294,967,295). However, integer form imposes some limitations. Only whole numbers can be represented, i.e., no fraction or decimal parts. This imposes an accuracy limitation. Also, numbers greater than 4,294,967,295 cannot be represented; this imposes a range limitation.

These limitations are imposed by the stationary position of the radix point (e.g., the decimal point in base 10 notation, or the binary point in base 2 notation). An integer's radix point is usually omitted in integer representation because it always marks the integer's least significant place. That is, there are never any digits to the right of a radix point. For this reason, an integer is sometimes called a fixed-point number.

Integer notation, however, can be modified to overcome the range and accuracy limitations imposed by the fixed radix point. This is done through the use of floating-point notation.

1.6.2 Floating-Point Numbers

Floating-point numbers, unlike integers, have no position restrictions imposed on their radix points. A popular type of floating-point representation is called scientific notation. With scientific notation, a floating-point number is represented by some basic value multiplied by the radix raised to some power.

Example 1

$$1{,}000{,}000 = \overset{\text{basic value}}{1.} \times \underset{\text{radix}}{10}^{\overset{\text{exponent}}{6}}$$

There are many ways to represent the same number in scientific notation, as shown in Example 2.

Example 2

Right-Shifts	**Left-Shifts**
$512 = 512. \times 10^{0}$	$512 = 512 \times 10^{0}$
$= 51.2 \times 10^{1}$	$= 5120 \times 10^{-1}$
$= 5.12 \times 10^{2}$	$= 51200 \times 10^{-2}$
$= .512 \times 10^{3}$	$= 512000 \times 10^{-3}$

The convention chosen for representing floating-point numbers with scientific notation in the FPA requires that the radix point always be positioned to the left of the most significant digit in the basic value (e.g., $.512 \times 10^{3}$ in the above example). This modified basic value is called a mantissa fraction.

Note that for each right-shift of the basic value, the exponent is incremented and for each left-shift the exponent is decremented. The value of the number remains constant if the exponent is adjusted for each shift of the basic value.

Additional examples of scientific notation are indicated in Example 3.

Example 3

Decimal Notation	**Decimal Scientific Number**	**Binary Notation**	**Hex Notation**	**Hex Scientific Number**
64	$.64 \times 10^{2}$	1000000.	40_{16}	$.4 \times 16^{-2}$
33	$.33 \times 10^{2}$	100001.	21_{16}	$.21 \times 16^{-2}$
1/2(.5)	$.5 \times 10^{0}$	0.1	$.8_{16}$	$.8 \times 16^{0}$
3/32(.09375)	$.9375 \times 10^{-1}$	0.00011	$.18_{16}$	$.18 \times 16^{0}$

1.6.3 Normalization

There are many ways to represent a particular floating-point number using scientific notation. The convention chosen by VAX and the FPA requires the radix point to be to the left of the most significant bit in the basic value, as in Example 4.

Example 4: Floating-Point Form

$29_{10} = 11101_2 =$		$1\ 1101.$	$\times\ 2^0 =$	$1\ 1101. \times 2^0$
		1110.1	$\times\ 2^1 =$	$11\ 1010. \times 2^{-1}$
		111.01	$\times\ 2^2 =$	$111\ 0100. \times 2^{-2}$
.11101		11.101	$\times\ 2^3 =$	$1110\ 1000. \times 2^{-3}$
Fraction		1.1101	$\times\ 2^4 =$	$1\ 1101\ 0000. \times 2^{-4}$
	Chosen $\longrightarrow$	$.11101$	$\times\ 2^5 =$	$11\ 1010\ 0000. \times 2^{-5}$
5	Form	$.011101$	$\times\ 2^6 =$	$111\ 0100\ 0000. \times 2^{-6}$
Exponent		$.0011\ 101$	$\times\ 2^7 =$	$1110\ 1000\ 0000. \times 2^{-7}$

The process of ensuring that the first significant bit is directly to the right of the binary point is called normalization. If the number is one or larger, it involves right-shifting the basic value and incrementing the exponent until the most significant bit (MSB) (a one) is directly to the right of the binary point. If the number is a fraction with leading zeros, the basic value is left-shifted and the exponent is decremented. Examples 5 and 6 show conversion of numbers to normalized form.

Example 5: Convert 75_{10} to a normalized binary number.

1. Integer conversion
   $75_{10} = 100\ 1011_2$

2. Floating-point form
   $100\ 1011_2 = 100\ 1011_2 \times 2^0$

3. Normalized form
   Right-shift fraction 7 times
   Increment exponent by 7

   $100\ 1011_2 \times 2^0 = .100\ 1011 \times 2^7$

   Fraction = .100 1011
   Exponent = 7

Example 6: Convert 3/16 (.01875) to a normalized binary number.

1. Integer conversions
   $.01875_{10} = .0011_2$

2. Floating-point form
   $.0011_2 = .0011_2 \times 2^0$

3. Normalized form
   Left-shift fraction twice
   Decrement exponent by 2

   $.0011_2 \times 2^0 = .11 \times 2^{-2}$

   Fraction = .11
   Exponent = -2

1.6.4 Floating-Point Notation

Two FPA conventions are used to conserve memory space without losing accuracy, and to aid in hardware manipulation. The first convention is called the hidden bit. All numbers transferred between the CPU and FPA are normalized floating-point numbers. This means that the first significant bit (always a 1) is always directly to the right of the binary point. To conserve memory space and data lines, the first significant bit is not stored or transmitted to the FPA. For example, the fraction part of the normalized binary number .11000... $\times$ 2^{-2} is stored and transmitted to the FPA as 100.... The normalized fraction of 1/2 (.100.. $\times$ 2^0) is stored and transmitted as 000.... In both cases the first 1 (the hidden bit) is added by hardware in the FPA. When the FPA transfers a normalized answer back to the CPU, the hidden bit is not sent.

The second convention is exponent bias notation. The exponent portion of a floating-point number is stored using excess 80_{16}, 400_{16}, or 4000_{16} notation. This notation simplifies the hardware that manipulates the exponent during floating-point arithmetic operation. Excess 80_{16} exponent notation is obtained by adding 10000000_2 (200_8, 80_{16}, or 128_{10}) to 2s complement notation. This allows the exponent to be stored as a positive value.

1.6.5 Floating-Point Addition and Subtraction

To perform floating-point addition or subtraction, the exponents of the two floating-point numbers involved must be aligned or equal. If they are not aligned, the fraction with the smaller exponent is right-shifted until they are. Each shift to the right is accompanied by an increment of the associated exponent. When the exponents are equal, the fractions can then be added or subtracted. The exponent value indicates the number of places the binary point is to be moved to obtain the integer representation of the number.

In Example 7, the number 7_{10} is added to the number 40_{10} using floating-point representation. Note that the exponents are first aligned and then the fractions are added. The exponent value dictates the final location of the binary points.

Example 7: Floating-Point Addition

$0.1010\ 0000\ 0000\ 000 \times 2^6 = 28_{16} = 40_{10}$

$+0.1110\ 0000\ 0000\ 000 \times 2^3 = 7_{16} = 7_{10}$

1. To align exponents, shift the fraction with the smaller exponent three places to the right and increment the exponent by 3. Then add the two fractions.

   $0.1010\ 0000\ 0000\ 000 \times 2^6 = 28_{16} = 40_{10}$

   $+0.0001\ 1100\ 0000\ 000 \times 2^6 = 7_{16} = 7_{10}$

   $0.1011\ 1100\ 0000\ 000 \times 2^6 = 2F_{16} = 47_{10}$

2. To find the integer value of the answer, move the binary point six places to the right.

   010 1111.0000 0000 0

1.6.6 Floating-Point Multiplication and Division

In floating-point multiplication, the fractions are multiplied and the exponents are added. In floating-point division, the fractions are divided and the exponents are subtracted. There is no requirement to align the binary point in floating-point multiplication or division. Example 8 shows floating-point multiplication; Example 9 shows division.

Example 8: Multiply 7_{10} by 40_{10}.

1. $$\begin{array}{rl}
0.1110000 \times 2^3 & = 7 = 7_{10} \\
\underline{\times\ 0.1010000 \times 2^6} & = 28_{16} = 40_{10} \\
1110000 & \\
0000 & \\
\underline{11100} & \\
.1000110000 \times 2^9 & \text{(Result already in normalized form)}
\end{array}$$

2. Move the binary point nine places to the right.

$$100011000.00000 = 118_{16} = 280_{10}$$

Example 9: Divide 15_{10} by 5_{10}.

1. $$\frac{.1111000 \times 2^4}{.1010000 \times 2^3}$$

$$\begin{array}{r@{\,}l}
 & 1.10000 \\
1010000 \big) & 1111000.000000 \\
 & \underline{1010000} \\
 & 101000 \\
 & \underline{101000} \\
 & 0
\end{array}$$

2. Exponent: 4-3 = 1

3. Result: 1.100000×2^1

Normalized Result: $.1100000 \times 2^2$

Normalized Fraction   Normalized Exponent

Move binary point two places to the right.

$11.000000 = 3_{16} = 3_{10}$

CHAPTER 2
DATA FORMATS

2.1 GENERAL

The FPA requires its input data (operands) to be formatted. Formatting allows the FPA to process operands in a meaningful way and produce correct results. There are five different formats for operands inputted to the FPA: single (F), double (D), grand (G), huge (H) precision, plus integer. The FPA output is in F, D, G, H, or integer format.

2.2 FLOATING-POINT FORMATS

Of the four floating-point formats (Figures 2-1 through 2-4), single (F) is 32 bits long. Double (D) and grand (G) are 64 bits long and huge (H) is 128 bits long. The words contain fraction and exponent fields, plus a sign bit. Figures 2-1 through 2-4 illustrate how the format is rearranged in the FPA.

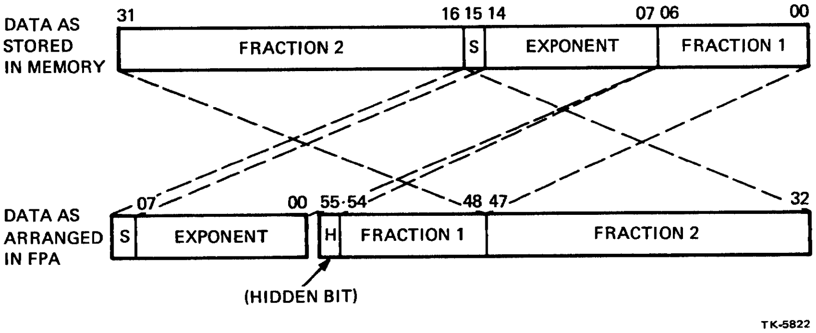


Figure 2-1 Single Precision Data Format

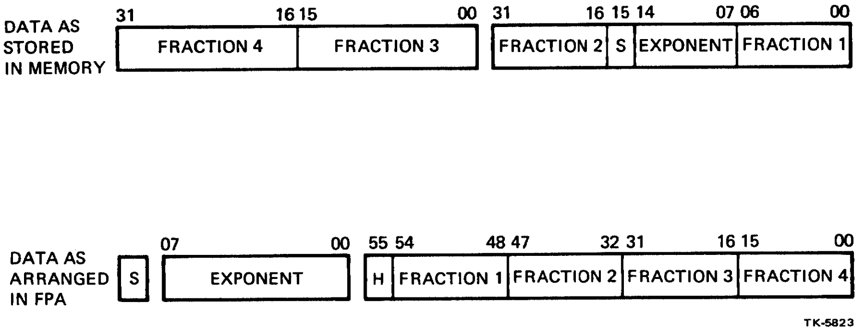


Figure 2-2 Double Precision Data Format

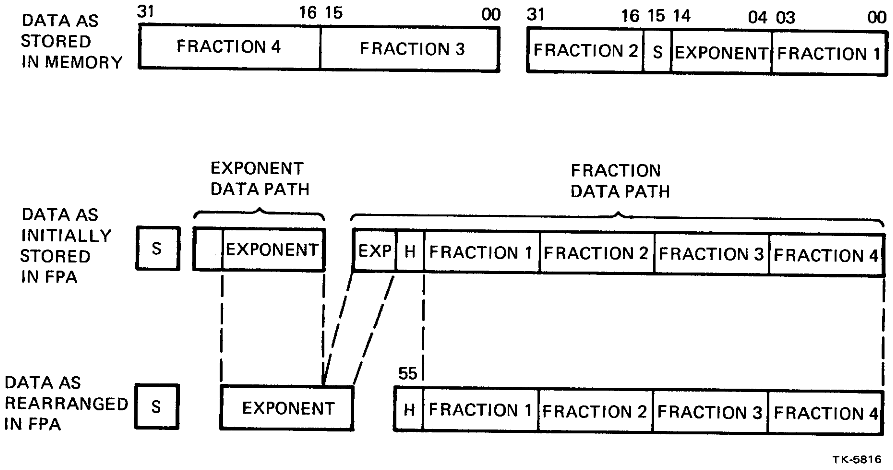


Figure 2-3 Grand Data Format

2.2.1 Fraction

The fraction is a normalized magnitude, binary representation. Table 2-1 explains sign and magnitude notation of the fraction. Only a change of sign bit is required to change the sign of a number in sign and magnitude notation. Note that a positive number is the same in both notations.

The fraction contains a binary number of the form:

0.1XXXXX....

The first bit of the fraction is always a one because the fraction is normalized at the end of every instruction. Normalization consists of aligning the MSB of the result with the MSB of the fraction and adjusting the exponent accordingly. For example:

$$[.1 \times 2^{**}1] \times [.1 \times 2^{**}3] = .01 \times 2^{**}4$$

$$\text{Normalize Result} = .1 \times 2^{**}3$$

The fraction contains a hidden bit. Since the MSB of every fraction is always a one, this bit is not stored in memory; this is the hidden bit. The FPA inserts this bit whenever it receives an operand.

Table 2-1 Fraction Sign and Magnitude Notation

	2s Complement Notation	Sign and Magnitude Notation
+2	000010	000010
−2	111110	100010

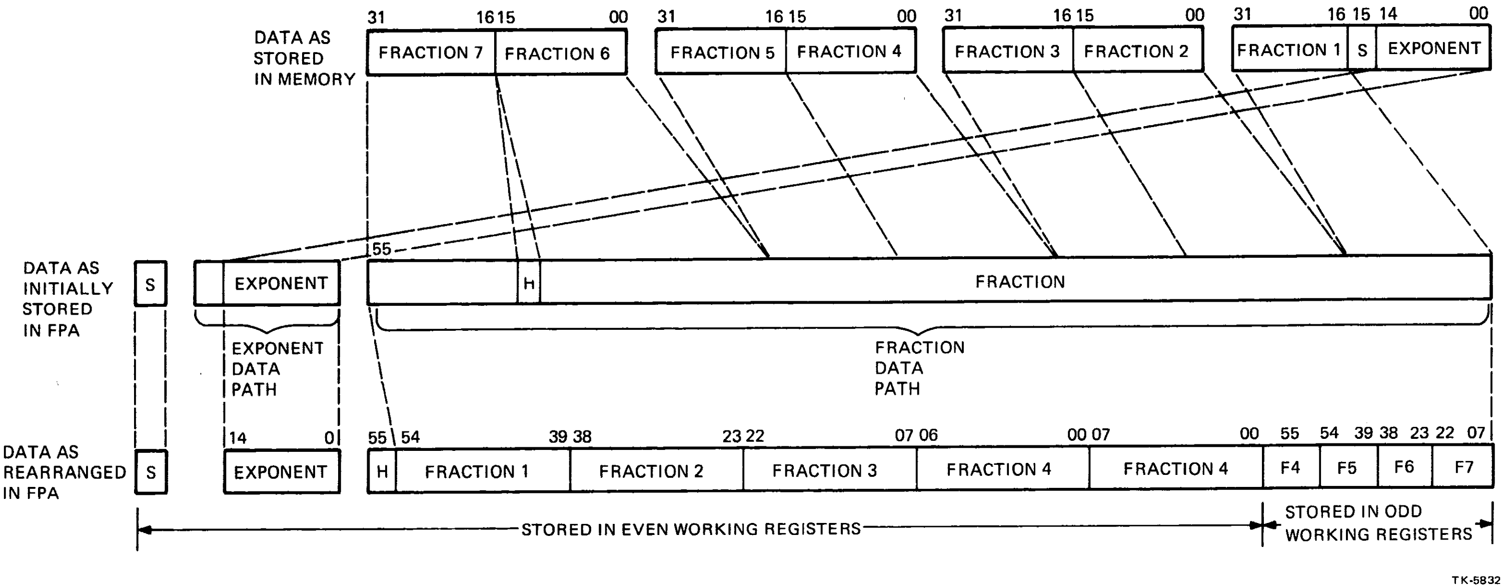


Figure 2-4 Huge Data Format

2.2.2 Exponent

As Figure 2-1 illustrates, an 8-bit exponent is used for single-(F) and double-(D) precision formats; an 11-bit exponent is used for grand (G) format (Figure 2-3); and a 15-bit exponent is used for huge (H) formats (Figure 2-4).

The exponent contains a power of 2 and can be expressed in excess 80, 400, 4000 (according to data type) notation (bias). (Refer to Table 2-2.) The bias is added to a power of 2 to yield the exponent.

Table 2-2 Excess Notation Usage

Bias (HEX)	
(Hexadecimal)	**Data Type**
80	F, D
400	G
4000	H

Excess 80/400/4000 notation is used to store and handle the exponent portion of floating-point numbers. The notations are used similarly; excess 80 notation is the 2s complement of the exponent plus 128_{10} or 80_{16}.

It is convenient to handle the exponent portion of the floating-point number in 2s complement notation. This allows a wide range of both positive and negative exponents to be represented. However, in 2s complement notation, an overflow must occur to go from the least negative number to zero. To avoid this, the bias of 128_{10} is added to the 2s complement number.

When multiply and divide operations are performed using floating-point numbers with excess 80 exponent notation (or 400 or 4000, as required), the resulting exponent must be adjusted by the bias to return the result to excess 80_{16} notation. When a multiplication is performed, exponents are added, and 80_{16} must be subtracted from the result to return it to excess 80 notation. The following example explains why 80_{16} must be subtracted from the exponent calculation during multiplication.

Exponent A + 80_{16}

Excess 80_{16} notation

Exponent B + 80_{16}

Exponent A + Exponent B + 100_{16}

Both exponent A and exponent B are biased by 80_{16} yielding a bias of 100_{16}. However, only a bias of 80_{16} is desired in excess 80_{16} notation.

Multiplication Example

$2 \times 3 = 6$

$$\begin{array}{lcl} \text{Fraction} & & \text{Exponent} \\ 2 = 0.100 \quad \times & & 82_{16} \\ 3 = 0.110 \quad \times & & 82_{16} \end{array}$$

$$\begin{array}{lcl} \text{Fraction Calculation} & & \text{Exponent Calculation} \\ 2 = 0.100 & & 82_{16} \\ \underline{3 = 0.110} & & \underline{+82_{16}} \\ 1000 & & 104_{16} \\ \underline{100} & & \underline{-80_{16}} \\ 6 = 0.011000 & \times & 84_{16} \end{array}$$

Normalize the fraction by left-shifting one place and decreasing the exponent by 1.

$$\begin{array}{ccc} \text{Fraction} & & \text{Exponent} \\ \downarrow & & \swarrow \\ 0.11000 & \times & 83 = 6 \end{array}$$

When a division is performed, exponents are subtracted and 80_{16} must be added (for excess 80 notation) to the result to return it to excess 80 notation. To understand why 80 must be added to the exponent calculation during division, consider the following:

$$\begin{array}{rl} & \text{Exponent A} + 80 \\ - & \underline{\text{Exponent B} + 80} \\ \multicolumn{2}{l}{\text{Exponent A} - \text{Exponent B} + 80 - 80 = \text{Exponent A} - \text{Exponent B} + 0} \end{array}$$

However, since the result is to be in excess 80 notation, 80_{16} must be added to the exponent, yielding Exponent A – Exponent B + 80.

Division Example

16/4 = 4

Fraction		Exponent
16 = .10000	×	85
4 = .10000	×	83

Fraction Calculation	Exponent Calculation
$\begin{array}{r} 1.000 \\ 0\ 10000.\overline{)\,0\ 10000.000} \end{array}$	$\begin{array}{r} 85 \\ \underline{-83} \\ 2 \\ \underline{+80} \\ 82 \end{array}$

Normalize the fraction by right-shifting one place and incrementing the exponent.

$$\underset{\substack{\uparrow \\ \text{Fraction}}}{.10000} \times \underset{\substack{\nearrow \\ \text{Exponent}}}{83} = 4$$

Figure 2-5 shows the relationship between an 8-bit floating-point exponent in 2s complement notation, and exponents in excess 80 notation.

Note that an exponent in excess 80 notation is obtained by simply adding 80 to the exponent in 2s complement notation. Thus, 8-bit exponents in excess 80 notation range from 0 to FF (−80 to +7F). A number with an exponent of −80 is treated by the FPA as 0.

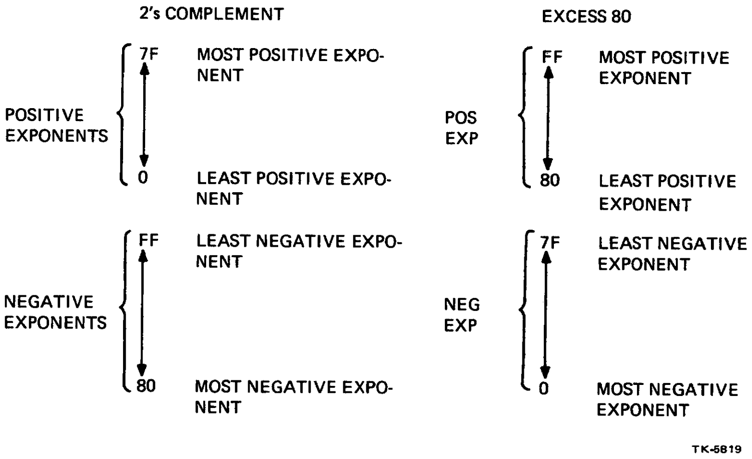


Figure 2-5 Excess 80 Notation for Single and Double Precision Format Exponents

2.3 INTEGER FORMAT

Integers processed by the FPA are 2s complement binary numbers (Figure 2-6). The MSB of the word received from memory is the sign bit.

Words and bytes in integer format can be loaded into the FPA for conversion to F, D, G, or H format. Also, the FPA can perform store operations whereby F, D, G, or H formatted data is loaded into memory as words or bytes.

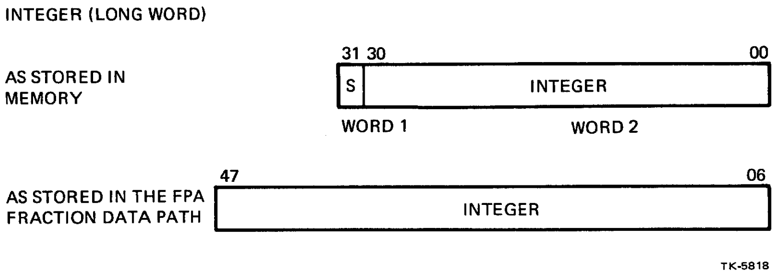


Figure 2-6 Integer Format

2.4 FLOATING-POINT EXCEPTIONS

The FPA monitors all operands and results for exceptional conditions. When the FPA senses one or more of these conditions, it informs the CPU via various bits and combinations of bits. Either one or both units begin special operations designed to minimize the effect of the condition. In some cases it stops the current FPA operation and returns the FPA to the instruction decoding (IRD) state where all logic and registers are cleared in anticipation of a new floating-point instruction.

2.4.1 Overflow

This exception occurs when the exponent is larger than the largest representable exponent for the data type, after normalizing and rounding. The destination in this case is unaffected and the condition codes, unpredictable.

2.4.2 Underflow

This exception occurs when the exponent is smaller than the smallest representable exponent for the data type after normalizing and rounding. If the floating underflow (FU) bit is set, the destination is unaffected and the condition codes (CCs) are unpredictable; otherwise, the result is zero.

2.4.3 Divide-by-Zero

This exception occurs when the divisor is a zero. The destination is unaffected and the CCs are unpredictable.

2.4.4 Reserved Operand Fault

This exception occurs when one of the operands is reserved. A reserved operand is a negative zero (sign bit = 1, exponent = 0).

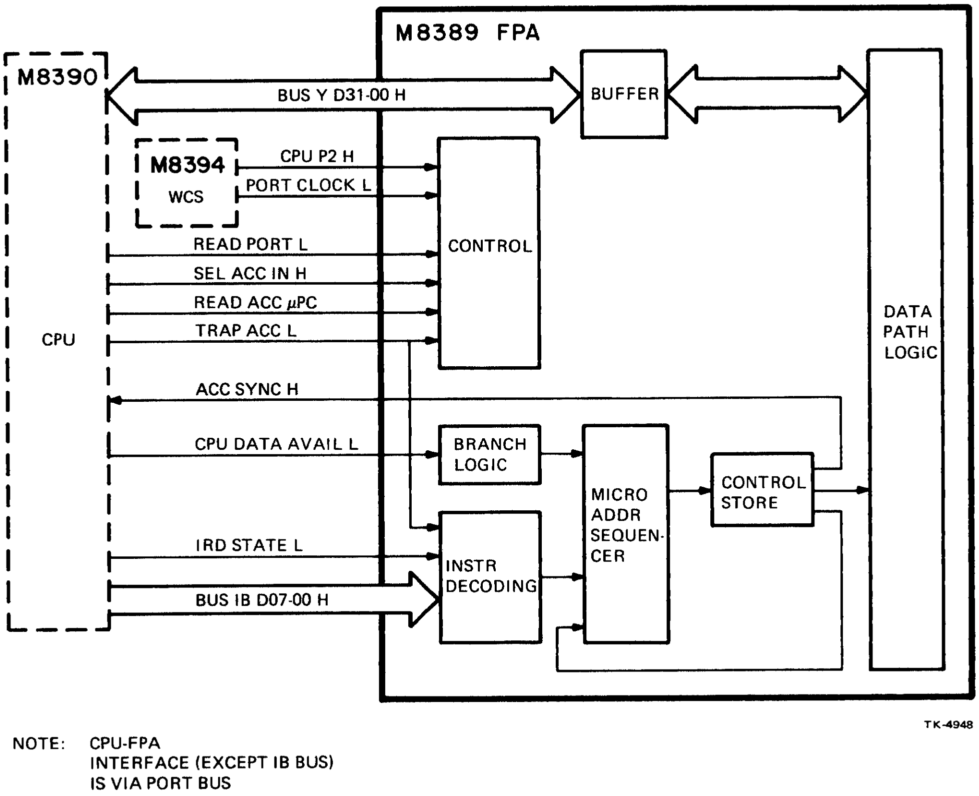
CHAPTER 3
INTERFACING

3.1 GENERAL

The CPU sends the FPA an instruction that indicates what operation and data type (F, D, G, or H) is to be processed. The FPA then sets up its data path logic to perform the required operations. The CPU next loads data (32-bit operands) into the FPA data path logic. After the data is processed, the result is stored by the CPU.

3.2 INTERFACE SIGNALS

FPA-CPU interface signals are illustrated in Figure 3-1, and described in Table 3-1. Timing signals CPU P2 H and PORT CLOCK L are continually applied to the FPA. The CPU controls FPA operation via READ PORT L, SEL ACC IN H, READ ACC UPC L, TRAP ACC L, IRD STATE L, and CPU DATA AVAIL L. ACC SYNC H is the only FPA output (other than the result it puts in the Y-Bus) the FPA sends to the CPU.



NOTE: CPU-FPA INTERFACE (EXCEPT IB BUS) IS VIA PORT BUS

Figure 3-1 FPA-CPU Interface

Table 3-1 Interface Signals

Signal	Description
Y-BUS	32-bit wide bus used for all data transfers to/from the CPU and the FPA.
CPU P2 H	90 ns pulse used to synchronize the FPA to the CPU. The total microcycle for this clock is 270 ns.
PORT CLOCK L	Basic 90 ns clock.
READ PORT L	Control line used by CPU to enable FPA tri-state output buffers.
SEL ACC IN H	Signal used by the CPU to select the FPA. When asserted, enables the FPA to drive the Y-Bus for transfer of result data.
READ ACC UPC L	CPU-generated signal. At the end of the microcycle in which it is issued, the FPA will stop its clocks so that its next microaddress (NUA) will not change. The next time the FPA asserts CPU RCV DATA L, the FPA will drive the Y-Bus with its next microaddress, and the FPA clocks will be restarted.
TRAP ACC L	Signal that forces the FPA to the microaddress present on the Y-Bus ⟨9:0⟩. Used to abort the FPA in cases of memory management aborts, interrupts, etc., and also used to invoke microdiagnostic routines in the FPA.
IB-BUS	Eight-bit wide op code bus.
IRD STATE L	Signal that indicates to the FPA that data on the IB-Bus is an op code.
CPU DATA AVAIL L	CPU signal used for transmitting operands to the FPA.
ACC SYNC H	FPA-generated signal that indicates to the CPU that the FPA is ready. Also used for synchronizing FPA to the CPU for transmitting (data store) data, and for synchronizing transfer of operand data from the CPU during execution of a POLY instruction.

3.3 INTERFACE OPERATION

3.3.1 Op Code Decoding

Figure 3-2 illustrates the timing and functional flow that occurs when the FPA decodes an op code on the instruction bus (IB) during IRD STATE L. Within the FPA, the instruction decoding logic encodes the op code into an initial starting address for the microsequencer. The microsequencer then generates a microaddress for the control store. The control store generates output signals that control the data path logic to handle the operands that will be loaded into it from the Y-Bus.

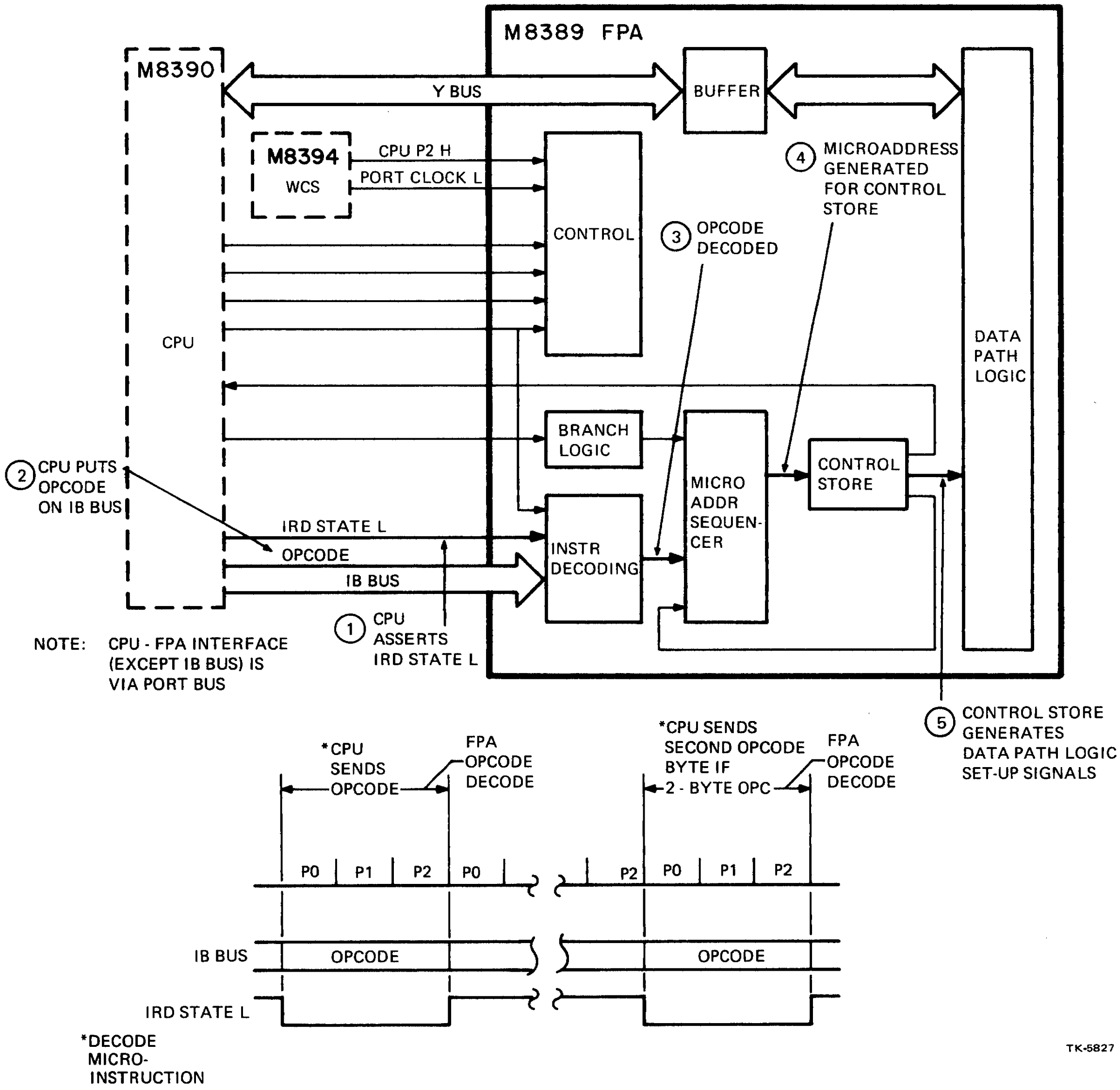


Figure 3-2 Op Code Decoding

3.3.2 Operand Loading

Figure 3-3 illustrates the timing and functional flow that occurs when the CPU loads operands into the FPA. Initially, the CPU asserts CPU DATA AVAIL L, a synchronizing signal that indicates to the FPA that the CPU is putting an operand on the Y-Bus. Within the FPA, CPU DATA AVAIL L is applied to the branch logic.

The CPU DATA AVAIL L signal changes the next microaddress by ORing a one into the least significant bit (LSB). This causes the microsequencer to branch out of the loop it is in. While in this loop (which continually loads the FPA data path and branches on CPU DATA AVAIL L), the ACC SYNC signal is asserted. The CPU ignores the signal when passing data to the FPA except when passing a polynomial coefficient.

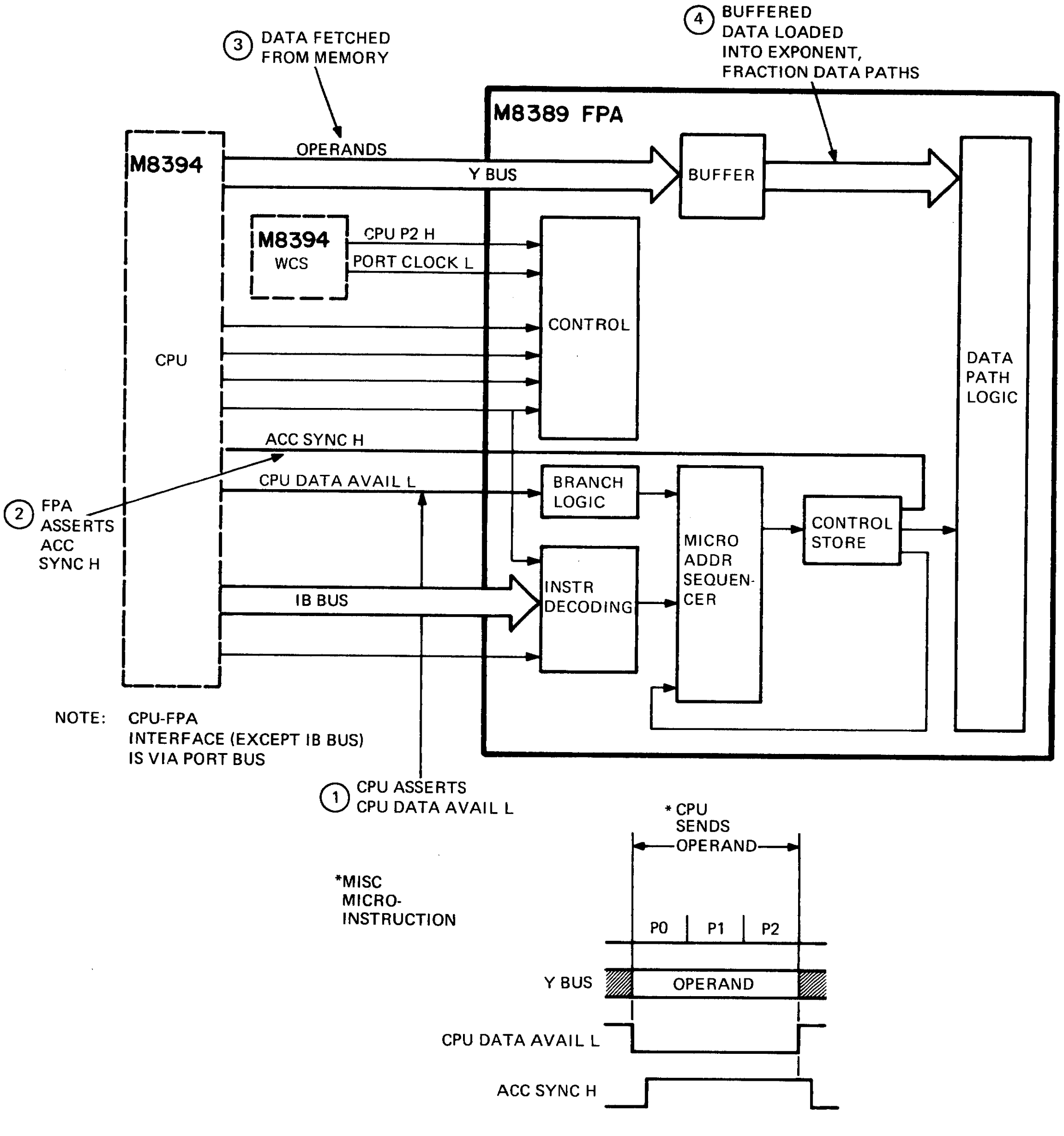


Figure 3-3 Operand Loading

3.3.3 Result Storing

Figure 3-4 illustrates the timing and functional flow that occurs when the FPA sends a result to the CPU. The CPU selects the FPA (since there may be other devices connected to the port bus) via SEL ACC IN H. The CPU then asserts READ PORT L.

The FPA NANDs both SEL ACC IN and the inverse of READ PORT. When the result goes low, the branch logic ORs a one into the LSB of the next microaddress. This causes the FPA to branch out of the loop it was in (which continually passed the result back to the CPU and asserted ACC SYNCH H). The FPA will never drive the CPU Y-Bus unless both SEL ACC IN and READ PORT are asserted.

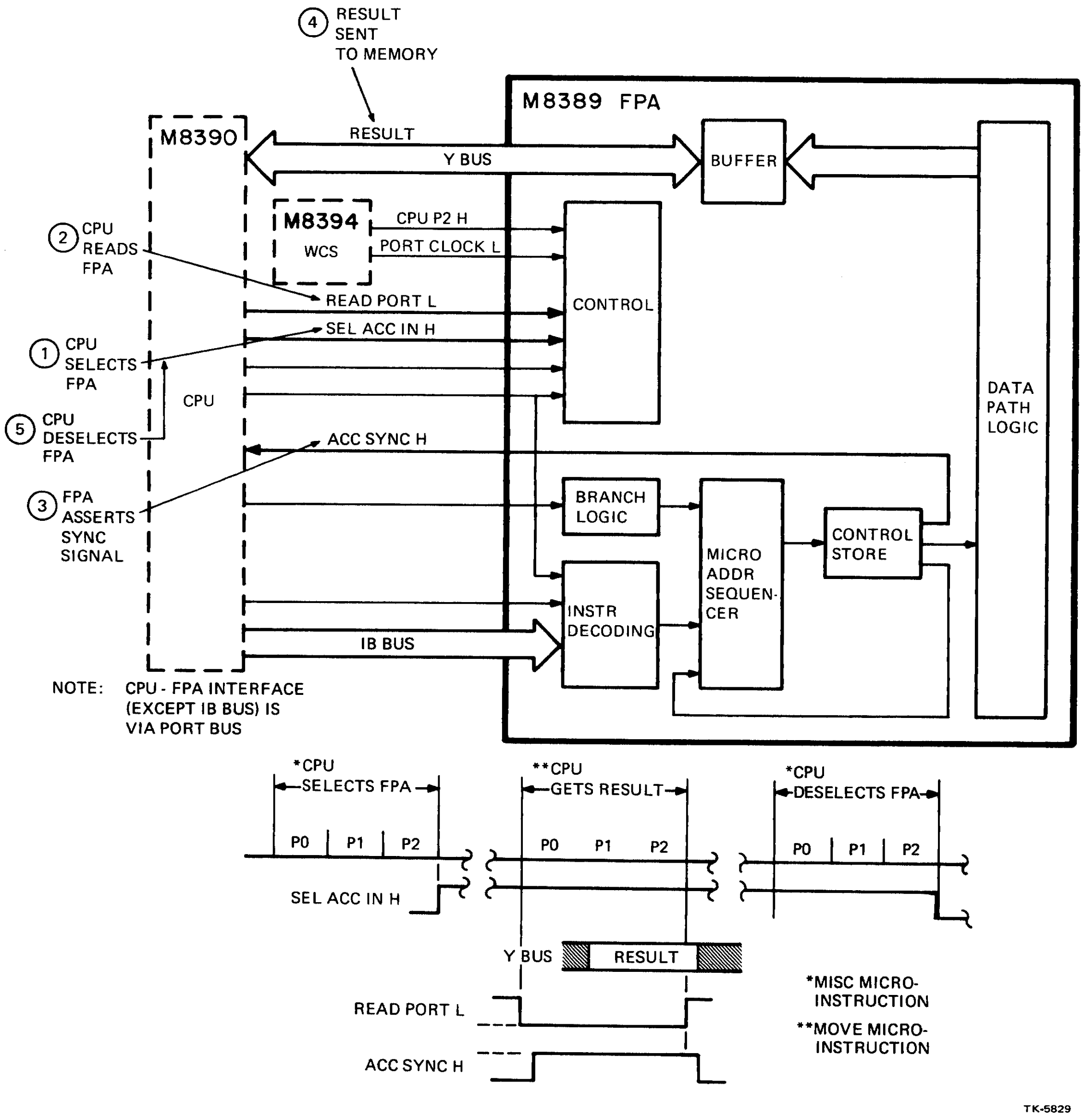


Figure 3-4 Result Storing

3.4 CPU FORCE/READ MICROADDRESS CONTROL

The CPU can inhibit operation of the FPA microaddress sequencer and force (load) a microaddress into the control store. This occurs when the CPU must abort a floating-point instruction due to a memory management error or an interrupt. The CPU can also read the current microaddress that is applied to the control store.

3.4.1 Force Microaddress Control

Figure 3-5 illustrates the timing and functional flow that occurs when the CPU forces a microaddress into the control store. When the CPU asserts TRAP ACC L, the FPA microaddress sequencer output is inhibited and the FPA clocks are slowed (switch from 180 ns to 270 ns) and become synchronized with the CPU. Next, the CPU applies an address on the Y-Bus. This input is gated onto the BUS NUA (09:00) in the FPA and applied to the control store.

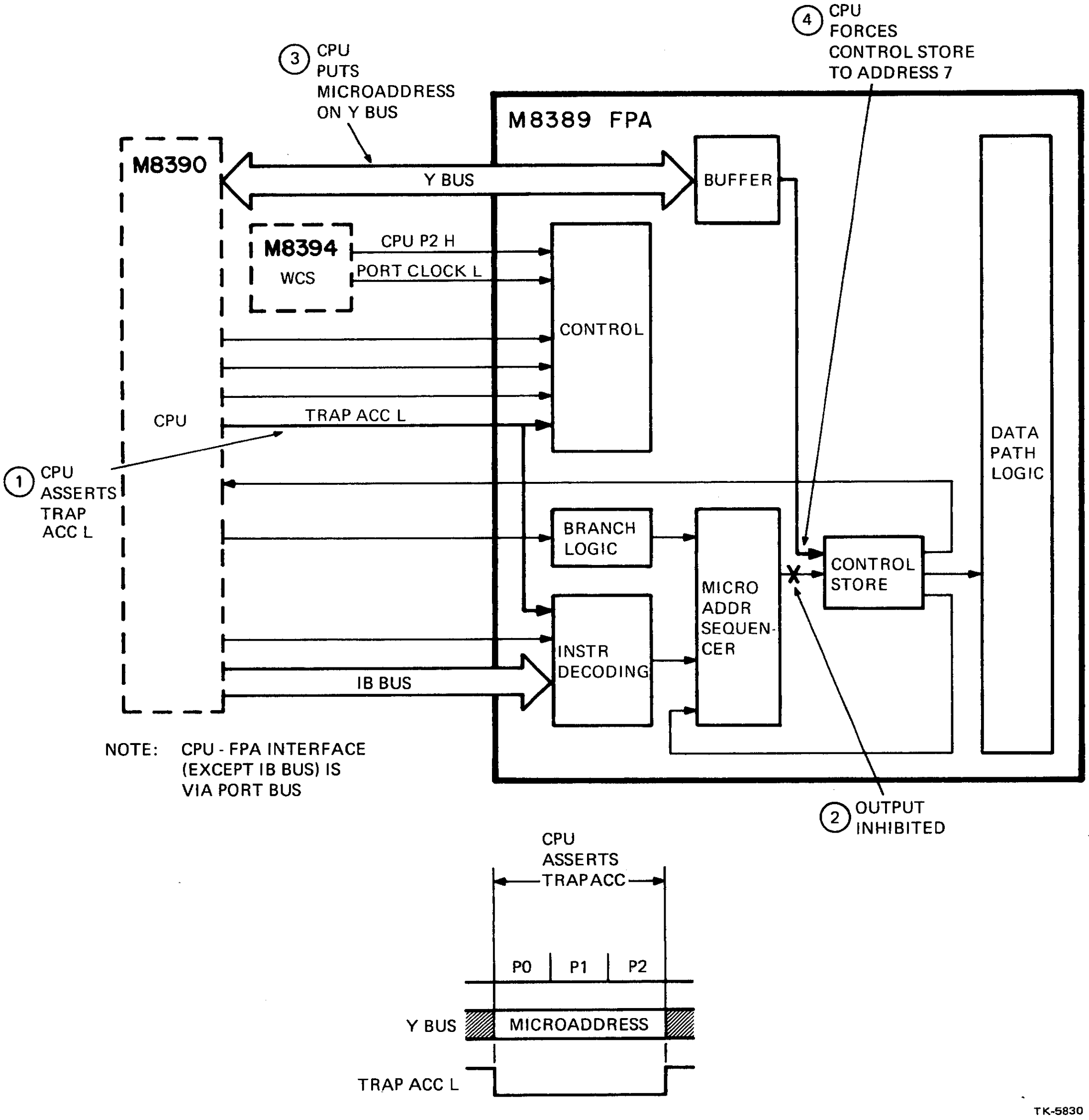


Figure 3-5 Force Microaddress Control

3.4.2 Read Microaddress Control

Figure 3-6 illustrates the timing and functional flow that occurs when the CPU reads the current FPA microaddress being applied to the control store. The CPU initially asserts READ ACC UPC L and then READ PORT L. These signals are gated in control logic in the FPA so the microaddress sequencer output is applied to the Y-Bus (after being buffered).

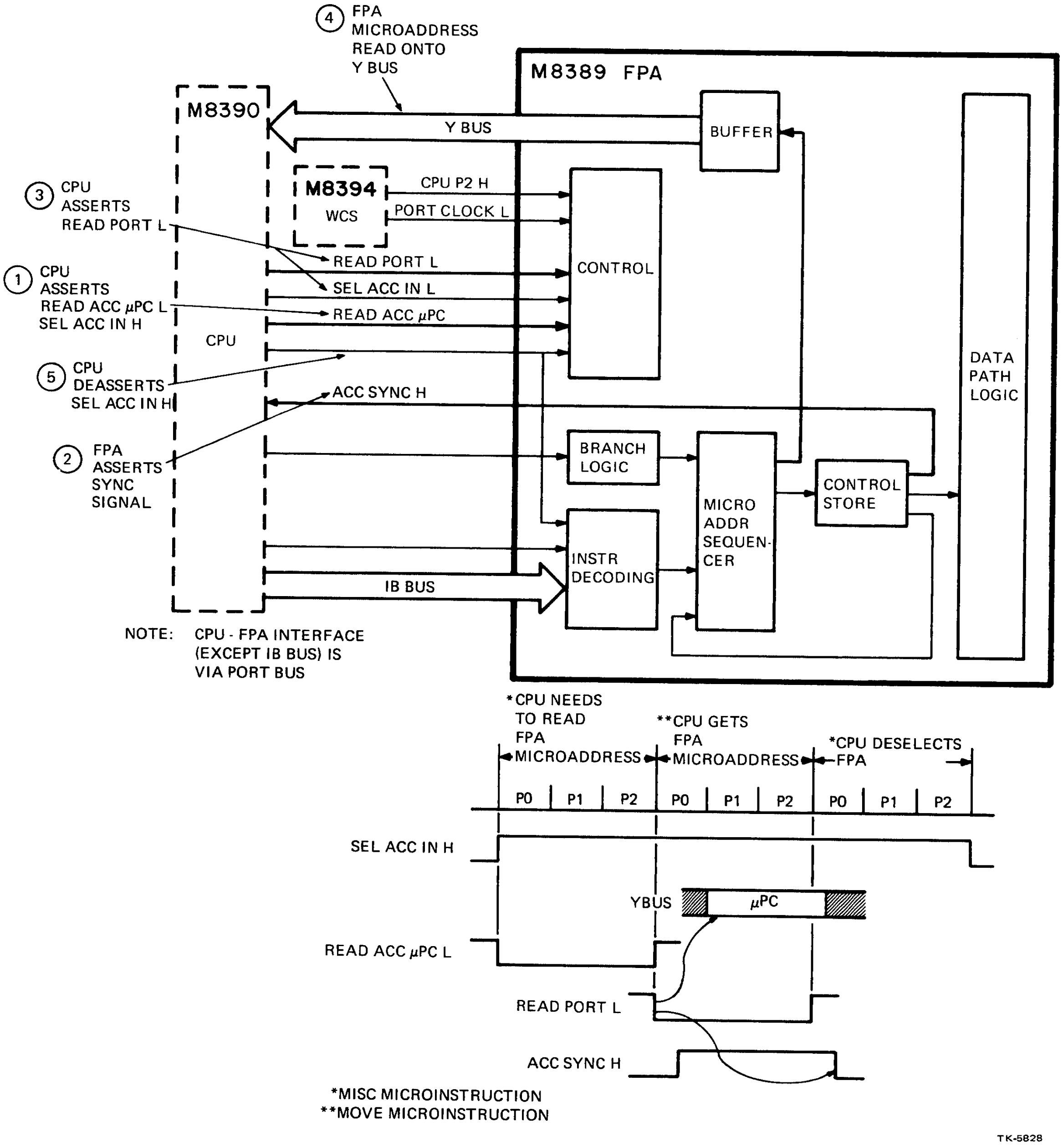


Figure 3-6 Read Microaddress Control

3.5 ERROR REPORTING

The FPA contains microword parity error logic and condition code logic that report status/errors to the CPU.

3.5.1 Parity

The FPA contains odd parity logic that monitors the control store for every microaddress the microaddress sequencer applies to it. If an error is detected, a 3-bit field is used to indicate (via the Y-Bus) what error(s) was detected.

3.5.2 Condition Codes

A condition code, programmable array logic (PAL in the FPA), is used to report errors (among other things) when operands are processed in the data path logic. These errors are:

1. Reserved operand – negative zero
2. Divide-by-zero
3. Floating overflow
4. Floating underflow
5. Parity error

CHAPTER 4
INSTRUCTIONS AND ALGORITHMS

4.1 GENERAL

Table 4-1 lists the FPA instruction set. All of the arithmetic instructions require two operands which are stored in the FPA in temporary storage register locations TEMP 0 and TEMP 2. TEMP 0 corresponds to the sign of the first operand (OP1) and the content of exponent working register (EWR) ET0, and fraction working register (FWR) FT0. TEMP 2 corresponds to the sign of OP2 and EWR ET2, plus FWR FT2.

Table 4-1 FPA Instructions

Instruction	Type	Description
ADD	Arithmetic	Add
CMP	Arithmetic	Compare
SUB	Arithmetic	Subtract
POLY	Arithmetic	Polynomial
DIV	Arithmetic	Divide
MUL	Arithmetic	Multiply
EMOD	Arithmetic	Extend modify
MULL	Arithmetic	Multiply longword
DIVL	Arithmetic	Divide longword
CVT F, D, G, H → B	Convert	Convert from floating to byte
CVT F, D, G, H → W	Convert	Floating to word
CVT F, D, G, H → LW	Convert	Floating to longword
CVT F, D, G, H → ROUNDED	Convert	Floating to longword Rounded
CVT to F from D, G, or H	Convert Precision	Convert D, G, D, or H to F
CVT to D from F or H	Convert Precision	Convert F or H to D
CVT to G from H or F	Convert Precision	Convert H or F to G
CVT to H from F, D or G	Convert Precision	Convert F, D, or G to H
CVT BYTE → F, D, G, H	Convert	Convert byte to floating
CVT WORD → F, D, G, H	Convert	Convert word to floating
CVT L WORD → F, D, G, H	Convert	Convert longword to floating

For arithmetic instruction using huge operands, the fraction part of the word requires two working registers. FWR FT0 and FWR FT1 are used for OP1, and FWR FT2 and FWR FT3 for OP2.

For the two FPA integer arithmetic instructions, operands are stored in FT0 (D47:16) and FT2 (D47:16).

4.2 ARITHMETIC INSTRUCTIONS

4.2.1 Add/Subtract

Before two floating-point numbers can be added or subtracted, (Figure 4-1), the exponents must be made equal (prealigned). If they are not equal, the fraction with the smaller exponent must be right-shifted until the exponents are equal. For each right-shift made to the fraction, the exponent is incremented.

1. Exponents not aligned $(.123 \times 10^{+5}) + (.456 \times 10^{+\underset{\uparrow}{2}})$

   Smaller exponent requiring prealignment

2. Smaller exponent prealigned $.000456 \times 10^5$

3. Numbers added

$$\begin{array}{l} .123 \quad\;\;\; \times 10^5 \\ \underline{.000456 \times 10^5} \end{array}$$

4. Result $.123456 \times 10^5$

At the start of an addition or subtraction, the FPA determines which exponent of two operands is larger, or if they are equal. It does this by subtracting the exponent of OP2 from the exponent of OP1. If the exponents are unequal, the FPA then performs a range test. This test determines whether the larger exponent is so much larger than the smaller that prealignment/addition is unnecessary. This is true if the number of prealignment steps is greater than one, plus the number of bits in the fraction. (For example, for F instructions there are 24 bits in the fraction. If the difference in exponents is greater than 25, prealignment is unnecessary.)

Prior to prealignment, the FPA determines if the operation required is a summation or a difference. A summation occurs for ADD when the two operand signs are the same. Summation also occurs for SUB when the two signs are not the same. Then, if the operation to be performed is a difference, the smaller number is negated before prealignment.

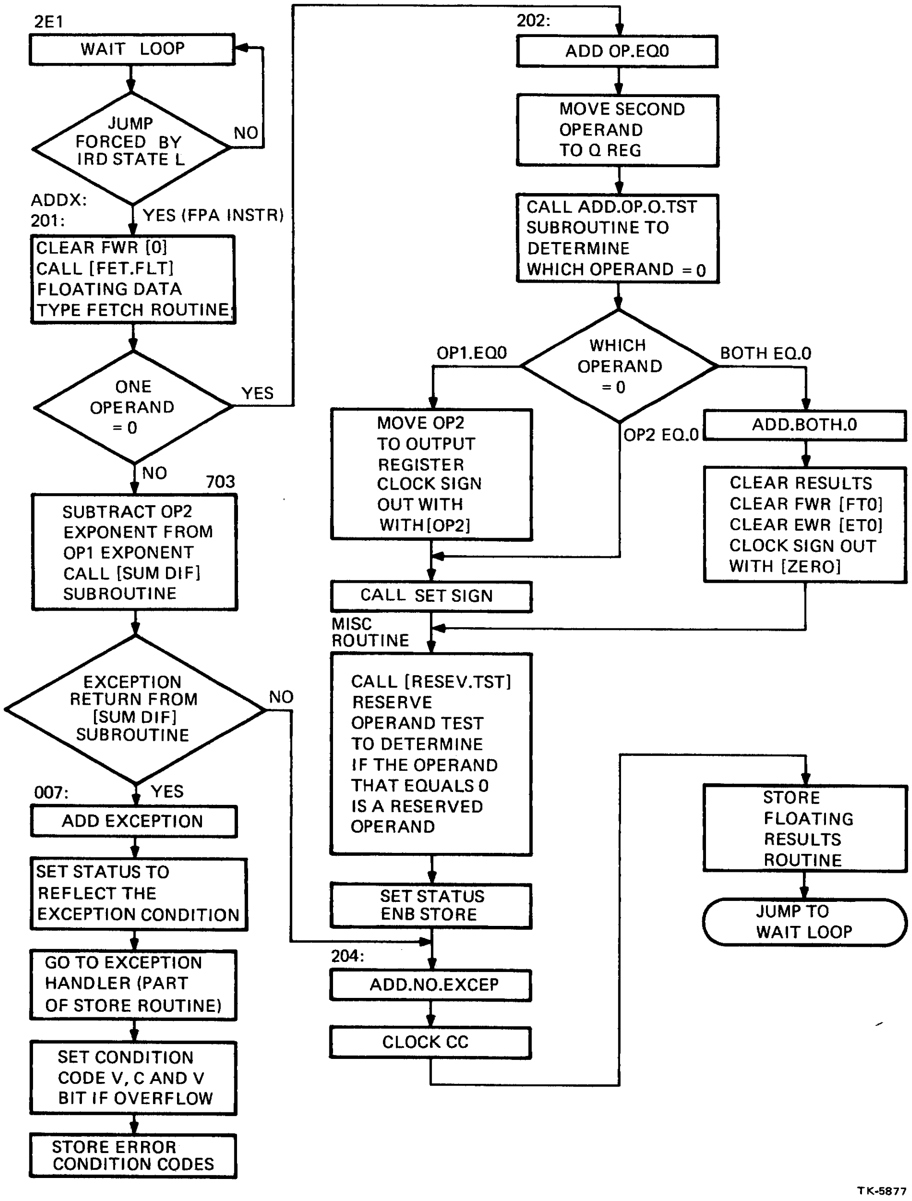

2E1
WAIT LOOP
JUMP FORCED BY IRD STATE L
NO
ADDX:
201:
YES (FPA INSTR)
CLEAR FWR [0]
CALL [FET.FLT]
FLOATING DATA
TYPE FETCH ROUTINE
ONE OPERAND = 0
YES
NO
703
SUBTRACT OP2 EXPONENT FROM OP1 EXPONENT CALL [SUM DIF] SUBROUTINE
EXCEPTION RETURN FROM [SUM DIF] SUBROUTINE
NO
007:
YES
ADD EXCEPTION
SET STATUS TO REFLECT THE EXCEPTION CONDITION
GO TO EXCEPTION HANDLER (PART OF STORE ROUTINE)
SET CONDITION CODE V, C AND V BIT IF OVERFLOW
STORE ERROR CONDITION CODES
202:
ADD OP.EQ0
MOVE SECOND OPERAND TO Q REG
CALL ADD.OP.O.TST SUBROUTINE TO DETERMINE WHICH OPERAND = 0
WHICH OPERAND = 0
OP1.EQ0
BOTH EQ.0
OP2 EQ.0
MOVE OP2 TO OUTPUT REGISTER CLOCK SIGN OUT WITH WITH [OP2]
ADD.BOTH.0
CLEAR RESULTS
CLEAR FWR [FT0]
CLEAR EWR [ET0]
CLOCK SIGN OUT WITH [ZERO]
CALL SET SIGN
MISC ROUTINE
CALL [RESEV.TST] RESERVE OPERAND TEST TO DETERMINE IF THE OPERAND THAT EQUALS 0 IS A RESERVED OPERAND
SET STATUS ENB STORE
204:
ADD.NO.EXCEP
CLOCK CC
STORE FLOATING RESULTS ROUTINE
JUMP TO WAIT LOOP
TK-5877

Figure 4-1 Add Flow (Sheet 1 of 6)

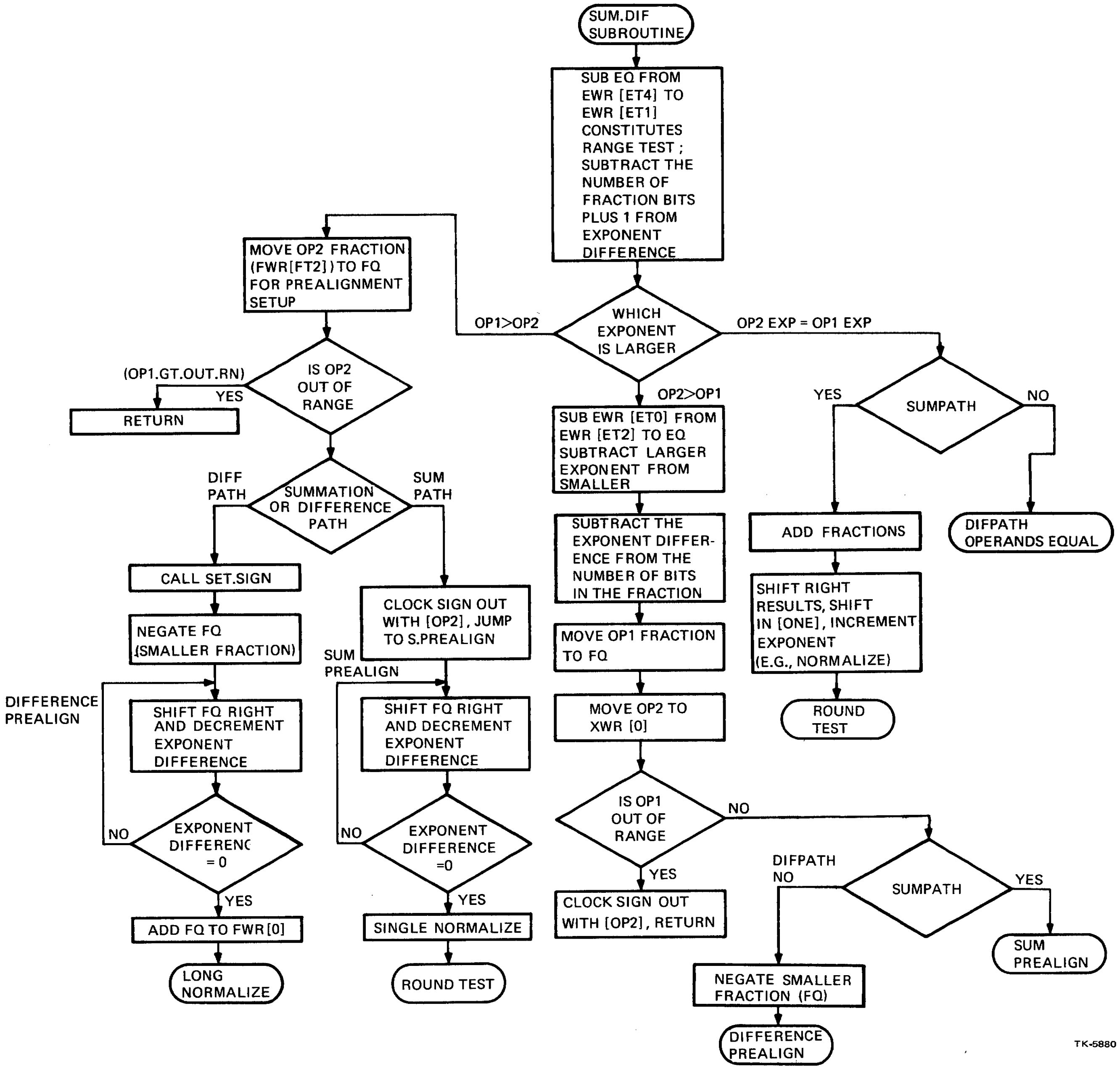

SUM.DIF SUBROUTINE
SUB EQ FROM EWR [ET4] TO EWR [ET1] CONSTITUTES RANGE TEST ; SUBTRACT THE NUMBER OF FRACTION BITS PLUS 1 FROM EXPONENT DIFFERENCE
WHICH EXPONENT IS LARGER
OP1>OP2
OP2 EXP = OP1 EXP
OP2>OP1
MOVE OP2 FRACTION (FWR[FT2]) TO FQ FOR PREALIGNMENT SETUP
IS OP2 OUT OF RANGE
(OP1.GT.OUT.RN)
YES
RETURN
SUMMATION OR DIFFERENCE PATH
DIFF PATH
SUM PATH
CALL SET.SIGN
NEGATE FQ (SMALLER FRACTION)
DIFFERENCE PREALIGN
SHIFT FQ RIGHT AND DECREMENT EXPONENT DIFFERENCE
NO
EXPONENT DIFFERENC =0
YES
ADD FQ TO FWR [0]
LONG NORMALIZE
CLOCK SIGN OUT WITH [OP2], JUMP TO S.PREALIGN
SUM PREALIGN
SHIFT FQ RIGHT AND DECREMENT EXPONENT DIFFERENCE
NO
EXPONENT DIFFERENCE =0
YES
SINGLE NORMALIZE
ROUND TEST
SUB EWR [ET0] FROM EWR [ET2] TO EQ SUBTRACT LARGER EXPONENT FROM SMALLER
SUBTRACT THE EXPONENT DIFFER- ENCE FROM THE NUMBER OF BITS IN THE FRACTION
MOVE OP1 FRACTION TO FQ
MOVE OP2 TO XWR [0]
IS OP1 OUT OF RANGE
NO
YES
CLOCK SIGN OUT WITH [OP2], RETURN
SUMPATH
YES
NO
ADD FRACTIONS
SHIFT RIGHT RESULTS, SHIFT IN [ONE], INCREMENT EXPONENT (E.G., NORMALIZE)
ROUND TEST
DIFPATH OPERANDS EQUAL
SUMPATH
DIFPATH NO
YES
SUM PREALIGN
NEGATE SMALLER FRACTION (FQ)
DIFFERENCE PREALIGN
TK-5880

Figure 4-1 Add Flow (Sheet 2 of 6)

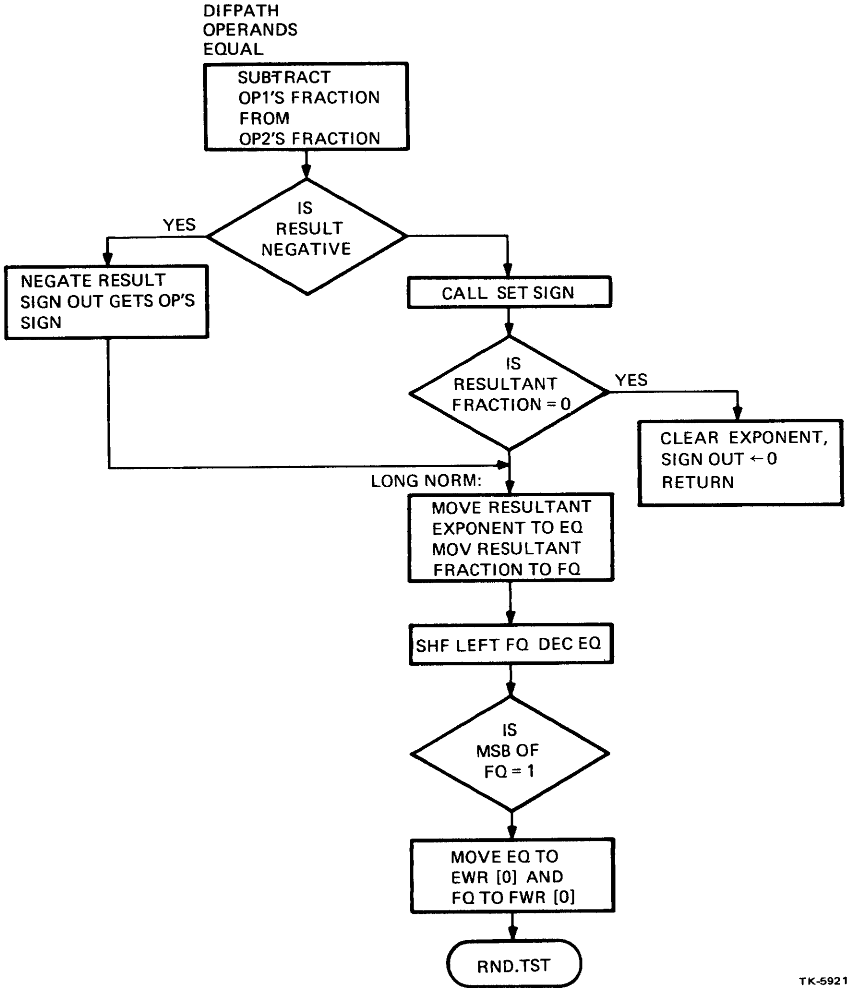
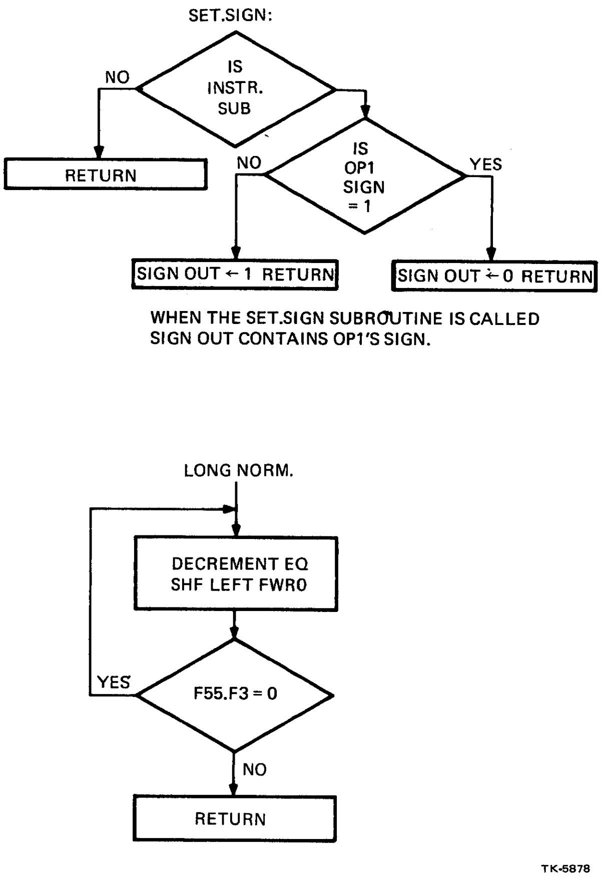

DIFPATH
OPERANDS
EQUAL
SUBTRACT OP1'S FRACTION FROM OP2'S FRACTION
IS RESULT NEGATIVE
YES
NEGATE RESULT SIGN OUT GETS OP'S SIGN
CALL SET SIGN
IS RESULTANT FRACTION = 0
YES
CLEAR EXPONENT, SIGN OUT ← 0 RETURN
LONG NORM:
MOVE RESULTANT EXPONENT TO EQ MOV RESULTANT FRACTION TO FQ
SHF LEFT FQ DEC EQ
IS MSB OF FQ = 1
MOVE EQ TO EWR [0] AND FQ TO FWR [0]
RND.TST
TK-5921

Figure 4-1 Add Flow (Sheet 3 of 6)



WHEN THE SET.SIGN SUBROUTINE IS CALLED SIGN OUT CONTAINS OP1'S SIGN.

Figure 4-1 Add Flow (Sheet 4 of 6)

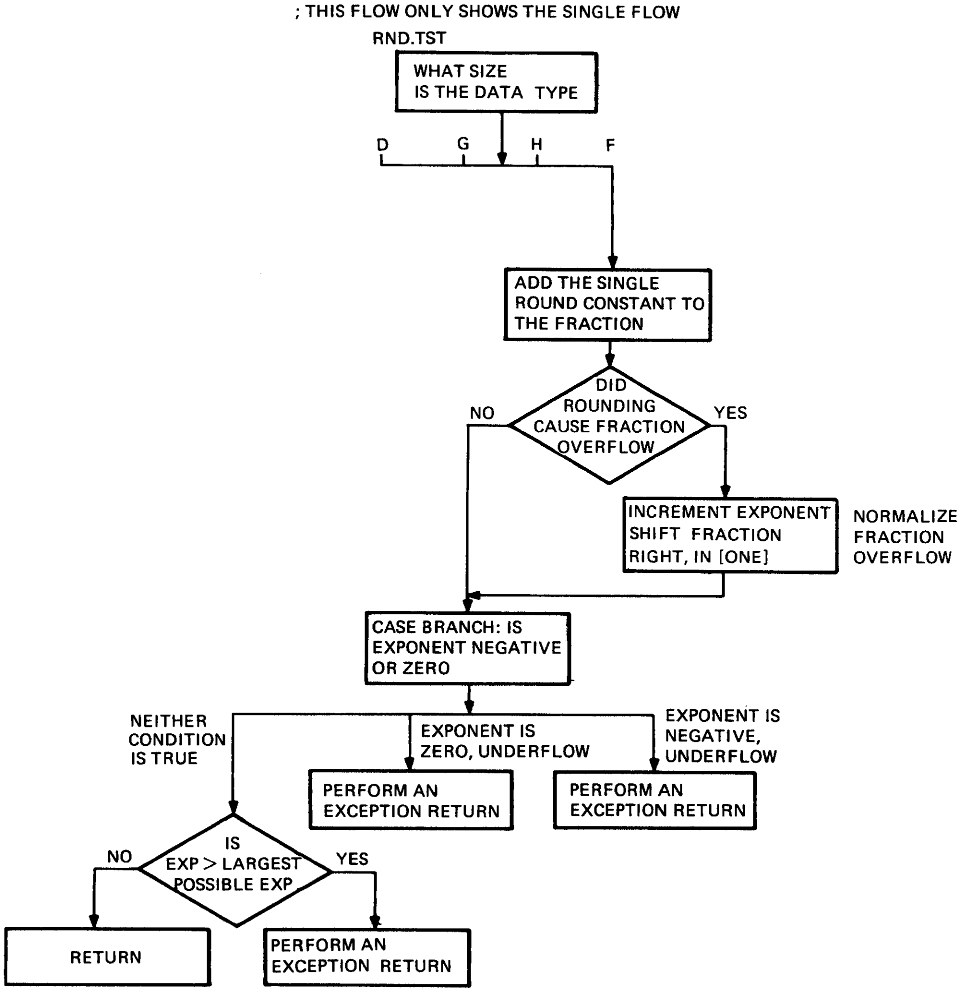
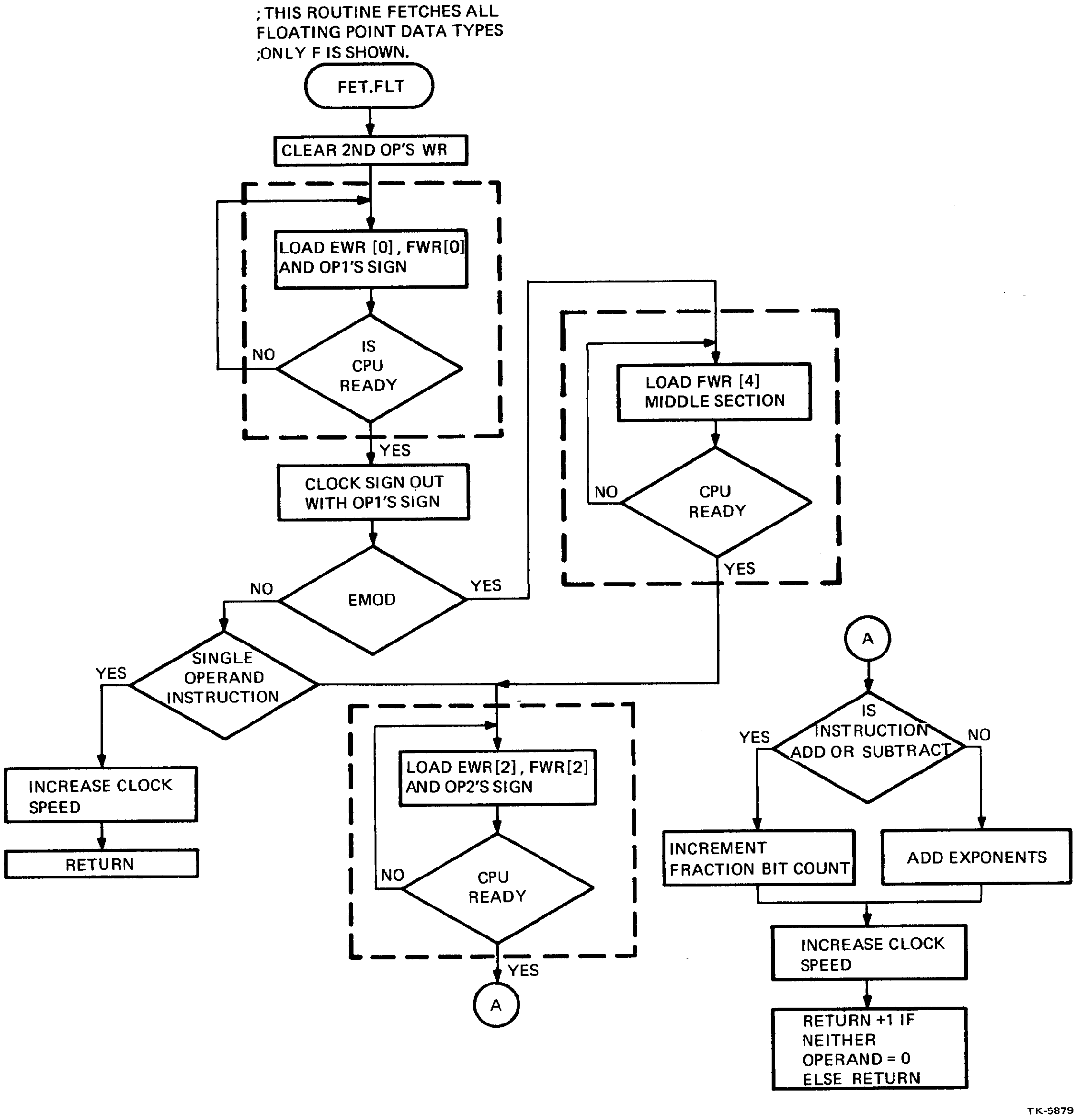

; THIS FLOW ONLY SHOWS THE SINGLE FLOW
RND.TST
WHAT SIZE IS THE DATA TYPE
D
G
H
F
ADD THE SINGLE ROUND CONSTANT TO THE FRACTION
DID ROUNDING CAUSE FRACTION OVERFLOW
NO
YES
INCREMENT EXPONENT SHIFT FRACTION RIGHT, IN [ONE]
NORMALIZE FRACTION OVERFLOW
CASE BRANCH: IS EXPONENT NEGATIVE OR ZERO
NEITHER CONDITION IS TRUE
EXPONENT IS ZERO, UNDERFLOW
EXPONENT IS NEGATIVE, UNDERFLOW
PERFORM AN EXCEPTION RETURN
PERFORM AN EXCEPTION RETURN
IS EXP > LARGEST POSSIBLE EXP
NO
YES
RETURN
PERFORM AN EXCEPTION RETURN
TK-5881

Figure 4-1 Add Flow (Sheet 5 of 6)



; THIS ROUTINE FETCHES ALL FLOATING POINT DATA TYPES
;ONLY F IS SHOWN.
FET.FLT
CLEAR 2ND OP'S WR
LOAD EWR [0], FWR[0] AND OP1'S SIGN
IS CPU READY
NO
YES
CLOCK SIGN OUT WITH OP1'S SIGN
EMOD
NO
YES
SINGLE OPERAND INSTRUCTION
YES
INCREASE CLOCK SPEED
RETURN
LOAD FWR [4] MIDDLE SECTION
CPU READY
NO
YES
LOAD EWR[2], FWR[2] AND OP2'S SIGN
CPU READY
NO
YES
A
A
IS INSTRUCTION ADD OR SUBTRACT
YES
NO
INCREMENT FRACTION BIT COUNT
ADD EXPONENTS
INCREASE CLOCK SPEED
RETURN +1 IF NEITHER OPERAND = 0 ELSE RETURN
TK-5879

Figure 4-1 Add Flow (Sheet 6 of 6)

To prealign the fraction with the smaller exponent, the exponent difference is placed in the exponent Q-register (EQ) and the smaller fraction is placed in the fraction Q-register (FQ). FQ is right-shifted and EQ is decremented until it is zero, at which time the fraction is properly aligned for the addition.

After prealignment, the numbers are added and then normalized. Normalization consists of aligning the MSB of the resultant fraction with the MSB of the fraction data path.

The sign of the result is set according to Table 4-2.

If the exponents are equal, the fractions are added when the operation is a summation, or subtracted when the operation is a difference. If the operation was a difference, the result must be tested for zero, in which case the answer is a zero.

The result is rounded and tested for underflow or overflow after the addition and normalization have been performed.

Table 4-2 Add/Subtract Sign Calculation

	Original Signs		**Resultant Sign**	
	OP1 Sign	OP2 Sign	OP1 > OP2	OP2 > OP1
Add	+	+	+	+
	+	−	+	−
	−	+	−	+
	−	−	−	−
Sub	+	+	−	+
(OP2-OP1)	+	−	−	−
	−	+	+	+
	−	−	+	−

4.2.2 Compare (CMP) Instructions

A compare (CMP) instruction compares two operands by subtracting the second operand from the first. The compare instruction loads the results in the condition codes, where

N ← 1	if	OP1 is less than OP2
Z ← 1	if	OP2 = OP1
V ← 0		
C ← 0		

CMP Algorithm:

1. If signs are not the same, then N ← OP1 sign, and the condition codes (CC) are stored.

2. If signs are the same, subtract the exponents OP1 EXP – OP2 EXP

3. If OP1 EXP > OP2 EXP N ← OP1's sign, store CCs.
   If OP1 EXP < OP2 EXP N ← Not [OP1's sign], store CCs.

4. If OP1 EXP = OP2 EXP, subtract fraction

5. If fraction = 0, the Z bit gets a one (Z ← 1), store CCs.

   If MSB of fraction = 0 but fraction ≠ 0, the N bit gets the sign of OP1 (N ← OP1's sign), store CCs.

   If MSB of fraction = 1, N ← Not [OP1's sign], store CCs.

4.2.3 Polynomial (POLY) Instruction

The Polynominal (POLY) instruction evaluates a polynomial expression of the form

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 \ldots$$

where the largest possible degree of x is 31. Three operand specifiers are required.

1. Arg – the argument, (e.g., x)

2. Degree – the highest power x is to be raised to

3. Tbladdr – the address of a table of coefficients. The first coefficient in the table is actually the last coefficient in the polynomial.

The polynomial expression is calculated as follows:

$[[[c\,(d) * x + c\,(d\text{-}1)] * x + c\,(d\text{-}2)] * x \ldots. + c\,(1)] - x + c(0)$

where c (d) = the coefficient of the largest powers of x.

After the multiplication, more than the normal number of bits are kept for the addition:

F: 31 bits

D: 63 bits

G: 63 bits

H: 127 bits

The next coefficient is then added to the product, the number is rounded, and exceptions are checked for. The next iteration is then initiated.

The FPA executes the POLY instruction by performing a multiply/addition iteration and then passing the result back to the CPU. This automatically starts the next iteration. If the instruction is done, the CPU must abort the FPA.

POLY Algorithm:

Initialization

1. Store argument in ET8, FT8 (FT9 for Huge).
2. Store first coefficient in ET2, FT2.
3. Sign out ← OP1 sign XOR OP2's sign.

NOTE
OP1 sign reflects the sign of the argument.

4. Go to POLY iteration.

POLY Iteration

1. Move argument to ET0, FT0, (FT1).
2. Call (MUL.ROUTINE).
3. Fetch next coefficient and load into ET2, FT2 (FT3 for Huge).
4. Call ADD routine.
5. Round and test for exception.
6. Truncate to data type, and store in ET2, FT2 (FT3).
7. Store condition codes and results.
8. Sign out ← Sign out XOR OP1's sign.
9. Go to POLY iteration.

NOTE
If an underflow occurs at the end of a MUL/ADD iteration, the partial results are cleared, and an error code is stored. If the FU bit is set, the CPU will abort the FPA. The FPA automatically starts the next iteration. For overflow, the FPA stores the error code and stops execution.

4.2.4 Divide (DIV) Instruction

4.2.4.1 DIV – For a divide operation the quotient ← OP2/OP1.

DIV Algorithm:

1. Sign ← OP1 SIGN XOR OP2 sign.
2. Clear FQ.
3. Load EQ with the fraction bit count.
4. Subtract the OP1 fraction from the OP2 fraction and then go to a DIV loop.

DIV Loop:

If previous result was positive:

a. Shift FQ left, shift in one.
b. Subtract OP1 from OP2.
c. Decrement EQ; if NEQ.0 go to DIV loop.

If previous result was negative:

a. Shift FQ left, shift in zero.
b. Add OP1 to OP2.
c. Decrement EQ; if NEQ.0 go to DIV loop.

DIV Loop Ends.

5. Normalize.
6. Round.
7. Set the condition code bits and store results.

4.2.4.2 DIVL Instruction – The DIVL instruction is for division of an integer by a longword only.

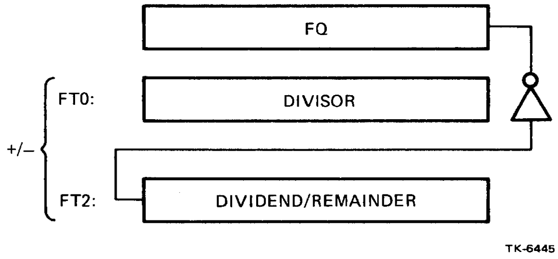
DIVL Algorithm:

1. Since the integers can be in 2s complement form, it is necessary to check for negative numbers. If an operand is negative, it is negated and ET1 is incremented (it was initialized to 0). Thus, if ET1 = 1 after both operands have been checked, and negated if necessary, then the result should be negative.

2. Is dividend ⩾ the divisor? If not, then results = 0.

3. Align the MSB of both dividend and divisor with FRAC47. Initialize EQ to 1 and increment EQ for each alignment shift the divisor requires over that of the dividend. This yields the loop count for the divide loop.

DIVIDE Loop:



Subtract (ADD) the divisor from the dividend (remainder). The inversion of the sign bit of the result is the next quotient bit, and it also controls the ALU function. After the divide loop, ET1 is examined. If ET1 equals 1, the result is negated. Overflow is then checked by examining FRAC47 for positive numbers. If FRAC47 equals one for positive numbers, then an overflow occurred.

4.2.5 Multiply (MUL) Instruction

4.2.5.1 MUL Algorithm – The MUL instruction executes MULF, D, G and H. The MUL algorithm is as follows:

1. Sign ← OP1's sign XOR OP2's sign.

2. Place OP1 (multiplier) in FQ.

3. Clear FT4 (product register).

4. Load EQ with the fraction bit count.

5. Shift FQ right.

NOTES

- **If LSB = 1, add OP2 to FT4 and shift right.**

- **If LSB = 0, shift FT4 right.**

6. Decrement EQ; If NEQ.0, go to 5.

7. Move FT4 (product) to FT0.

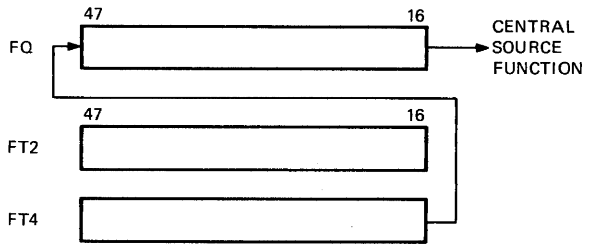
8. Normalize
When the fraction is normalized, the exponent is adjusted at the same time. For every left-shift, the exponent is decremented; for every right-shift, the exponent is incremented.

9. Round
   The FPA always rounds the result of a floating-arithmetic operation. This is accomplished by adding a round constant to the result. The round constant depends on the data type, and will have a one in the bit position which is one less than the LSB. (For example, for F the rounding constant will be all zeros, with a one in bit position 31).

10. Set CCs and store.

NOTE
The LSB of the multiplier depends on the data type. The Multiply/Divide (MUL/DIV) PAL selects that LSB according to the data type.

4.2.5.2 MULL Instruction – The FPA MULL instruction is an integer multiply for longwords only. An integer multiply involves basically the same algorithm as MUL float, except it uses the integer data path.



The test for overflow is also different: FQ at the end of the multiply should be the sign extension of the sign bit (FRAC47) of FT4. If it is not, an overflow has occurred.

4.2.6 Extended Precision Multiply and Integerize (EMOD)

The main function of the EMOD instruction routine is to multiply the multiplier (mier) extension by the multiplicand (mand), set up to use the multiply loop subroutine for the remaining mier bits, and the CVT.FLT subroutine. This flow also contains the zero operand handler, condition code setting, and an exception handler.

The EMOD operation is as follows:

```
          OP1     OP2           OP3
            ↘      ↓             ↓
TEMP ←    (MIER#MIER.EXT)*(MAND)
                ↖
            (CONCATENATE)
```

The MIER.EXT is a byte for F and D, 11 bits for grand (left-justified), and 15 bits for huge (left-justified).

There are two results to this instruction:

1. Fraction (same data type as instruction)
2. Integer (longword)

The hardware is set up so that the multiplier extended (MIER.EXT) is loaded into bits 32:16 of FT4. A microcode function can force the MUL/DIV PAL to select Q16 as the default LSB of the multiplier. Thus, the multiplier extension is multiplied and then OP1 is multiplied. This allows the MUL routine to be shared.

The EMOD flow is as follows:

1. Load FT4 into FQ – (MIER.EXT → FQ).
2. EQ ← loop count (8 = F, D, 11 = G, 15 = H).
3. Set Q16 default.
4. Perform MUL loop until EQ = 0; MUL loop is same as in MUL routine.
5. FQ ← FT0; FQ gets multiplier.
6. EQ ← integer bit count.
7. Call MUL routine.
8. Set up for integerize routine.
9. Call integer routine.
10. Normalize fraction.
11. Round.
12. Test for integer overflow.
13. Set CCs and store.

4.3 CONVERSION INSTRUCTIONS

4.3.1 Floating-Type-to-Integer Conversion

The two FPA instructions, CVT(F, D, G, H) to (B, W, L) CVTR(F, D, G, H, L) convert any floating data type to any integer data type.

All of the conversion instructions are basically similar; the major difference for the various data types is the loop counts.

If the floating-point number is too large to be represented in integer form, the V-bit will be set, and the integer results will reflect the least significant bits of the fraction.

The CVT flow is as follows.

1. Subtract the bias from the exponent; this will indicate the number of integer bits.

   EQ ← ET0 – ET4

   where ET0 = exponent
   ET4 = exponent bias

2. If EQ is negative, there are no integer results. Store a 0.

3. If EQ is not negative, test for overflow.

   EQ = ET0-ET4 (number of bits in the integer)
   E7 ← ET6-EQ

   where ET6 = integer bit count (e.g., 32 for longword).

4. If ET7 is not equal to or less than 0, go to convert loop.

NOTE
ET7 = fraction bit count (number of integer bits).

5. If the number of integer bits is greater than the integer bit count, the number is too large to fit in resultant data type.

6. If ET7 ⩾ zero, then test for significance. (That is, will any integer bits show up in results?)

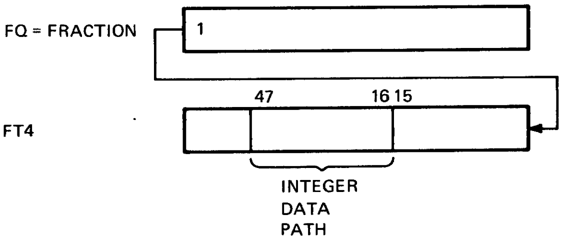
   ET7 ← ET7-ET4 ET7 = number of integer bits in data type of results.

   ET4 = number of integer bits in results.

7. If ET7 < 0, then the result = 0 and the V-bit should be set.

   If ET7 ⩾ 0, then the V-bit should be set; go to the convert loop.

   Convert Loop: Move FT0 to FQ



TK-6444

Right-shift FQ and FT4 the number of times specified by EQ, which contains the number of integer bits.

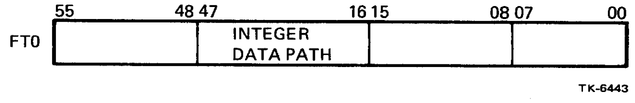
At the end of the convert loop, the number must be aligned with the fraction data path by 12 double shifts.

4.3.2 Integer-to-Floating Type Conversion

The FPA CVT(B, W, L)(F, D, G, H) instruction converts integer to floating type data.

Any integer data type can be converted to any floating data type without overflow or underflow. Because the CVTLF convert instruction can lose significance, this particular convert instruction requires rounding.

1. The integer is loaded into the integer data path



2. Integer MSB is aligned with FRAC55.

   For byte the MSB = 23
   For word the MSB = 31
   For longword the MSB = 47

   This requires:

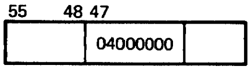
   4 double left-shifts for longword.
   12 double left-shifts for word.
   16 double left-shifts for byte.

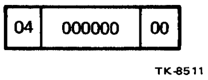
3. After the integer is aligned with FRAC55, the MSB is checked; if it equals 1 the number is negated and the sign bit is set.

4. EQ ← Floating bias plus the number of integer bits in the integer data type.

5. The number is normalized (and rounded if CVTLF), CCs set, and result stored.

Example: CVTLF where LW = 4000000

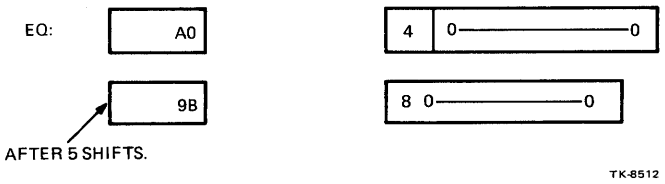
1. Load FT0: 

2. Align FRAC 47 with FRAC 55: 

3. Load EQ with bias plus number of integer bits: EQ ← 80 + 20.

4. MSB of fraction = 0, therefore sign ← 0.

5. Normalize fraction.



4.3.3 Precision Conversion

There are four FPA instructions that convert one floating-point data type to another. They are:

- CVTF (D, G, H)
- CVTD (F, H)
- CVTG (F, H)
- CVTH (F, D, G)

To convert from one floating-type to another:

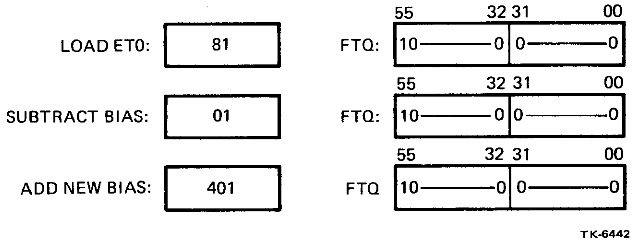
1. Subtract the bias from the exponent, where the bias is the original bias.

2. Add the new bias.

3. Round, if necessary (e.g., CVTFD does not require rounding).

4. Check for overflow or underflow.

Example: CVTFG 4080



No overflow or underflow (not possible for this convert)

Adjust grand number and store results: 4010

CHAPTER 5
THEORY OF OPERATION

5.1 GENERAL

The major circuit in the FP-11/730 (Figure 5-1) is data path logic that processes variable length operands. The operands are passed to the FPA from the CPU in 32-bit sections via the CPU Y-Bus. The FPA buffers the Y-Bus onto its BUS FPA. The data path consists of exponent and fraction sections (fields), plus sign and condition code control sections. The data path logic functions in accordance with control signals generated in a control store.

Floating-point instructions to be processed by the FPA are received from the CPU via an IB-Bus as BUS IB D7:0 and are applied to an instruction decoding/encoding circuit. This circuit encodes a floating-point op code into an address that is applied to a microsequencer circuit, as DECODE ROM 4:0 H. The microsequencer then generates a target address (BUS NUA 9:0 H) that accesses a certain 48-bit microword in the control store. The accessed microword gets clocked with control store registers which produce signals that set up the data path logic for operand processing.

During instruction execution for each control store microword access made, a 10-bit (CS9:0) micropointer field (UPF) in the 48-bit microword is applied to a register in the microsequencer. In most instances, the UPF is used in the microsequencer as the base for the next microaddress that will be generated and applied to the control store.

The five LSB of the 10-bit micropointer field that is applied to the microsequencer can be branched on, in accordance with status bits generated by the data path logic and instruction type signals. The two LSB (1:0) of the micropointer field is normally branched on via a branch control circuit. An extended branch function allows status signals to be ORed in with the next three LSB bits (4:2) in the micropointer field. Thus, a maximum of five bits can be branched on.

Parity logic in the FPA monitors each word accessed from the control store. If a parity error is detected the parity logic generates an output (FORCE ADDR LOW) that forces all ten of the microsequencer output lines to logical 0. This all-zero output is the starting address of a parity handler routine and is applied as the next microaddress to the control store.

Two buffers in the FPA function as a force/read circuit used during diagnostics to read the microsequencer control store address (BUS MUA 9:0 H) output onto the Y-Bus (as BUS Y D9:0H) for subsequent checking in the CPU. The circuit is also used to force a CPU-generated microaddress (from the Y-Bus) into the control store as the next microaddress. These force/read operations are used to test the microsequencer, control store, and data path logic. The force function is also used to abort the FPA and to execute some instructions.

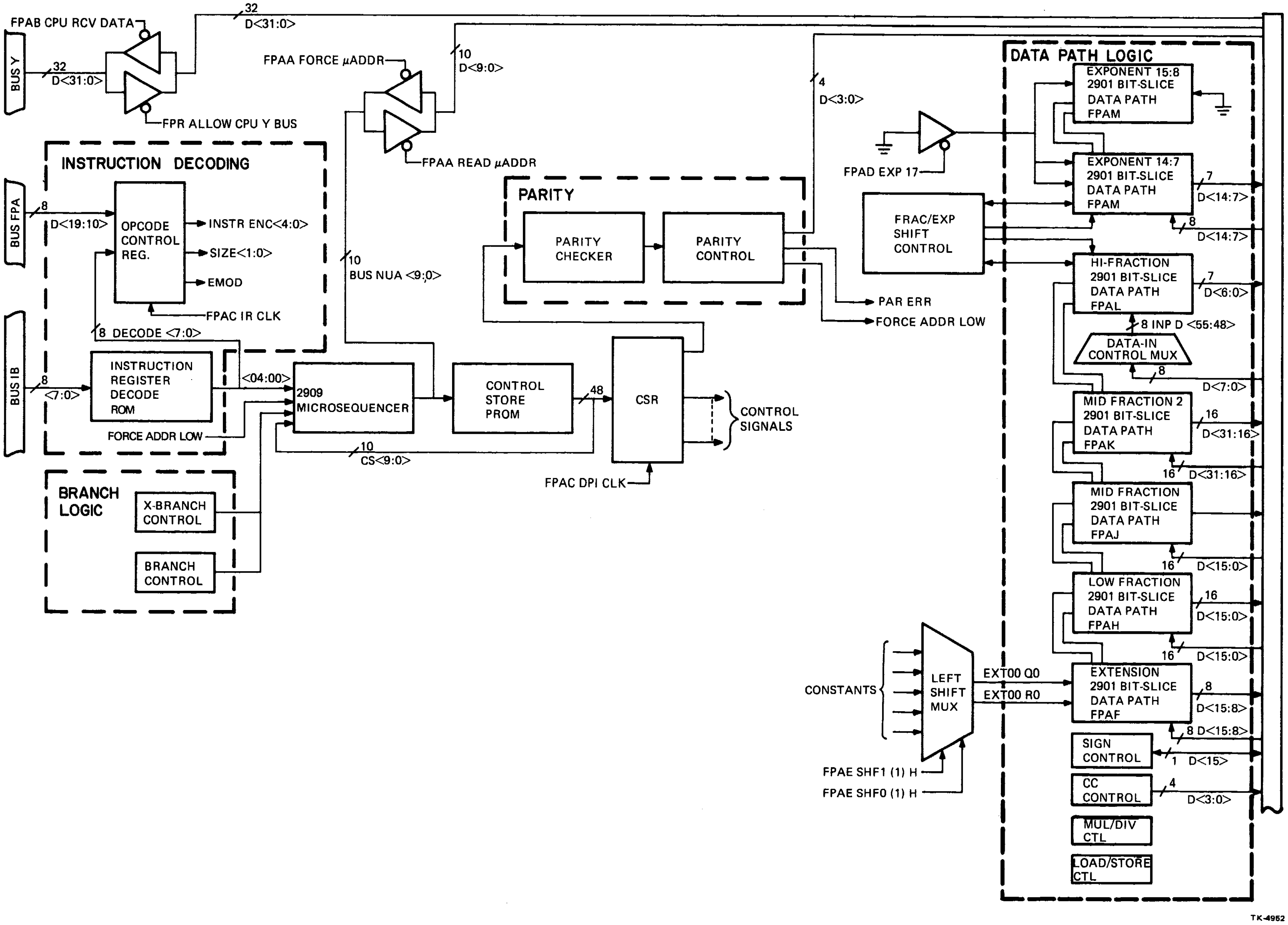


Figure 5-1 FPA-11/730 Block Diagram

5.2 DATA FLOW

The CPU fetches op codes, puts them on the IB-Bus, and after the FPA decodes them, it (FPA) jumps to a microcode routine which executes the instruction. The CPU next sends the FPA operands via the Y-Bus. The FPA then operates on the data input in accordance with the instruction decoded from the operation code on the IB-Bus. The FPA result is then put on the Y-Bus and sent to the CPU.

As the FPA data path logic operates on the operands, it continually sends status signals to branch logic. These signals effect branches that modify the microaddress, prior to gating the microaddress onto BUS NUA ⟨09:00⟩.

During an FPA-CPU data transfer, the CPU aborts the FPA if certain conditions occur. Also, during the data transfer the FPA reports exceptions or error conditions to the CPU via the Y-Bus until the data transfer has completed.

5.2.1 Operand Fetching

When the operands are being fetched, the FPA data path logic is conditioned to operate on data that will appear on the Y-Bus. Initially, an operation code decoded from the IB-Bus addresses a decode ROM in the FPA instruction register. The result is a 5-bit field that is applied to a 2909 microsequencer. The microsequencer then generates a BUS NUA 9:0 output that is applied to the control store PROM. The microword selected from the PROM causes a 48-bit field (microword) to select certain CSR data path control signals. The signals effect the following conditions:

1. The 2901s in both the fraction and exponent data paths are set up to clear the exponent working register EWR (0) and fraction working register FWR (0) so that the first operand (OP1) to appear on the Y-Bus can be loaded into them.

2. A load signal will be set to enable loading of the EWRs and FWRs. This signal is the result of certain values of CLK and MOD fields in the microword accessed from the control store PROM.

NOTE
The load signal is always cleared at the beginning of every instruction.

3. Another BUS NUA 9:0 input applied to the control store PROM will access the appropriate fetch routine. In the FPA microcode this would appear as:

   CALL (FET.FLT)
   or
   CALL (INT.FLT)

Once in the fetch routine, a microword will executes that continually loads a data path logic working register (WR) until the CPU asserts CPU DATA AVAIL L.

Figures 5-2 and 5-3 illustrate how an operand is loaded into the data path logic. For those instructions whose operands are more than one longword (D, G, or H), the FPA will become synchronized with the CPU on the first section, and then expect the remaining longwords to be passed in every other microcycle that follows, without further synchronization.

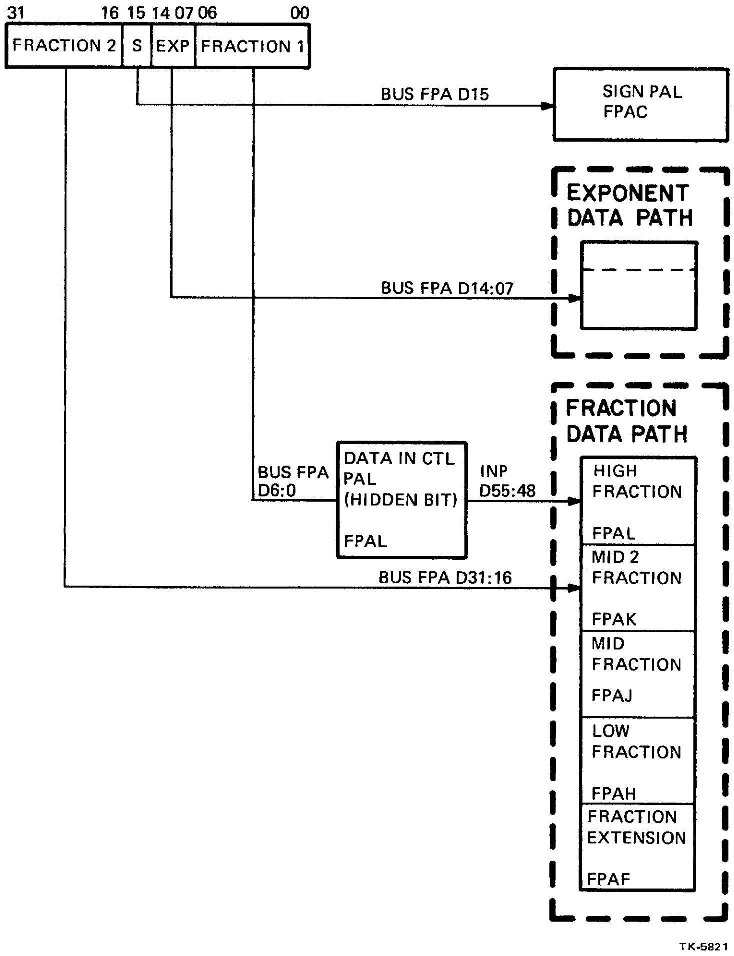


Figure 5-2 Single Format Loading

After all data has been fetched the FPA clock speed will be increased from 270 ns to 180 ns. This increase occurs at the beginning of an instruction execution routine.

Because the exponent of grand and huge data is not totally aligned with the exponent data path, part of it must be loaded into the fraction data path. This part must later be shifted into the exponent data path. A grand adjust microroutine will adjust both operands simultaneously. This is accomplished by placing OP2 into the exponent Q-register (EQ) and into the fraction Q-register (FQ), and then shifting both EQ and FQ while shifting working registers EWR (0) and FWR (0), which contain OP1. A fraction shift control circuit will then direct the MSB of FQ and FWR (0) to the shift-left inputs of EQ and EWR (0).

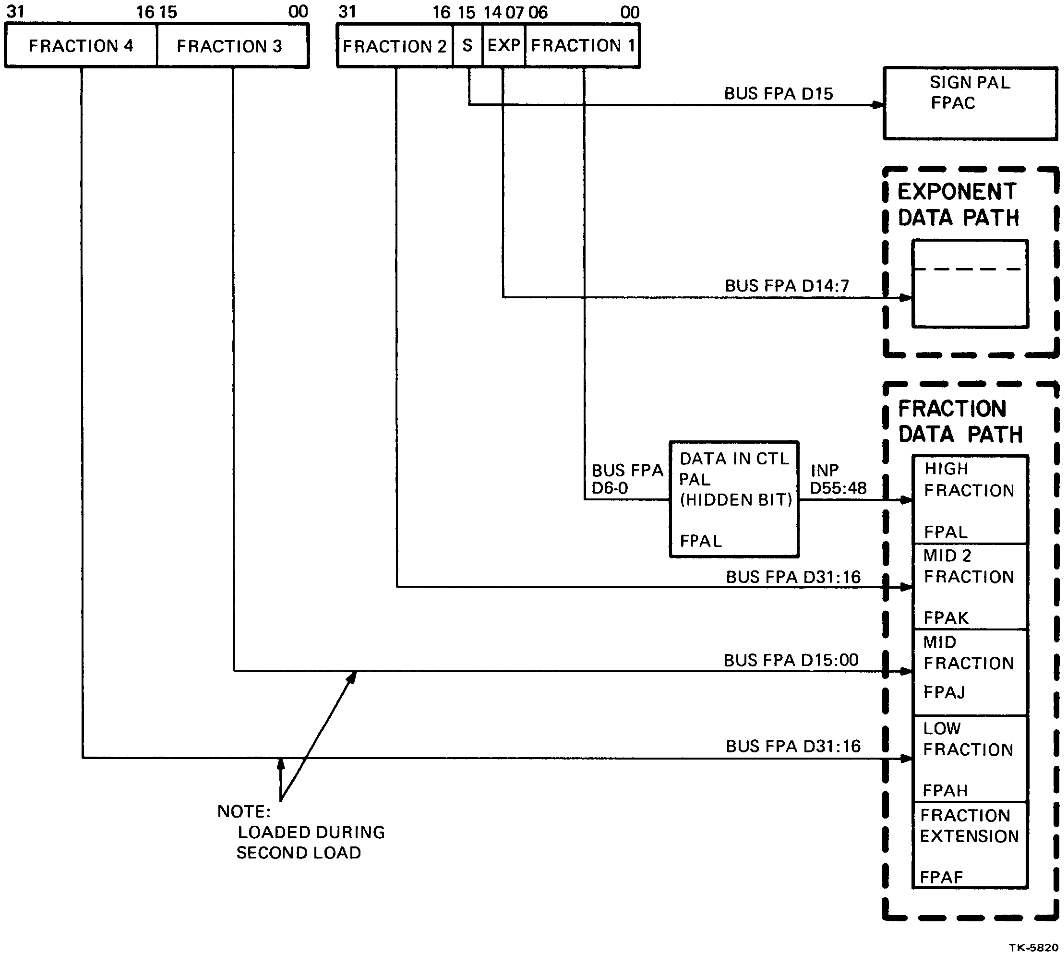


Figure 5-3 Double Format Loading

Only one huge word can be adjusted at a time because both the fraction working register (FWR) and the fraction Q-register (FQ) are needed to shift one huge fraction. The lower half of a huge fraction is initially loaded into FQ and the high half is placed in a temporary FWR. A left-shift is then performed and the MSB of the FQ is directed into the left-shift input for the temporary FWR. The MSB of the FWR is then directed into the EWR. Because of this, seven shifts are required for adjustment of a huge word.

After grand or huge operands are adjusted, OP1 EQ 0 and OP2 EQ 0 flags are set in the branch 3 PAL. For F and D operands this is done automatically as the sign bits are clocked. However, this cannot be done with G and H operands because part of the exponents for these data types is loaded into the fraction data path.

5.2.2 Result Storing

When the CPU finishes passing operands and probing the destination address, it gets ready to accept the condition code (by asserting READ PORT L) and then loops until the FPA asserts ACC SYNC H or an interrupt occurs. If an interrupt occurs the CPU usually aborts the FPA and services the interrupt.

The FPA performs a similar function when storing data. It stores the condition codes and performs a branch that will loop until the CPU asserts READ PORT L. The FPA also asserts ACC SYNC in this word.

The FPA must adjust the results during a store operation. This means shifting out of the hidden bit and performing the required number of shifts for the exponent into the fraction data path. The FPA will also ensure that a data path logic load signal is not asserted.

5.2.3 Aborts

The CPU aborts the FPA for:

1. Interrupts
2. Memory management errors
3. Illegal address mode
4. End of a POLY instruction

The CPU aborts the FPA by forcing microaddress 7 into the FPA control store. This starts a routine that initializes some FPA registers and puts the FPA in a wait loop.

5.2.4 Exceptions or FPA Errors

For the FPA-CPU data flow interface there are error conditions the FPA must indicate to the CPU.

1. Overflow (exception)
2. Underflow (exception)
3. Reserved operand
4. Divide-by-zero
5. Parity error

If any of the error conditions occur, the FPA sets the C-bit in the condition codes, which is the LSB of the FPA output on the Y-Bus. Because the CPU examines this bit first during a result store operation the bit will immediately go to an error handler routine whenever it is set by the FPA. In the CPU the error handler receives a longword error code from the FPA. This error code, in conjunction with the condition codes, is used by the CPU to determine what exception occurred in the FPA. The error codes are constructed by FPA microcode and sent to the CPU. The values of the error codes are listed in Table 5-1.

Table 5-1 Error Codes

Code	Error
0	Overflow if V-bit = 1
0	Underflow if V-bit = 0
7F80	Reserved operand
FF80	Divide-by-zero
X—X1 (LSB=1)	Parity error

After the FPA passes the error code to the CPU via the Y-Bus, it sets up for the next instruction and then goes to a wait loop. However, if a parity error occurs the FPA stays in microword 1, and the CPU must then force the FPA to start again.

5.3 TIMING

The FPA operates with 180 ns and 270 ns cycle times. The fast 180 ns cycle time is the normal FPA cycle time and is used during instruction execution. The slower 270 ns cycle time is used when the FPA is waiting for operands or instructions from the CPU, or when it is storing results to the CPU. Timing logic (Figure 5-4) consists of a clock generator PAL and NAND gates. Figures 5-6 through 5-11 illustrate FPA timing.

The timing logic generates DP0 CLK L, DP1 CLK L, and REG CLK L which are applied to control store, data path logic, branch logic, and control logic. Although these clocks are produced by three separate NAND gates (for loading purposes), they are generated identically. The timing logic also generates IR CLK L and IR CLK H which are applied to instruction decoding logic. A 45 ns TRISTATE DISA H output, which occurs at the start of every timing cycle, disables FPA transceivers to prevent them from being simultaneously enabled.

In the timing logic (Figure 5-4) the clock generator PAL generates either SLOW PATH ENAB H or FAST PATH ENAB H, plus FP PH1 and CPU PH0 H (Figures 5-5 and 5-6). These are applied to gates used for selection of a 270/180 ns cycle time. Clock PAL inputs ENB CLK (1) H and BASIC CLOCK H (memory controller PORT CLOCK L) inputs are used to generate DP1 CLK L, DP0 CLK L, and REG CLK L. BASIC CLOCK H is also used for generation of IR CLK H and IR CLK L.

When FAST CYCLE L is not asserted the slow path is enabled. During slow path operation the clock generator PAL generates SLOW PATH ENB, and the CPU P2 H clock (Figure 5-7) controls when the FPA clocks are asserted (Figure 5-8).

Figure 5-8 illustrates fast/slow cycle gating. During normal fast path gating (Figure 5-9) in the timing logic, when TRAP ACC or READ ACC UPC are not asserted by the CPU, FP PH1 and FAST PATH ENAB H are used to generate CLK ENB H.

If the CPU asserts TRAP ACC L or READ ACC UPC L, and the CPU is operating in PH1, FP PH1 H and FAST PATH ENB H from the clock PAL are used to generate CLK ENB H (Figure 5-8).

When the CPU asserts READ ACC UPC L, the clock generator PAL generates CLOCK OFF that disables the fast and slow path gates. This prevents the FPA registers from being clocked.

Also, a fast signal (internal to the clock generator PAL) is cleared when the CPU asserts TRAP ACC L. This ensures that the FPA clocks will be restarted in synchronization with the CPU.

The READ ACC UPC L input to the timing logic also causes BUS NUA from the microsequencer to be sent to the CPU when CPU RCV DATA L is asserted.

When the CPU asserts FORCE UADDR L, the FAST CYCLE signal (internal to the clock generator PAL) is reset, and the FPA fast cycle is stretched (as required) so that, at the end of the current cycle, the FPA will be in synchronization with the CPU.

Figures 5-10 and 5-11 illustrate slow path timing with the FPA synchronized with the CPU. This can occur when the FPA slows its clocks (via microcode function) or when the CPU asserts TRAP ACC L or READ ACC UPC L. Either signal will slow the FPA clocks until they are synchronized with the CPU.

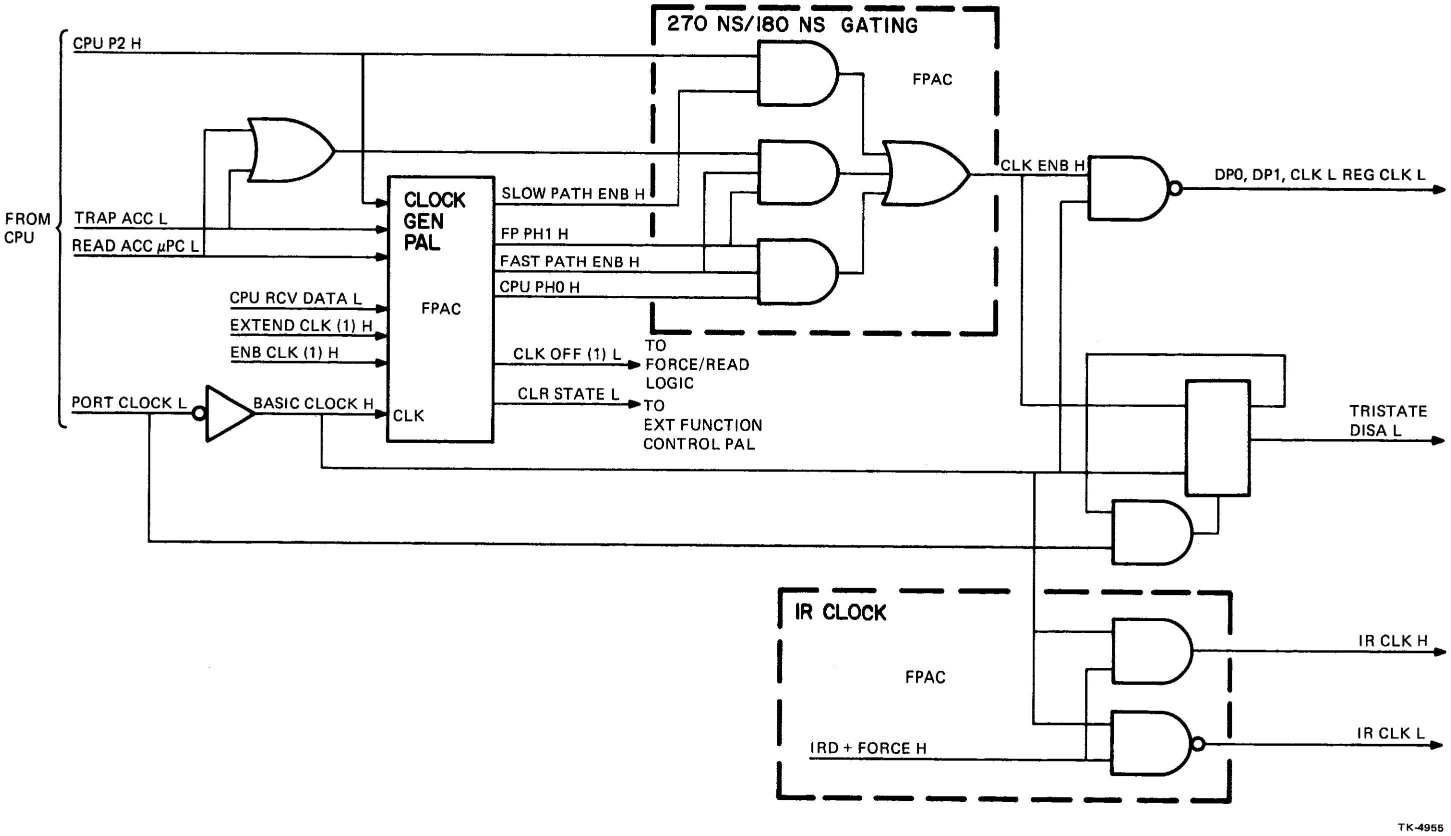


Figure 5-4 Timing Logic

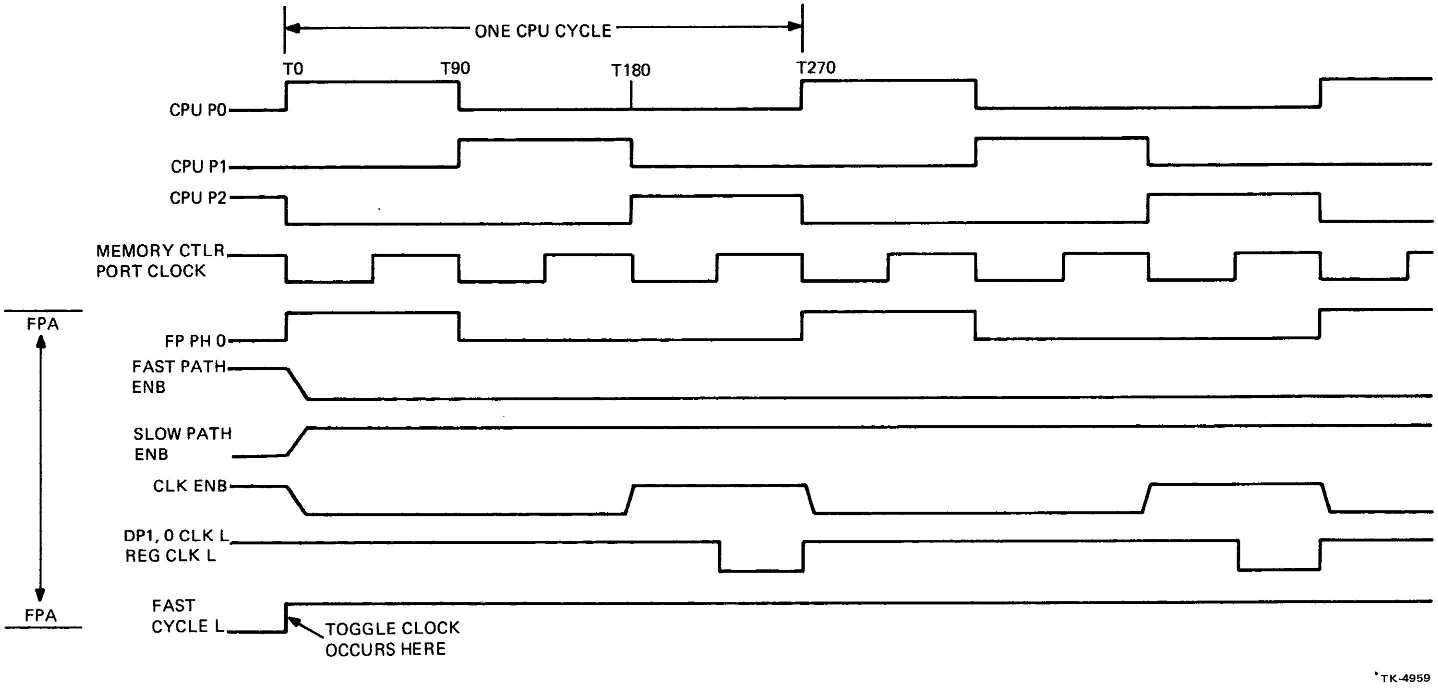


Figure 5-5 FPA Synchronization via Toggle Clock During CPU PH0

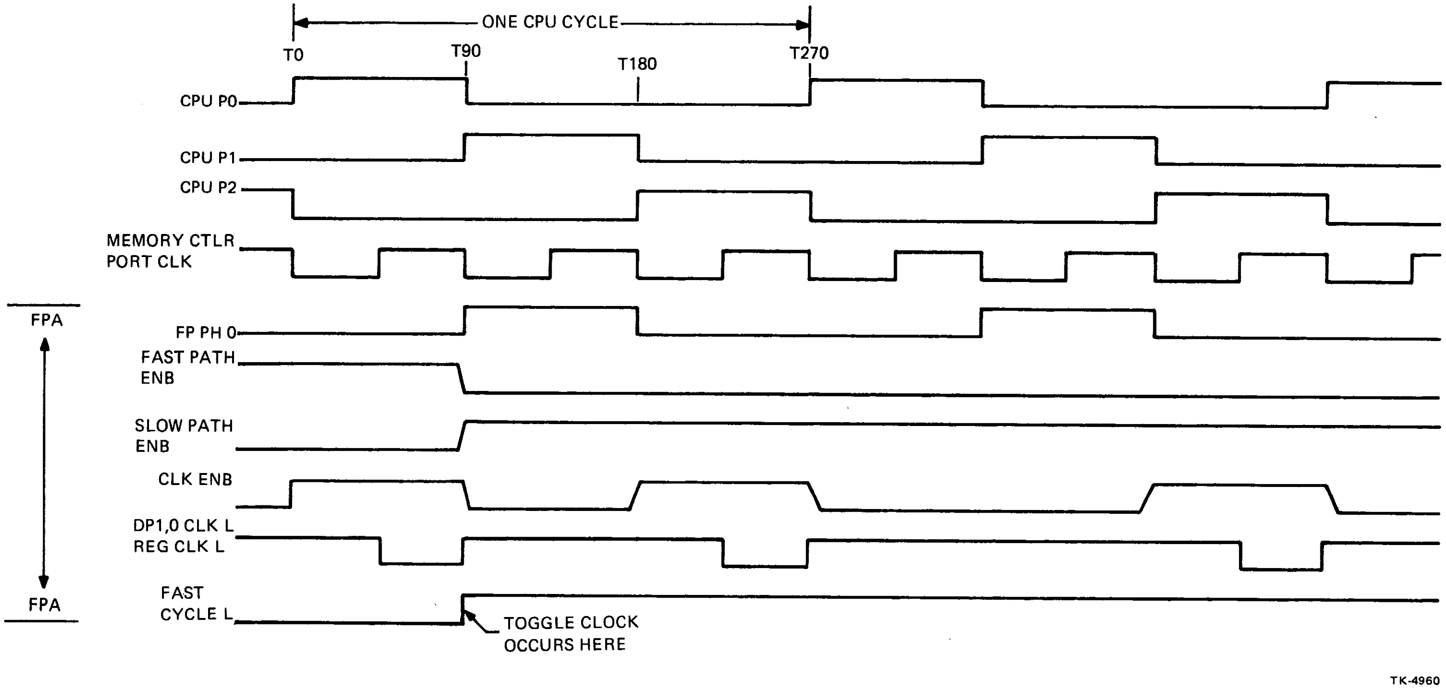


Figure 5-6 FPA Synchronization via Toggle Clock During CPU PH1

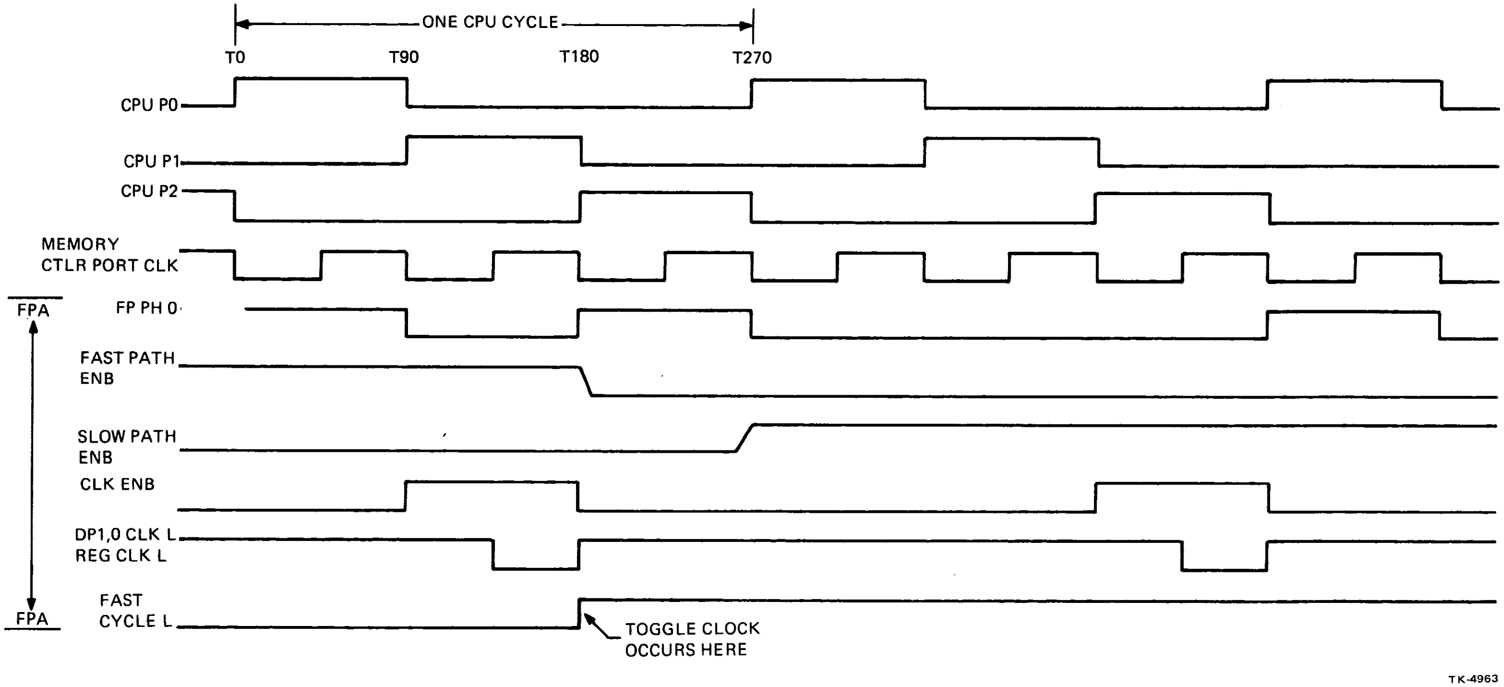


Figure 5-7 FPA Synchronization via Toggle Clock During CPU PH2

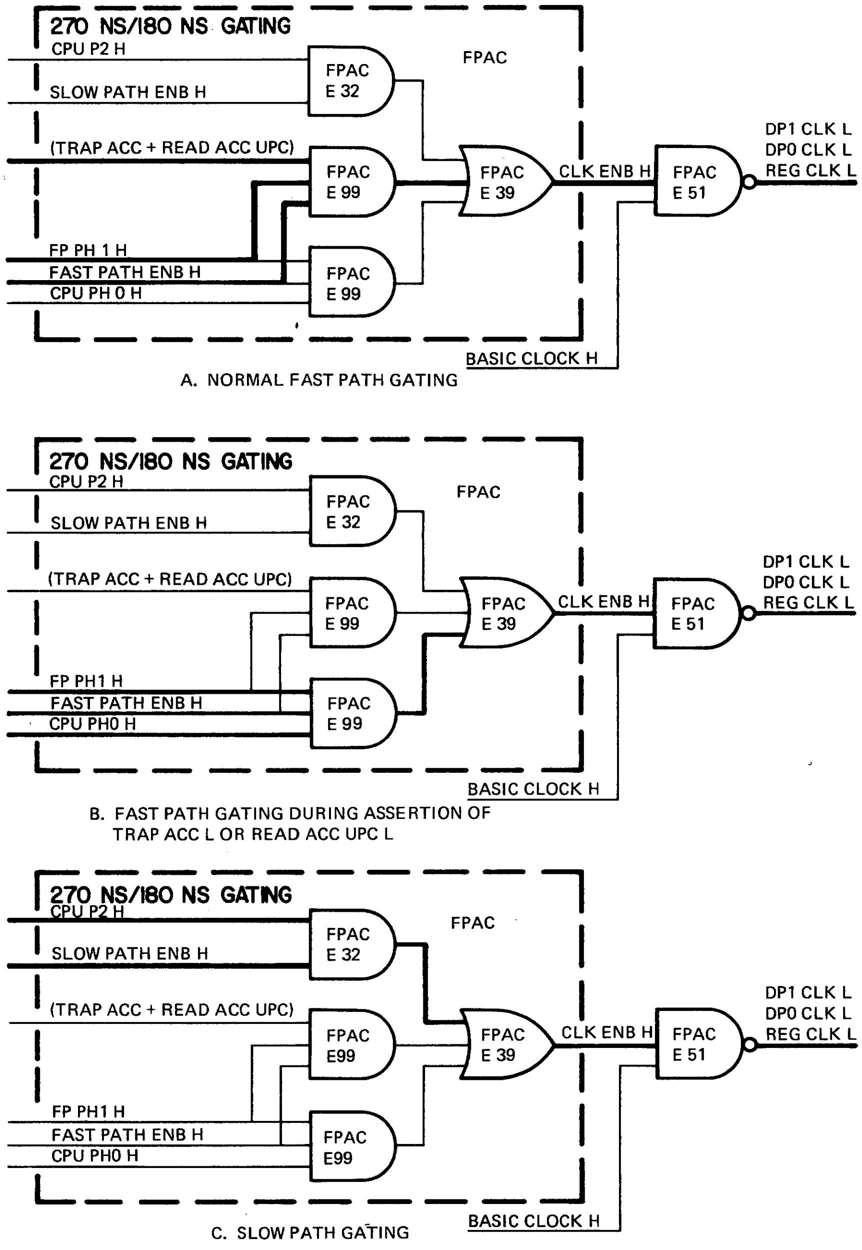


Figure 5-8 Fast/Slow Cycle Gating

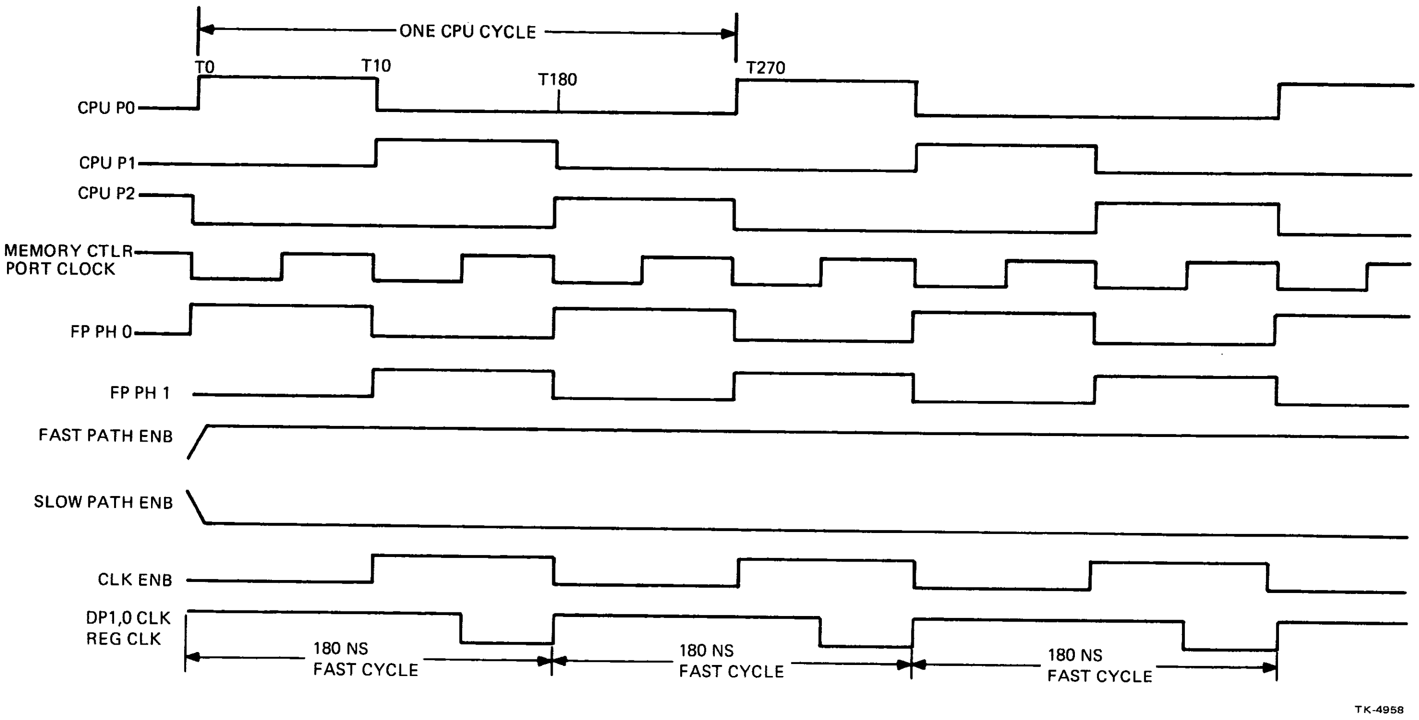


Figure 5-9 Fast Cycle Timing

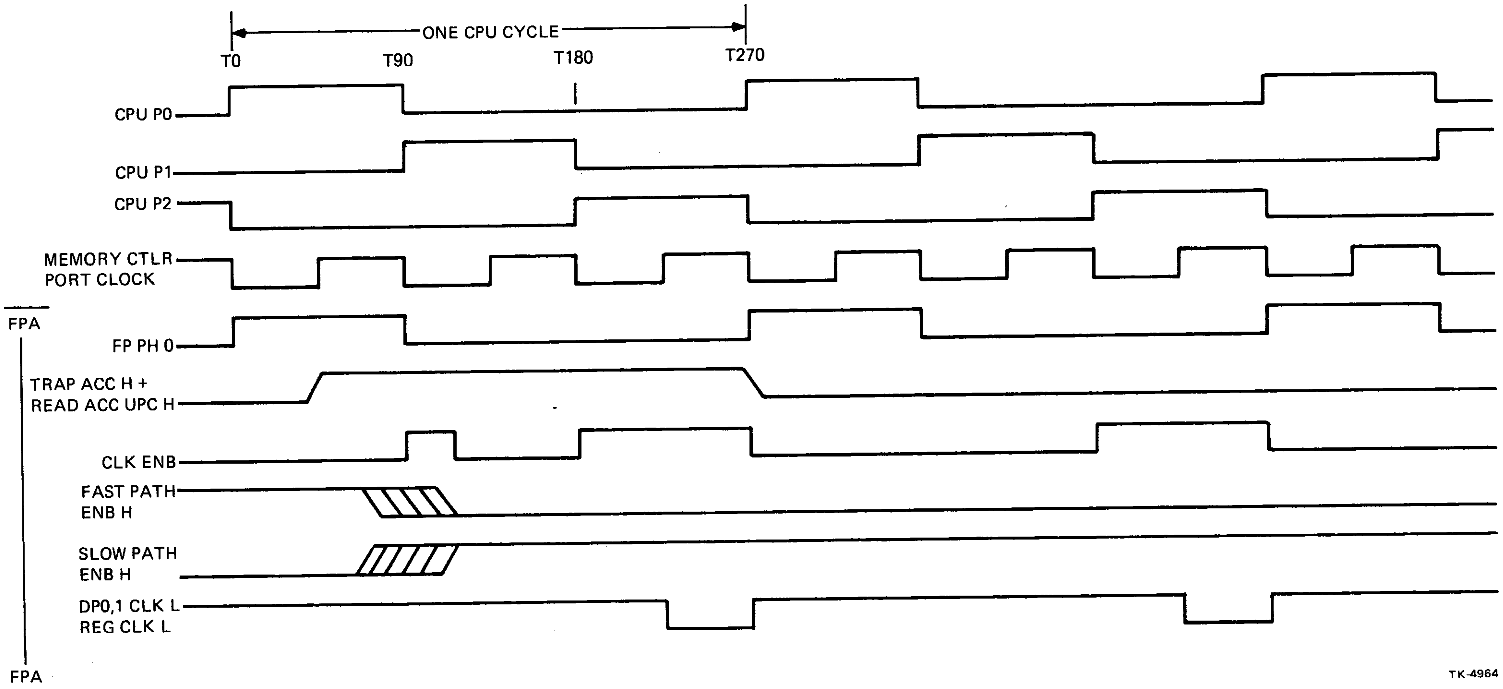


Figure 5-10 FPA Synchronization via CPU Force Trap or Read During FPA PH0

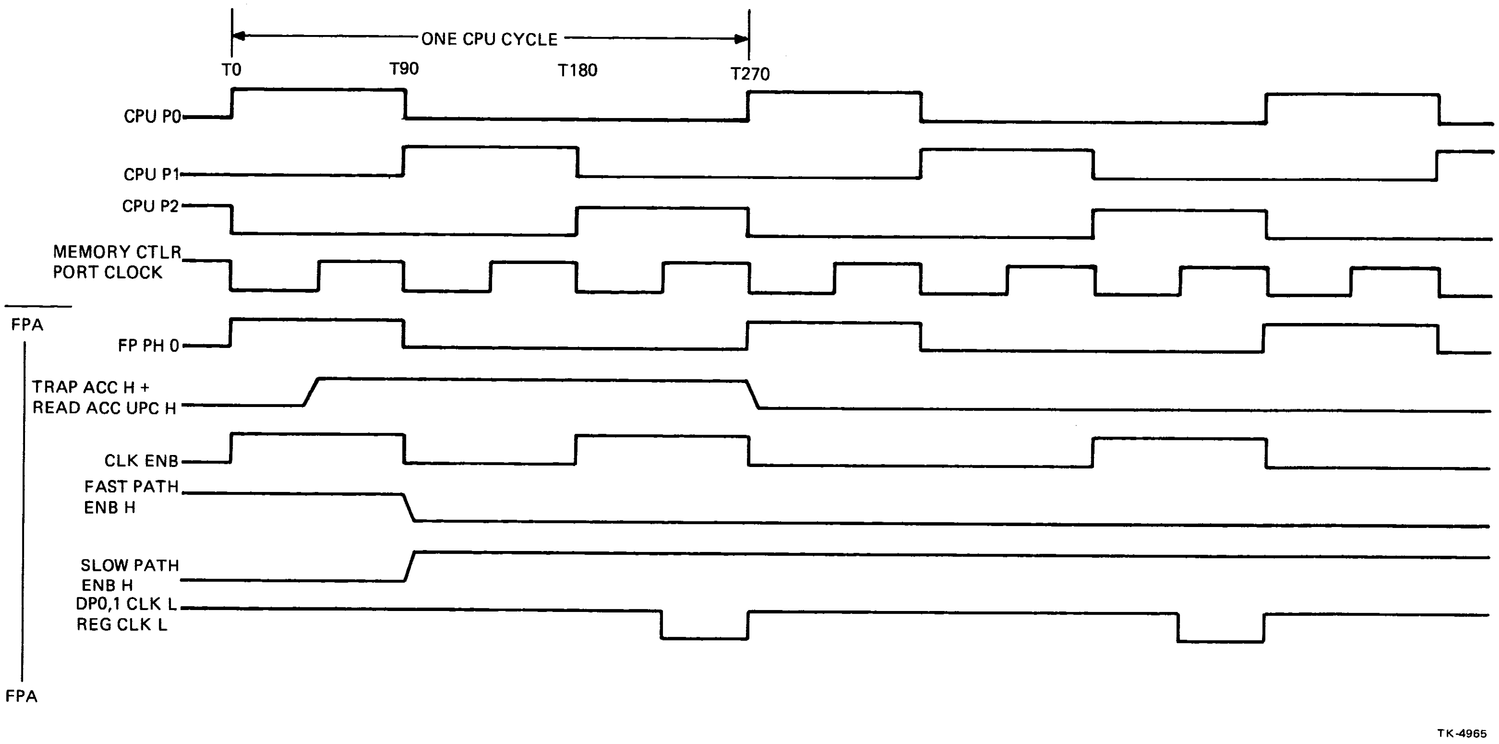


Figure 5-11 FPA Synchronization via CPU Force Trap or Read During FPA PH1

5.4 INSTRUCTION DECODING

The FPA instruction decoding logic (Figure 5-12) decodes a floating-point instruction (received on the IB-Bus) into: 1) a 5-bit starting offset address for the microsequencer logic and 2) a 2-bit data size code (SIZE 1:0 H). The data size code indicates to the control logic the data type size (F, D, G or H) that will be received from the CPU via the IB-Bus, and also causes the FPA to be set up to process data type operands. Instruction decoding is performed via a ROM, an extended function control, and a multiplexer.

At the start of a floating-point routine, an operation code (BUS IB D7:0 H) is applied, as the address to a 512 × 8 ROM (Figure 5-12). The ROM output is DECODE ROM 7:0 H and causes the microsequencer to generate a microaddress (BUS NUA 9:0 H) for control store. This is the starting address of the FPA routine in the control store (see Table 5-1 and Figure 5-13). At the ROM output, DECODE ROM 6:0 is applied to a multiplex latch that is controlled by IRD STATE L and IR CLK H. The latch outputs are INSTR ENC 4:0 H and SIZE 1:0 H.

At the latch output the SIZE 1:0 H lines are decoded with the data type (F, D, G, or H) that will be received from the CPU via the Y-Bus. Table 5-2 explains SIZE field decoding.

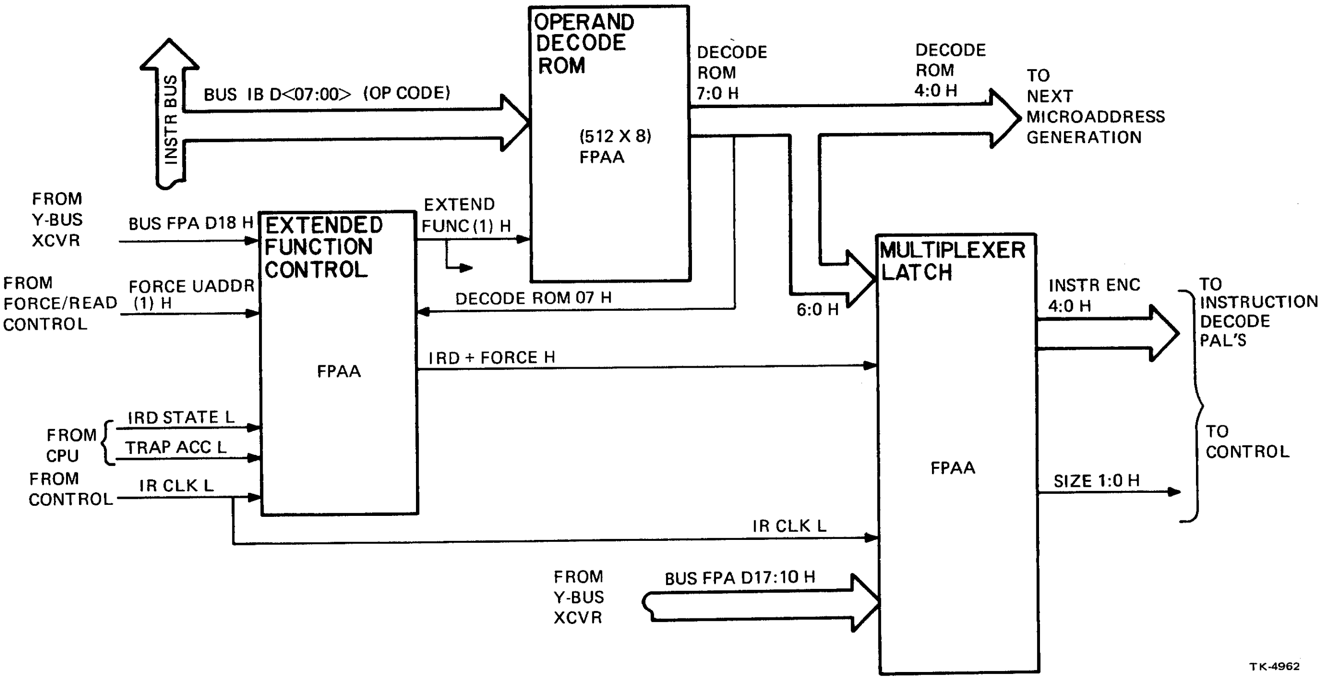
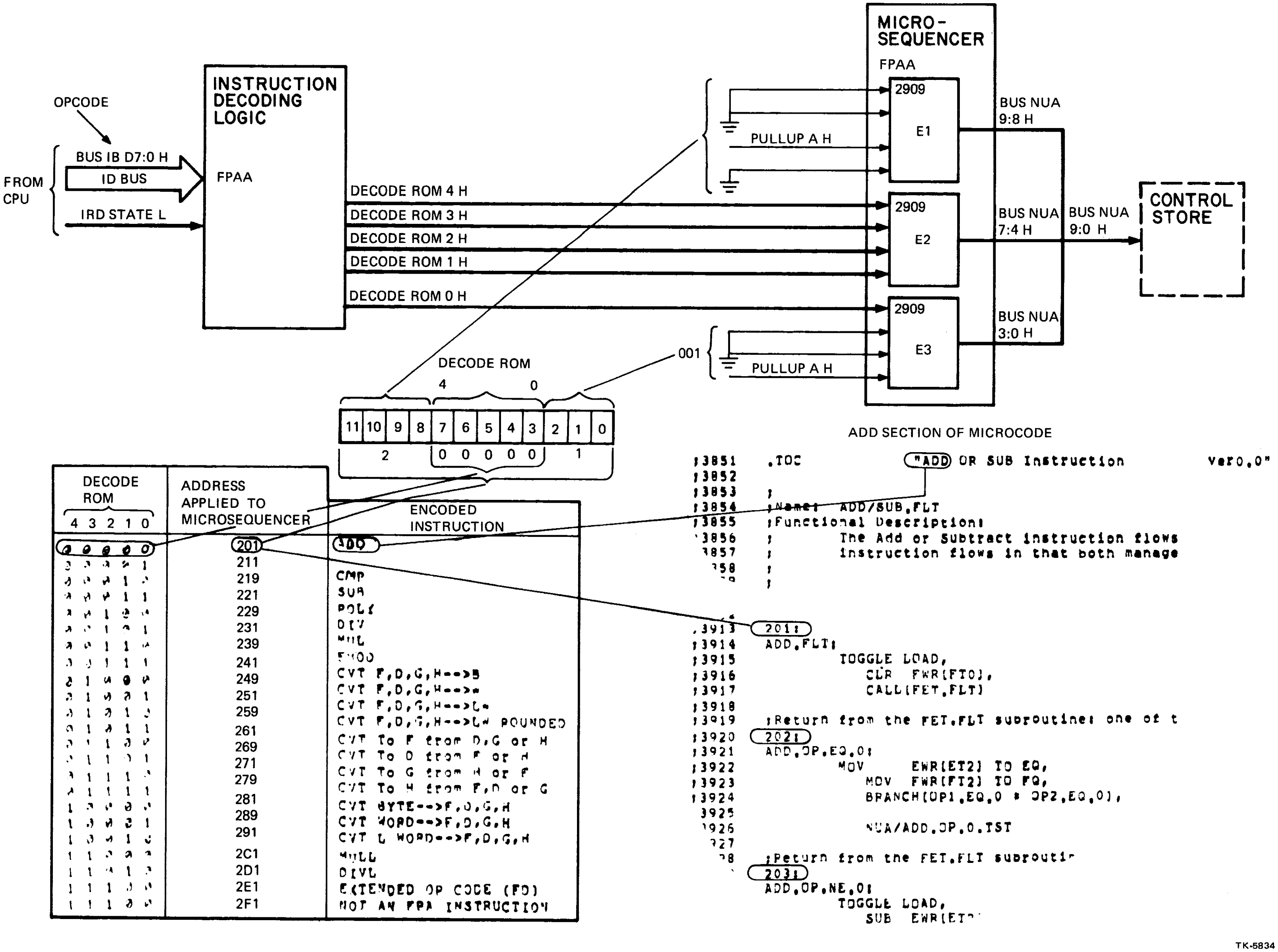


Figure 5-12 Instruction Decoding

Table 5-2 SIZE 1:0 Encoding

SIZE 1:0 H Value	Data Type Indicated
0	F (Single-precision)
1	D (Double-precision)
2	G (Grand)
3	H (Huge)



DECODE ROM 4 3 2 1 0	ADDRESS APPLIED TO MICROSEQUENCER	ENCODED INSTRUCTION
0 0 0 0 0	201	ADD
0 0 0 0 1	211	
0 0 0 1 0	219	CMP
0 0 0 1 1	221	SUB
0 0 1 0 0	229	POLY
0 0 1 0 1	231	DIV
0 0 1 1 0	239	MUL
0 0 1 1 1	241	EMOD
0 1 0 0 0	249	CVT F,D,G,H-->B
0 1 0 0 1	251	CVT F,D,G,H-->W
0 1 0 1 0	259	CVT F,D,G,H-->LW
0 1 0 1 1	261	CVT F,D,G,H-->LW ROUNDED
0 1 1 0 0	269	CVT To F from D,G or H
0 1 1 0 1	271	CVT To D from F or H
0 1 1 1 0	279	CVT To G from H or F
0 1 1 1 1	281	CVT To H from F,D or G
1 0 0 0 0	289	CVT BYTE-->F,D,G,H
1 0 0 0 1	291	CVT WORD-->F,D,G,H
1 0 0 1 0	2C1	CVT L WORD-->F,D,G,H
1 1 0 0 0	2D1	MULL
1 1 0 1 0	2E1	DIVL
1 1 1 0 0	2F1	EXTENDED OP CODE (FD)
1 1 1 0 0		NOT AN FPA INSTRUCTION

```
        .TOC            "ADD OR SUB Instruction          Ver0.0"

        ;
        ;Name:  ADD/SUB.FLT
        ;Functional Description:
        ;           The Add or Subtract instruction flows
        ;           instruction flows in that both manage
        ;
        ;

        201:
        ADD.FLT:
                    TOGGLE LOAD,
                       CLR  FWR[FT0],
                       CALL[FET.FLT]

        ;Return from the FET.FLT subroutine: one of t
        202:
        ADD.OP.EQ.0:
                    MOV       EWR[ET2] TO EQ,
                       MOV  FWR[FT2] TO FQ,
                       BRANCH[OP1.EQ.0 * OP2.EQ.0],

                       NUA/ADD.OP.0.TST

        ;Return from the FET.FLT subrouti
        203:
        ADD.OP.NE.0:
                    TOGGLE LOAD,
                       SUB  EWR[ET
```

Figure 5-13 Op Code Instruction Decoding

Figure 5-14 illustrates the latch signal inputs during normal and diagnostic checks operation. During microdiagnostic operation the CPU causes BUS FPA D17:10 to clock through the instruction decoding circuit multiplexer to check its operation. Clocking is enabled by BUS FPA D18 H and TRAP ACC L, which causes IRD + FORCE H to be ANDed in the FPA clock generator with CPU P2 H to produce IR CLK H. If IRD STATE L is not asserted, it then selects BUS FPA D17:10 to be loaded into the instruction register. BUS FPA D17:10 then causes INSTR ENC 4:0 H and size ⟨1:0⟩ to be output from the instruction register.

The EXTENDED FUNC (1) H output of the extended function control is asserted when the operation code on the IB-Bus indicates an extended op code is on the IB-Bus. This is applied to the decode ROM and alters the ROM address during the next instruction decode state.

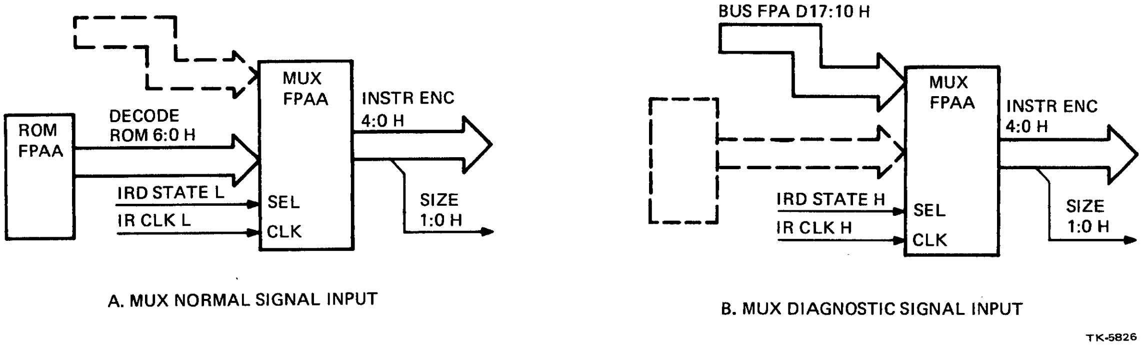


Figure 5-14 Instruction Decoding MUX Signal Inputs

5.5 NEXT MICROADDRESS GENERATION

The FPA microsequencer logic (Figure 5-15) generates a sequence of 10-bit microaddress outputs (as BUS NUA 9:0 H) that are applied to the control store. They cause the control store to generate data path logic setup control signals for operand processing.

The microsequencer logic (Figure 5-15) consists of three 2909 4-bit microprogram sequencers, plus control circuitry. Although the three 2909 chips could generate 12 output bits, they are configured in the FPA to generate only a 10-bit output. This is all that is required to access the control words contained in the control store.

The microsequencer has two data inputs. One is a direct input driven at the start of an FPA operation by DECODE ROM 4:0 H from the instruction decoding logic. The other input is a register input that is driven by a 10-bit micropointer field (CS9:0 H) from the control store. This input can be branched upon.

The three 2909 microprogram sequencers (Figure 5-16) contain a four-input multiplexer that is used to select:

1. an address register
2. the direct inputs
3. a microprogram counter
4. a stack file

as the source for the next microinstruction base address. The selection is done via encoding on two output lines of address select logic (Figure 5-15). The encoding is controlled via a UBCTL 4:2 (1) H input from the FPA in the FPA branch logic.

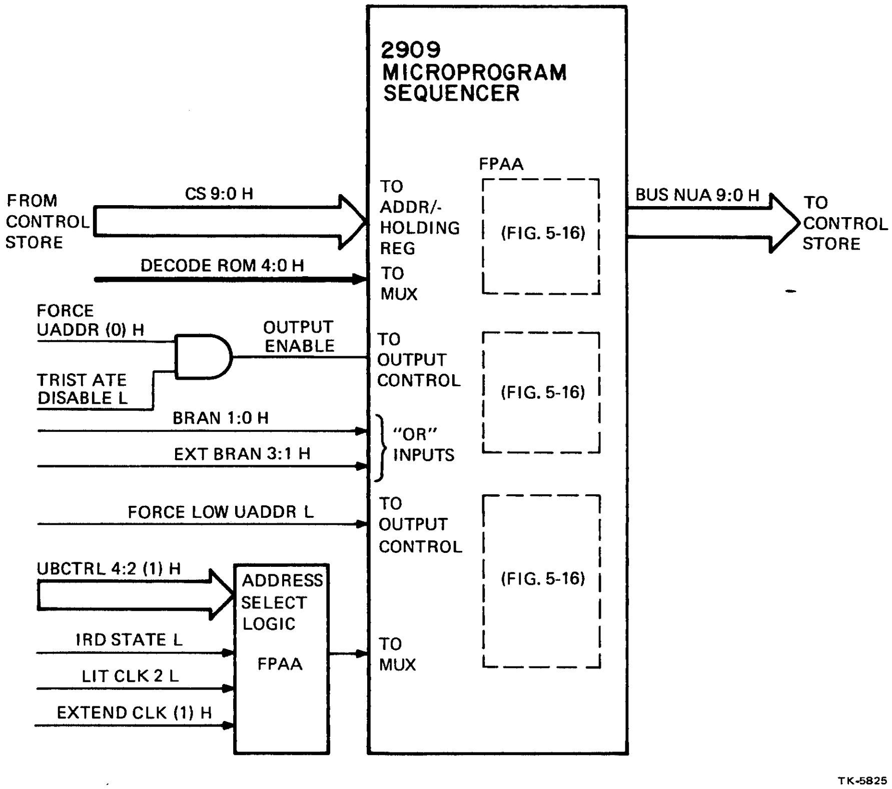


Figure 5-15 Microsequencer Logic

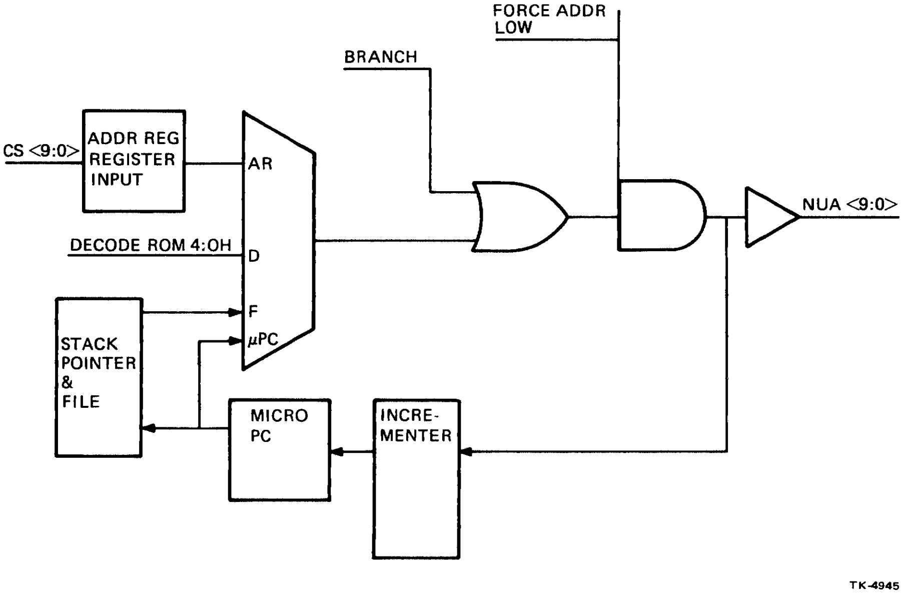


Figure 5-16 2909 Microprogram Sequencer

The 2909 address register consists of four D-type, edge-triggered flip-flops enabled by DPO CLK L from the FPA timing logic. Because the register (REG EN) lines are hard-wired to logic ground (Figure 5-13), new data is entered into the register on the low-to-high transition of DPO CLK. The address register output is available at the multiplexer in the 2909 as a source for the next microinstruction address (microaddress NUA 9-0 H).

The direct input to the multiplexer is driven by DECODE ROM 4-0 H from the instruction decoding logic. This input is used for the next microaddress in the IRD state.

The CN input to the 2909s causes the microprogram register in the 2909s to sequentially increment on the next DPO CLK cycle with the current NUA 9-0 H output, plus 1.

The stack (file) content can also be used as the source for the next microaddress. The stack is used to provide return address linkage when executing microsubroutines. The stack contains a built-in pointer (SP) that always points to the last file word written. This allows stack reference operations (looping) to be performed without a push or pop.

The SP operates as an up/down counter with separate PUSH and FILE ENB inputs. When the FILE ENB input is low and the PUSH input to the 2909s is high, a push operation is enabled. This causes the stack pointer to increment and the file to be written with the micro-PC, which contains the address of the current microinstruction, plus 1.

If the FILE ENB input to the 2909s is low and PUSH control is low, a stack pop operation occurs. This implies the usage of the return linkage during this cycle and thus a return from the subroutine. The return address is the calling address, plus 1. The next low-to-high DPO CLK transition will cause the SP to be decremented. If FILE ENB is high, no action is taken by the SP regardless of any other input.

The stack pointer linkage is such that any combination of pushes, pops or stack references can be achieved. Only microinstruction subroutines can be performed. Since the stack is 4 words deep, up to four microsubroutines can be nested.

The FORCE ZERO input applied to the 2909 microproogram sequencers is used to force the 10 BUS NUA 9:0 H outputs of the sequencer to zero. When FORCE LOW UADDR L is asserted in the force/read logic, all 10 outputs are low regardless of any other inputs (except OUTPUT ENABLE). Each BUS NUA output bus also has [at the 2909 tristate output (Y3-)] separate OR logic that permits a logical 1 to be forced at each BUS NUA 9:0 output. This allows branching to different microinstructions on programmed conditions.

5.6 NEXT MICROADDRESS BRANCHING

Branching is performed on status signals from the data path logic and instruction signals. The signals cause either BUS NUA 1:0 H or BUS NUA 4:0 H at the microsequencer output to be affected. The branch logic consists of a status register and five PALs. Four of the PALs are used for normal branching on the two low NUA bits, and all of the PALs are used during extended branching.

Status signals from the data path logic are applied to the status register. They are clocked by DPO CLK L, and then appear as inputs for the branching PALs. The PALs are controlled via UBCTL 4:0 (1) H from the control store. This field selects which status bit or combination of bits, will be directed onto the BRANCH 1:0 H output lines of the PALs. Table 5-3 lists signals selected by the branch control field.

Extended branching affects NUA 4:2 of the microsequencer output. This branching is sometimes used for wide branches, and is selected by the CLK CTL and MOD fields in the control store. Of UBCTL branch control bits 4:2, the upper two bits (4:3) determine what type of extended branch is to be taken. Table 5-4 lists the extended branches.

Table 5-3 Branch 1:0 Encoding

UBCTL 4:0 (1) H Value (Hex)	**Branch PALs Output Lines**		
	BRAN1	**BRAN0**	**Special Conditions**
0	EXP COUT	GRAND	
1	SIGN OUT	HUGE	
2	CPU DATA AVAIL	SINGLE	ASSERT OPTION SYNC
3	CPU DATA AVAIL	ADD + SUB	ASSERT OPTION SYNC
4	FRAC COUT	EXT FUNC	
5	OP1 SIGN	EMOD	
6	FRAC55 F3	SINGLE	
7	OP2 SIGN	ADD + SUB	
8	EXP COUT	EXP15 F3	
9	SIGN OUT	OP2=0	
A	CPU DATA AVAIL	ZERO	
B	CPU DATA AVAIL	ZERO	
C	OP2 SIGN	(OP1 + OP2)/=0	
D	OP1 SIGN	(OP1 + OP2)/=0	
E	FRAC55 F3	0	
F	FRAC COUT	EXP15 F3	
10	MUL I1	FRAC55 Q3	
11	F47.F3	EXT00 Q0	
12	FRAC⟨55:00⟩=0	DIV 13	
13	FRAC⟨47:16⟩=0	ZERO	
14	FRAC⟨55:00⟩=0	CPU RCV DATA	
15	ZERO	ZERO	NULL BRANCH
16	ZERO	CPU RCV DATA	OPTION SYNC
17	FRAC⟨55:7⟩=0	ZERO	
18	EXPONENT=0	EXP15 F3	
19	OP1=0	OP2=0	
1A	ZERO	ZERO	CALL SUBROUTINE
1B	SUMPATH	ZERO	
1C	ZERO	(OP1 + OP2)/=0	RETURN FROM SUBROUTINE
1D	ZERO	(OP1 + OP2)/=0	RETURN FROM SUBROUTINE
1E	ZERO	ZERO	RETURN FROM SUBROUTINE
1F	ZERO	EXP15 F3	RETURN FROM SUBROUTINE

Table 5-4 Extended Branching

UBCTL 4:3 (1) H Value	Extend Branch Bits		
	BRAN4	BRAN3	BRAN2
0	DOUB OPER	ADD + SUB	FRAC31-EXT00=0
1	SIZE1	SIZE0	FRAC⟨31:0⟩=0
2	DOUB OPER	ADD + SUB	ZERO
3	INSTR ENC2	INSTR ENC1	INSTR ENC0

5.7 CONTROL STORE

During floating-point calculations a sequence of microinstructions (data control signals) is accessed from control store (Figure 5-17) and applied to the data path logic. After operands from the Y-Bus are loaded into the data path logic, the latter then operates on the data input in accordance with the commands it receives from the control store. The FPA control store consists of a PROM and several registers.

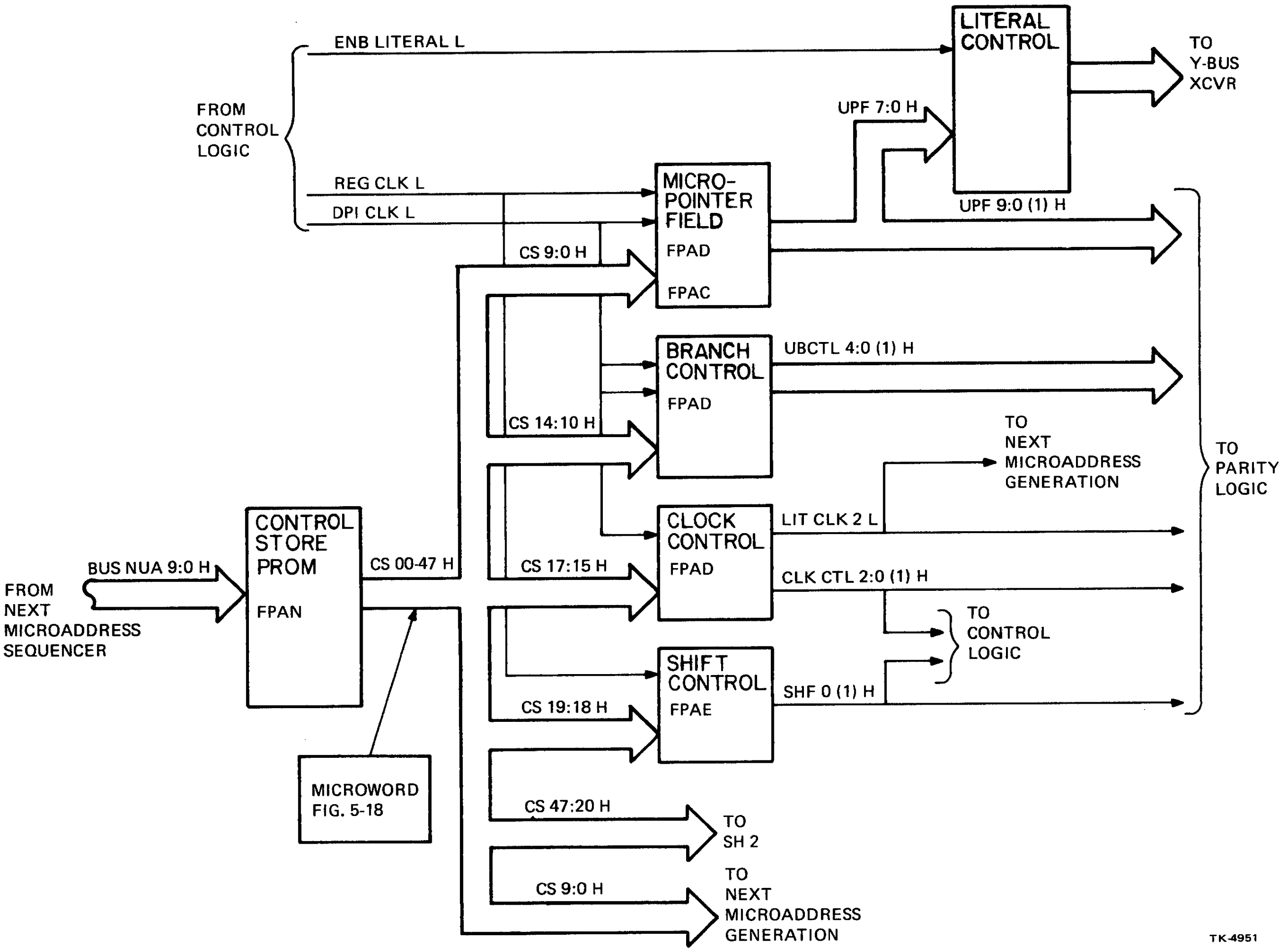


Figure 5-17 Control Store Logic (Sheet 1 of 2)

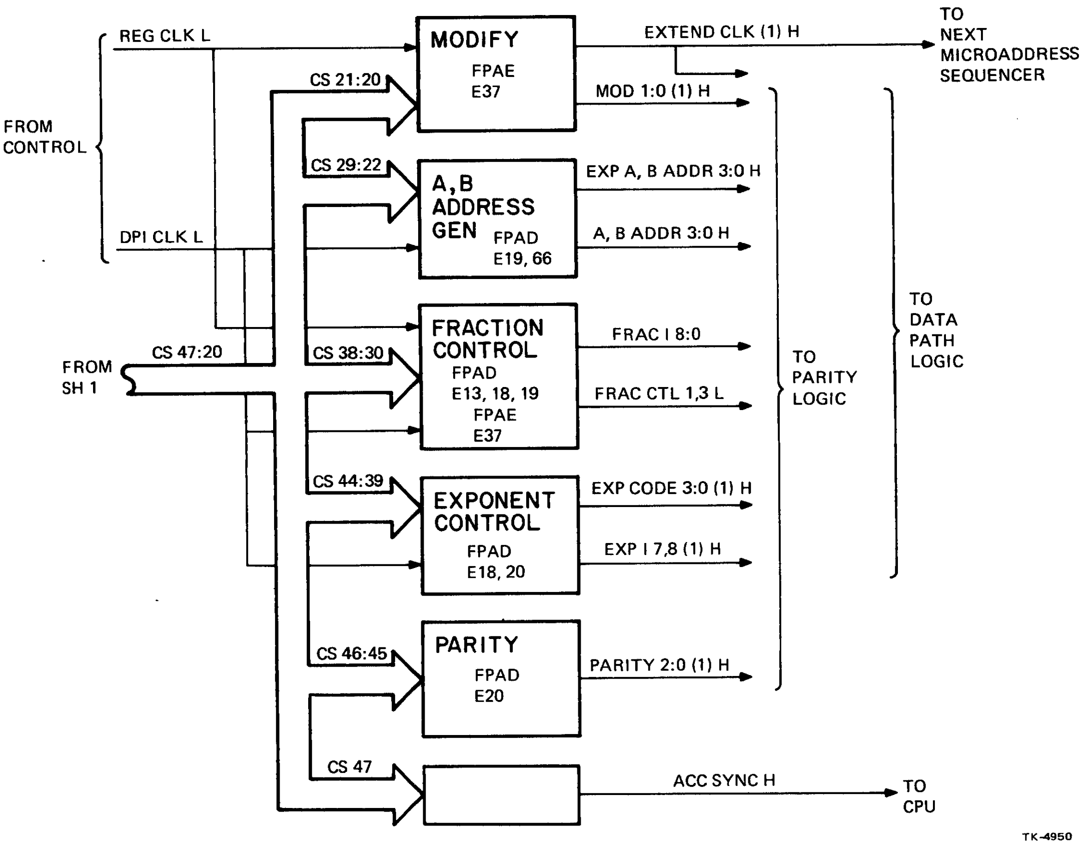


Figure 5-17 Control Store Logic (Sheet 2 of 2)

The control store PROM contains 1K 48-bit microwords. Each of the microwords contains a 2-bit parity field. When the control store PROM is addressed with BUS NUA 9:0 H from the microsequencer, the total 48-bit microword PROM output is applied to control store registers. These registers then generate data path logic control signals, plus a micropointer field that is applied to the microsequencer. Figure 5-18 illustrates the microword accessed from the PROM. Table 5-5 explains the fields in the microword.

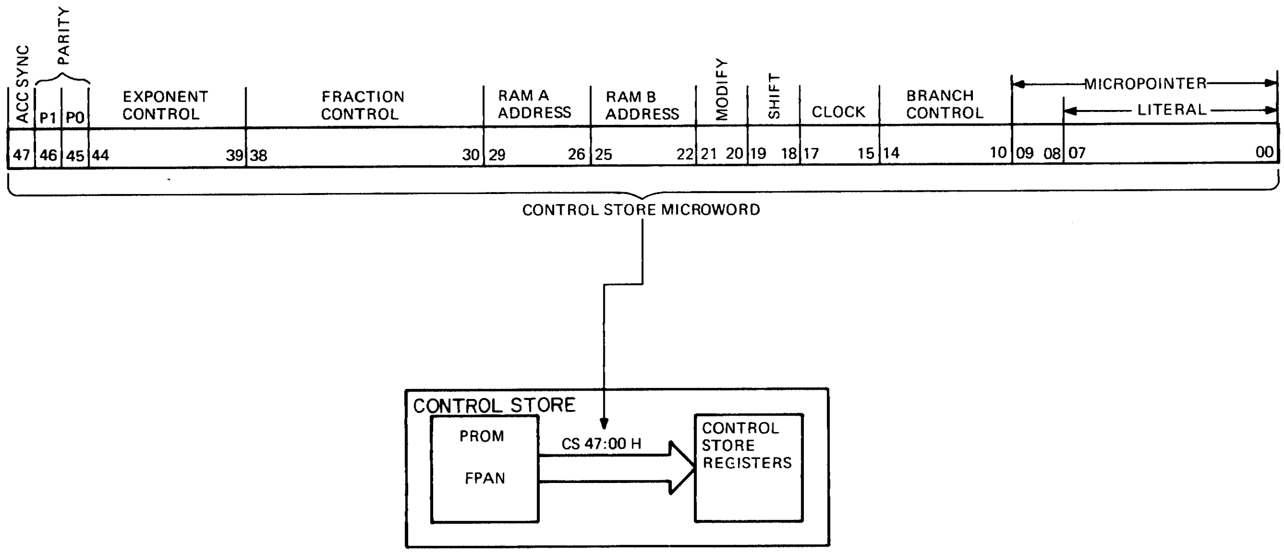


Figure 5-18 Control Store Microword

Table 5-5 Control Store Field

<table>
<tr><th>CS</th><th>Function</th><th>Description</th></tr>
<tr><td>47</td><td>ACC SYNC</td><td>Option synchronization signal</td></tr>
<tr><td>46</td><td>Parity bit P1</td><td>Parity bit for checking CS<14:13>, CS<36:30>, CS<39>, CS<44:43> and CS<12:10>.</td></tr>
<tr><td>45</td><td>Parity bit 0</td><td>Parity for checking CS<8:0>, CS<17:15>, CS<21:18> and CS<39:37>.</td></tr>
<tr><td>44:43</td><td>Exponent destination control field (EXP DST)</td><td>Controls the destination of the ALU output. Normally, the ALU's output can be clocked into either the working register (WR) or Q-register.
<table>
<tr><th>EXP DST<1:0></th><th>Destination</th></tr>
<tr><td>00</td><td>Q-register</td></tr>
<tr><td>01</td><td>Working register (WR)</td></tr>
<tr><td>10</td><td>Right-shift and write the WR</td></tr>
<tr><td>11</td><td>Left-shift and write the WR</td></tr>
</table>
</td></tr>
<tr><td>42:39</td><td>Exponent data path control (EXP CTL)</td><td>This field encodes the 2901 ALU functions for both the source and destination. Most of the functions can be clocked into the working register (WR) or Q-register, depending on the exponent destination code. The functions marked with an asterisk (*) can be clocked into the working register (WR) only.
<table>
<tr><th>EXP CTL<3:0></th><th>Function</th><th>EXP CTL<3:0></th><th>Function</th></tr>
<tr><td>0000</td><td>D or 0</td><td>1000</td><td>Q – 1</td></tr>
<tr><td>0001</td><td>B – A</td><td>1001</td><td>Q + 1</td></tr>
<tr><td>0010</td><td>A – B</td><td>1010</td><td>A</td></tr>
<tr><td>0011</td><td>B + A</td><td>1011</td><td>Q</td></tr>
<tr><td>0100</td><td>A OR B</td><td>1100</td><td>0</td></tr>
<tr><td>0101</td><td>A AND B</td><td>1101</td><td>SHIFT</td></tr>
<tr><td>0110</td><td>A – Q</td><td>1110</td><td>A + 8 + 1</td></tr>
<tr><td>0111</td><td>A + B + FRAC COUT</td><td>1111</td><td>NOOP</td></tr>
</table>
</td></tr>
</table>

Table 5-5 Control Store Field (Cont)

CS	Function	Description
38:30	Fraction data path control (FBAC CTL)	This field directly corresponds with the 2901 signals I11:8.
29:26	A address field (A ADDR)	This field addresses the A port of the 2901's working register (WR) from both the exponent and fraction data path. If the clock field equals clock sign out, then the lower 3 bits of the A address control which function the sign out flip-flop is clocked with. **A ADDR<2:0>** — **SIGN OUT Gets:** 000 — OP1 SIGN 001 — OP2 SIGN 010 — OP1 SIGN XOR OP2 SIGN 011 — OP1 SIGN XOR SIGN OUT 100 — ZERO 101 — ONE 110 — ZERO 111 — ONE
25:22	B address field (B ADDR)	This field addresses the B port of the 2901's WR for both the exponent and fraction data path. This is the write back address.
21:20	Modification field (MOD)	This field extends the use of other fields, as well as enabling special functions. 1. MOD<1:0>=00 — Noop 2. MOD<1:0>=01 — Extend clock field 3. MOD<1:0>=10 — Enable MUL/DIV 4. MOD<1:0>=11 — Enable load or store The clock extend function doubles the functions that can be performed by the clock field. The enable MUL/DIV mod field enables some conditional logic for multiple and divide. The op code control determines what is actually enabled.

Table 5-5 Control Store Field (Cont)

CS	Function	Description
		The enable load or store field makes it possible to load or store sections of the fraction and exponent data path. Whether a store or load is performed is determined by the load signal which is set by a clock code. The actual section to be loaded or stored is determined by the shift field.
19:18	Shift field (SHF)	This field has many different functions, depending on the operation being executed.
		LOAD The SHF field determines what section is loaded.
		1. SHF=00 First floating Load: SIGN EXP<7:0> FRAC<55:32>
		2. SHF=01 Mod load: EXT<7:0>
		3. SHF=10 Second floating load or integer load or integer load FRAC<31:16> or FRAC<55:00> depending on whether or not an integer is being loaded. If an integer is being loaded the lower 16 bits must be masked out by the microcode.
		4. SHF=11 Third huge load: EXT<7:0> FRAC<55:32>
		STORE
		1. SHF=00 First word store: SIGN#EXP<7:0>=FRAC<55:32>
		2. SHF=01 Condition code store
		3. SHF=10 Second word store: FRAC<31:00>
		4. SHF=11 Huge store: EXT<7:0>#FRAC<55:32>
		SHIFTS - The shift field also determines what is shifted into the exponent Q0 and R0, FRAC55 Q3 and R3 and EXT00 Q0 and R0.

Table 5-5 Control Store Field (Cont)

CS	Function	Description
		Right-Shift - The shift field controls what is shifted into the MSB of the fraction data path.

SHF<1:0>	FRAC55 Q3	FRAC55 R3
00	EXPONENT Q0	EXPONENT R0
01	EXTENSION R0	FRAC COUT
10	ZERO	EXT00 R0 SAVE
11	EXTENSION R0	ZERO

When the clock field equals alter fraction shift, the shift field is extended to include:

00	EXTENSION R0	EXPONENT R0
01	ONE	ONE
10	ZERO	EXT00 R0 SAVE
11	ZERO	ZERO

Left-Shift - when performing a left-shift, the shift field determines what is shifted into both the fraction and exponent.

SHF<1:0>	EXPONENT	FRACTION	
Q0	R0	Q0	R0
00 FRAC55 Q3	FRAC55 Q3	ZERO	ZERO
01 ZERO	ZERO	ZERO	FRAC55 R3 SV
10 ONE	ONE	ONE	ONE
11 FRAC55 Q3	FRAC55 R3	QIN	FRAC55 Q3

The last selection is for huge alignment shift; with the high part of the huge word in a QR and the low part in FQ it is possible to shift the entire huge word at once. Upon completion the huge word will be in FWR 55 - Ext 0 and FQ 55:7. Note that Qin drives the lower extension bit in the Q-register; this is always a zero for nondivide shifts.

Table 5-5 Control Store Field (Cont)

CS	Function	Description
17:15	Clock control field	This field can perform up to 11 functions when used in conjunction with the clock extend mod function. MOD not equal to clock extend. 1. CLK CTL=000 Enable clock for OP1=0 & OP2=0 This enables the clocks of two flip-flops (internal to a PAL) that indicate which, if any, of the operands are zero. The OP2=0 flip-flop is loaded with the EXP=0 signal, while the OP1=0 flip-flop is loaded with OP2=0. 2. CLK CTL=001 Clock Huge R3 Save This clock code saves FRAC55 R3 until the next time it is clocked by this code. This is needed to save R3 for huge divide. 3. CLK CTL=010 Null 4. CLK CTL=011 Alter fraction shift With this code, in conjunction with the shift field, it is possible to shift a one and zero into the MSB of the fraction SP and Q-register. 5. CLK CTL=100 Clock sign out This code enables the resultant sign flip-flop to be clocked. What function gets clocked into it is determined by the low three bits of the A address field. 6. CLK CTL=101 Clock OP2 sign This signal enables the clocking of the second operand's sign bit.

Table 5-5 Control Store Field (Cont)

CS	Function	Description
		7. CLT CTL=110 Clock CC This clocks the condition codes. The shift bits will set the V and C bits; this is for an error condition. Normally both shift bits should be cleared. 8. CLK CTL=111 Clock OP1 sign This signal enables the clocking of the first operand's sign bit. MOD=Extended clock function 1. CLK CTL=000 Toggle Alter Store This inverts the normal store from a floating store to an integer store, and vice versa. This is to be used for EMOD. 2. CLK CTL=001 Clock fast cycle This toggles the fast clock flip-flop. When this flip-flop is set, the cycle time is 180 ns; when clear it is 270 ns, in synchronization with the CPU. 3. CLK CTL=010 Enable Literal This enables an eight-bit literal onto the FPA BUS D14 - D07. This can be loaded into the exponent data path and the fraction datapath. When loading a constant into the fraction data path, the constant is loaded into EXT<6:0> and FRAC<30:23> simultaneously. In most cases it is desired to load the extension with a constant; the other sections should be masked out. 4. CLK CTL=011 Toggle load flip-flop This clock code sets the load flip-flop, so when the MOD field equals a load or store, the hardware interrupts it as a load. This signal clears the next time this code is asserted. The load signal is initialized to a zero by the FORCE UADDR signal.

Table 5-5 Control Store Field (Cont)

CS	Function	Description
		5. CLK CTL=100 Clock sign out This code enables the resultant sign flip-flop to be clocked. 6. CLK CTL=101 Alter CIN This clock enable forces the next state's fraction carry in to equal the current state's fraction carry out. This is used for huge addition. 7. CLK CTL=110 Default Q16 The code sets a bit which forces the multiplication logic to select FRAC16 Q0 as the LSB of the multiplier. This is used to multiply the mier extension. This signal is initialized to zero by the FORCE UADDR signal. 8. CLK CTL=111 Extended Branch This code extends the branch from 2 to 5 bits wide. (See the sequencer section for the actual branches.)
14:10	Branch control field (BCTL)	This field selects what status bits are to be ORed in with the UPF to generate the next microaddress (NUA). See the sequencer section for specific branches.
9:0	Micropointer field (UPF)	This field specifies the next microaddress. The UPF can be altered by the branch field. The lower 8 bits of this field serve as a literal field. When this function is used, the UPC must be used to address the control store.

5.8 DATA MANIPULATION

Floating-point operands that the CPU passes into the FPA are processed in data path logic (Figure 5-19) that manipulates the data (per control store output signals) until a result is sent to the CPU. As Figure 5-19 illustrates, the data path logic consists of exponent and fraction sections. All of the sections consist of 2901 4-bit slices.

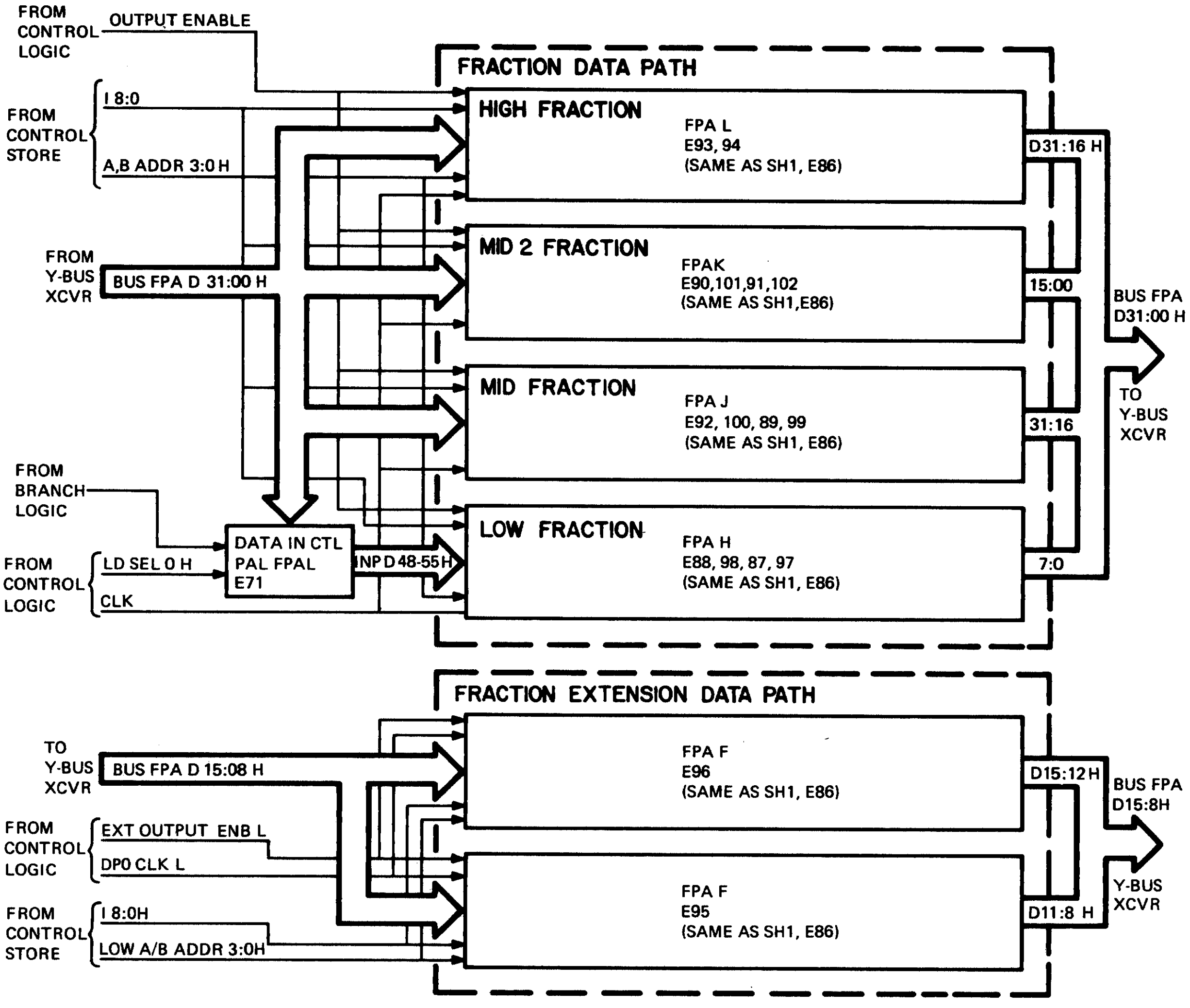


Figure 5-19 Data Path Logic (Sheet 1 of 3)

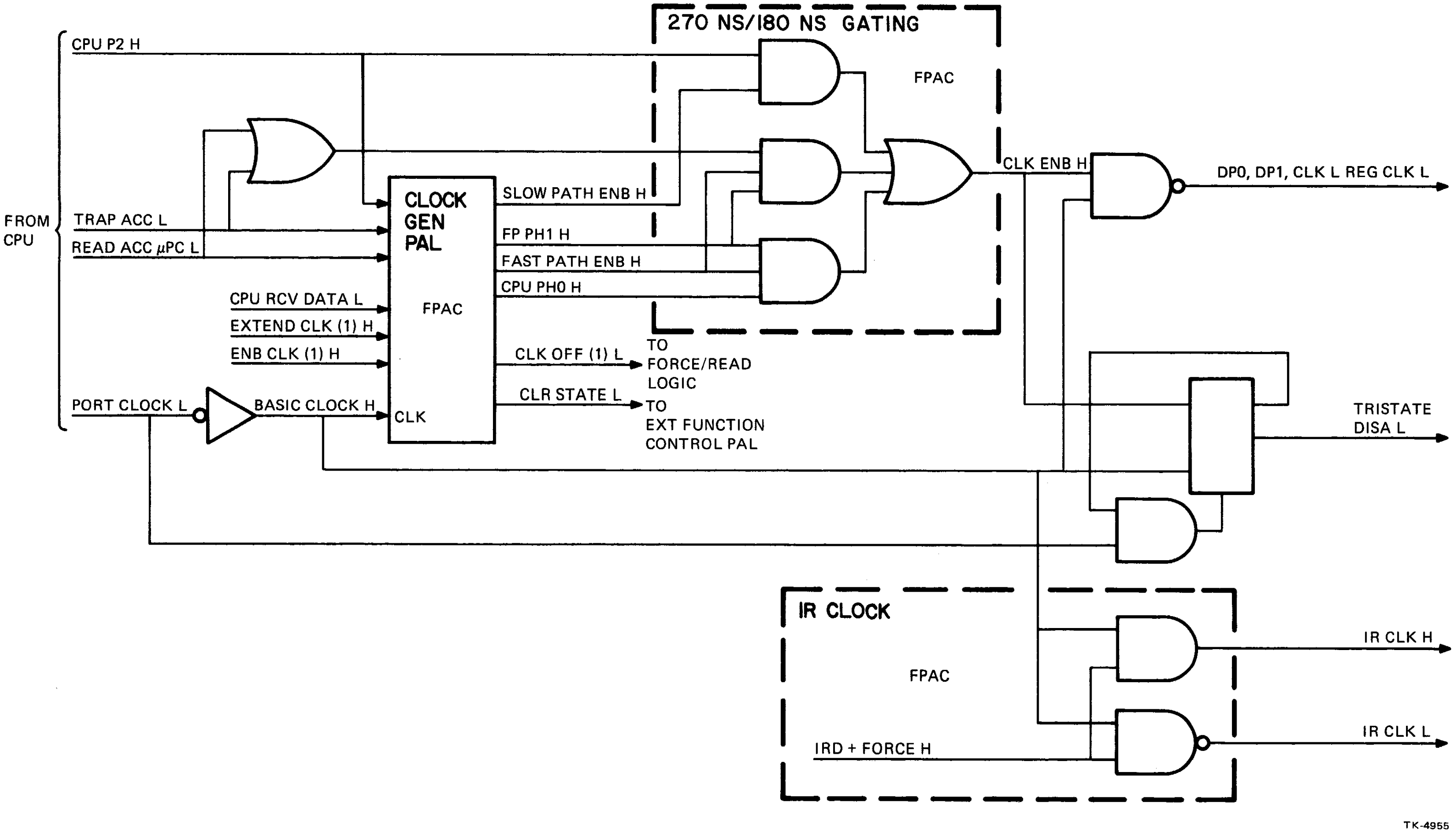


Figure 5-19 Data Path Logic (Sheet 2 of 3)

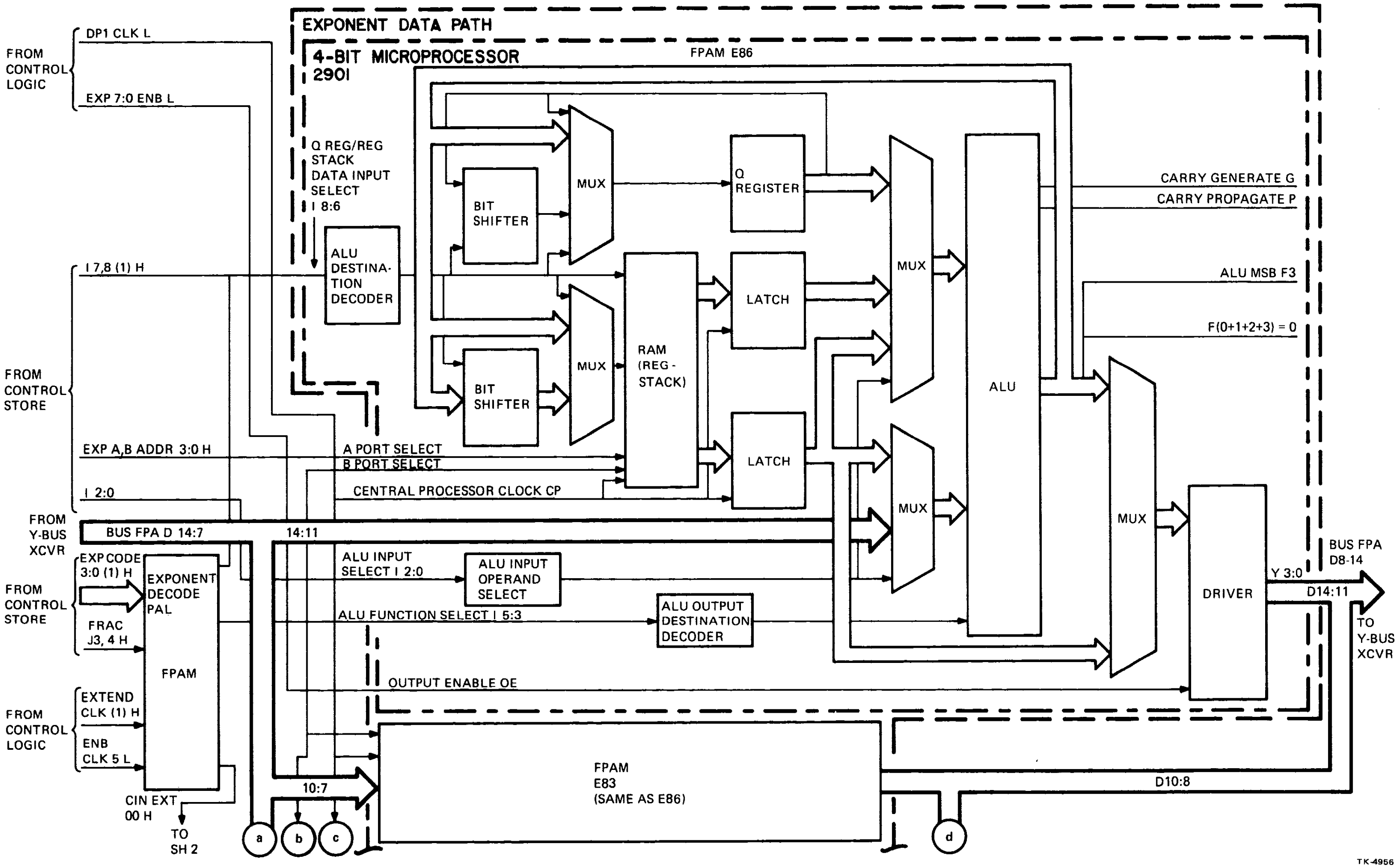


Figure 5-19 Data Path Logic (Sheet 3 of 3)

5.8.1 2901 Four-Bit Slice

The 2901 consists of a working register (RAM) (Figure 5-20), Q register, arithmetic logic unit (ALU), and control circuitry.

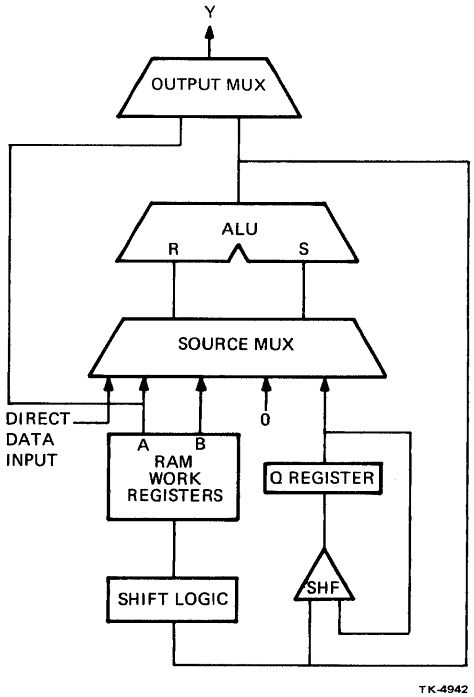


Figure 5-20 2901 Block Diagram

Working Register – The working register (WR) is the scratchpad area where results of arithmetic and logical operations can be stored.

Arithmetic Logic Unit (ALU) – The ALU is the data path component used to perform FPA arithmetic/logical operations, per commands in the control store output. The R inputs are applied to the ALU via a 3-input multiplexer, the inputs of which are direct data inputs, the output of the RAM A-port, and a zero. The ALU S input includes the RAM A- and B-ports, Q-register outputs, and a zero.

ALU output data (F) can be routed to the Q-register or WR, or multiplexed with the A-port output data from WR to drive the FPA bus. The ALU function decode determines the arithmetic or logical function to be performed, while the ALU destination decode determines which of the indicated registers the data is routed to, or whether it will be a data output of the device itself.

Q-Register – The Q-register is loaded from the ALU and is used to accumulate the quotient during division routines. It also functions as a temporary storage register. The Q-register output can be loaded back into itself, anad shifted right or left as during fraction, multiplication, and division operations.

5.8.2 Exponent Data Path

The exponent data path (Figure 5-21) is used for exponent operations, loop counting, and overflow and underflow testing. The exponent data path consists of four 4-bit microprocessors, each containing 16 working registers (WR). All 16 WRs are addressed via EXP A/B ADDR 3:0 from the control store. Some of the WRs contain constants which are listed in Table 5-6.

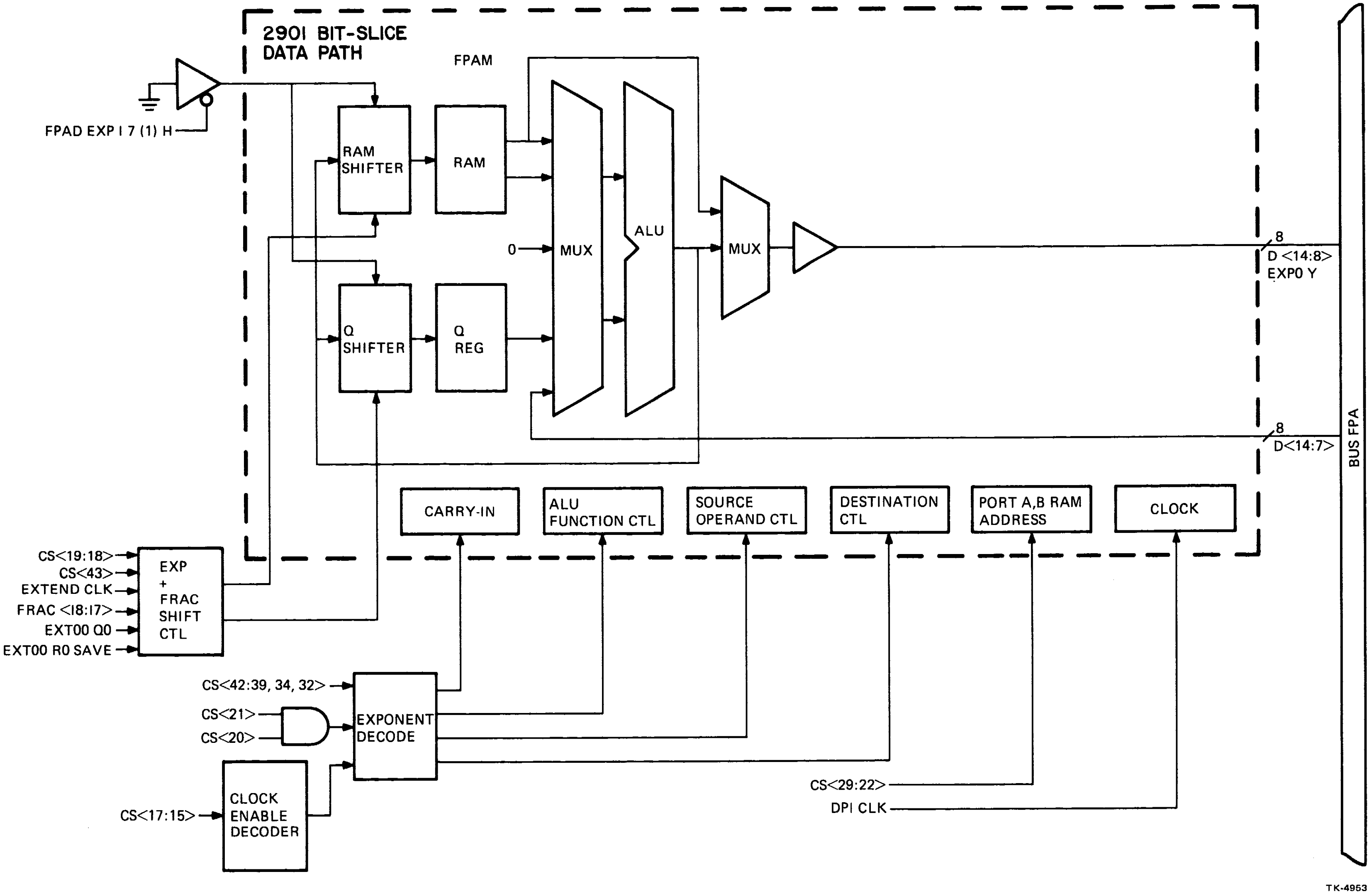


Figure 5-21 Exponent Data Path Logic

Table 5-6 **Exponent Working Register (RAM) Constants**

WR Address	Constant	Use
F	7FFF	Huge maximum exponent
E	0400	Grand bias
D	07FF	Grand maximum exponent
C	00FF	Float and double maximum exponent
B	4000	H-bias
A	0000	Zero constant
9	0001	One constant
3	18	Fraction bit count

The exponent data path source, ALU, and bit I_6 of the exponent destination field ($I_{6:8}$) are controlled by a decoding of EXP CODE 3:0 (1) H from the control store. Because of this, all of the 2901 functions (Table 5-7) are not available.

Table 5-7 **Exponent Function Selection**

EXP CODE 3:0 (1) H	Function Selected
0000	D OR 0
0001	B – A
0010	A – B
0011	B + A
0100	A OR B
0101	A AND B
0110	A – Q
0111	A + B + FRAC COUT
1000	Q – 1
1001	Q + 1
1010	A
1011	Q
1100	0
1110	SHIFT
1110	A + B = 1
1111	NOOP

5.8.3 Fraction Data Path

The fraction data path consists of 16 2901s and, therefore, is 64 bits wide. This width accommodates loading of huge operands. The fraction data path (Figure 5-19) consists of high fraction (55:32), middle fraction (31:00), and integer fraction (47:16) sections, plus an extension data path EXT (7:0).

The fraction data path is controlled by $I_{8:0}$ and A, B ADDR 3:0 H from the control store. Bits $I_{8:0}$ select the fraction function and A, B ADDR 3:0 H control scratchpads. The low and middle fraction sections are loaded directly from the FPA data bus. Part of the high fraction section (55:48) is loaded with data that passes through the hidden bit PAL.

Of the 16 64-bit working registers (RAM) in the fraction data path, seven contain constants as listed in Table 5-8.

Table 5-8 Fraction Data Path Working Register Constants

BR Address	Constant	Use
E	0000000000004000	Huge round
F	0000000000000080	Double round
G	0000000000000400	Grand round
C	0000008000000000	Floating round
B	00000000000000FF	Ext mask
A	00000001FFFFFFFF	Mid frac and ext mask
09	0000000000FFFFFF	Integer mask

The FPA internal 32-bit bus (BUS FPA D31:00) is not wide enough to load the entire 64-bit wide fraction data path. Working registers in the fraction data path are, therefore, loaded in sections. Whenever the working registers are loaded, the control fields are set up to perform

$$\mathbf{WR(X) \leftarrow D \text{ or } 0.}$$

Also, sections of the fraction data path can be forced to NOOP (no operation) by forcing I_7 to the fraction 2901's low. This changes a write WR function to a NOOP. The control store microword determines which sections are written via the modify and shift (MOD and SHF) fields.

5.8.4 Sign Logic

The FPA indicates to the CPU, via BUS FPA D15 H, what the resultant sign of the operation is. Sign logic consists of a PAL that is clocked with data from the FPA control logic.

The sign PAL (Figure 5-22) latches the sign of the first and second operands, the resultant sign (SIGN OUT), and a SUMPATH signal that indicates whether a sum or a difference operation is to be performed from an ADD or SUBtract instruction. The sign PAL contains a SIGN OUT register (resultant sign) that can be loaded with:

1. First operand's sign (OP1)
2. Second operand's sign (OP2)
3. First operand's sign XOR second operand's sign
4. First operand's sign XOR SIGN OUT
5. One
6. Zero

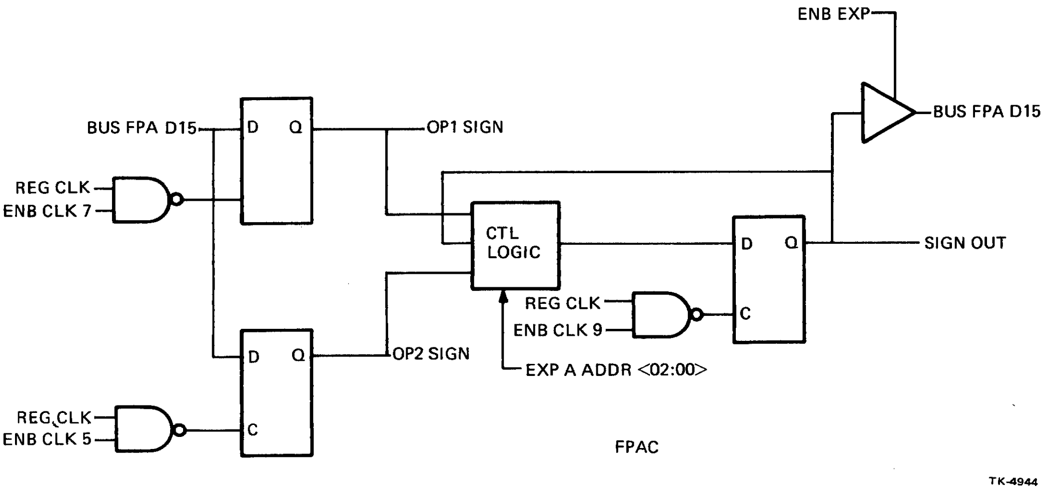


Figure 5-22 Sign Control PAL Logic

For most instructions performed by the FPA, the sign bits of the first and second operands are loaded into the PAL OP1 and OP2 flip-flops, during operand load routines. The SIGN OUT flip-flop in the PAL is then clocked with the resultant sign.

When the FPA processes a POLY instruction, the OP1 flip-flop in the PAL is loaded with the argument sign. Once loaded, it remains the same throughout the instruction. The OP2 flip-flop in the PAL is loaded each time with the coefficient sign. The PAL SIGN OUT flip-flop then contains the current resultant's sign. The sign PAL receives POLY H and EXP A ADDR 2:0 H inputs. It generates BUS FPA D15 H, SUMPATH (1) H, OP 1, 2 SIGN (1) H, and SIGN OUT (1) H outputs. The POLY H signal is from the FPA branch logic, and EXP A ADDR 2:0 H is generated in the control store. BUS FPA D15 H is sent to the CPU and the other outputs [SUMPATH (1) H, OP1, 2, SIGN (1) H, SIGN OUT (1) H] are applied to the FPA branch logic. The sign PAL SIGN OUT function is controlled via the control store EXP A ADDR 2:0 H output. The functions selected, via encoding of this field, are listed in Table 5-9.

Table 5-9 Sign PAL Function Control Encoding

EXP A ADDR 2:0 H Octal Value	SIGN OUT PAL Signal
0	OP1 SIGN
1	OP2 SIGN
2	OP1 SIGN XOR OPS SIGN
3	OP1 SIGN XOR SIGN OUT
4	ZERO
5	ONE
6	ZERO
7	ONE

5.9 MAINTAINABILITY FUNCTIONS

The FPA contains logic that enables the CPU to force the FPA to any microaddress. This is done via a TRAP ACC L or READ ACC UPC L signal, and microaddress force/read logic that consists of a force/read control, transceiver enable, and bus transceiver.

5.9.1 Force Microaddress

When the CPU generates TRAP ACC L the microaddress force/read logic (Figure 5-23) generates FORCE UADDR (1) H. This is used to inhibit the microsequencer output. The CPU applies an address to the Y-Bus transceiver as BUS Y D09:00 H. The BUS NUA 9:0 H output of the FPA microaddress force/read logic is then applied to the control store in lieu of the inhibited microsequencer BUS NUA 9:0 H output.

5.9.2 Read Microaddress

During microdiagnostics the microaddress read logic is used to read the microsequencer BUS NUA 9:0 H output onto the Y-Bus for subsequent transmission to the CPU. During a force read operation (Figure 5-23) the CPU asserts READ ACC UPC L. This inhibits operation of the FPA clocks. It also places the microsequencer BUS NUA 9:0 H output onto the FPA data bus via the microaddress force/read logic bus transceiver. The next time the CPU generates RCV DATA L, the BUS NUA 9:0 H output will be applied to the Y-Bus as BUS Y D9:0 H. The RCV DATA L signal will also restart the FPA clocks.

5.10 PARITY LOGIC

Parity is checked on each 48-bit microword that the microsequencer accesses from the control store. There are only two parity bits and each corresponds to certain sections of the microword. Figures 5-24 and 5-25 illustrate which fields are checked by the parity bits. The parity logic consists of three parity checkers, a PROM and a parity control PAL. The sum of the parity bit and the bits in the field that it covers should be even.

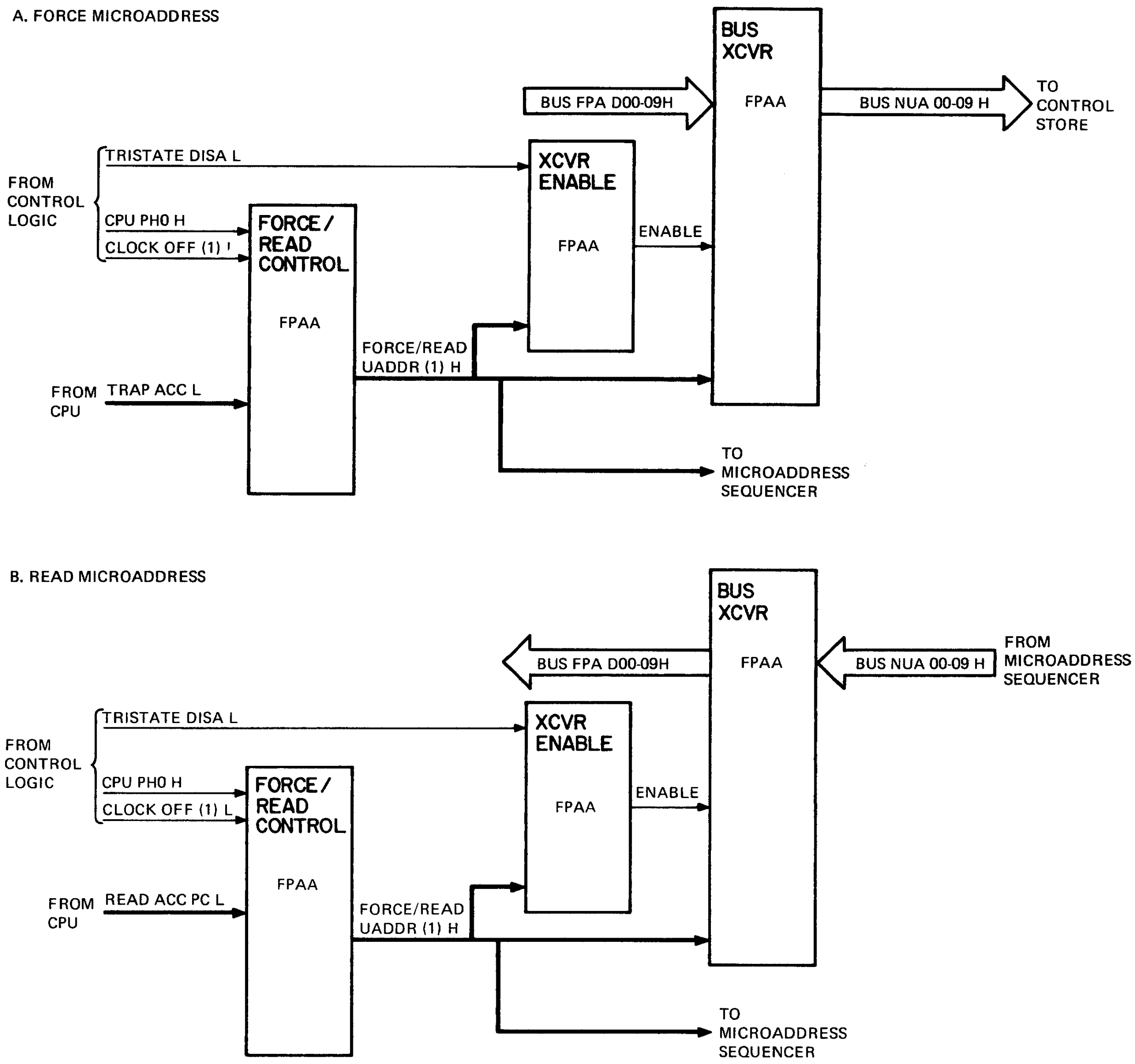


Figure 5-23 Force/Read Microaddress Control

When a parity error is detected the parity logic generates a FORCE LOW UADDR L output that drives the microsequencer NUA 9:0 H output to logical 0. This starts a parity handler routine that simply loops in microaddress 0, continuously storing the parity error. The CPU initially interprets this as an exception and asks for an error code. The FPA then passes the error code. The FPA passes the parity error again which the CPU interprets as a parity error. The FPA must be forced out of the error routine by the CPU.

The parity control PAL output is BUS FPA D3:0 H and FORCE LOW UADDR L. Of the 4-bit field output, BUS FPA D00 will be set to logical 1 whenever parity error 1 or 0 is detected. This bit informs the CPU that a parity error has occurred.

The error bits that become set in the parity control PAL will remain set on the BUS FPA D3:0 H output lines until cleared by FORCE UADDR (1) H. They are placed on the BUS FPA bus by the READ UADDR (1) H signal.

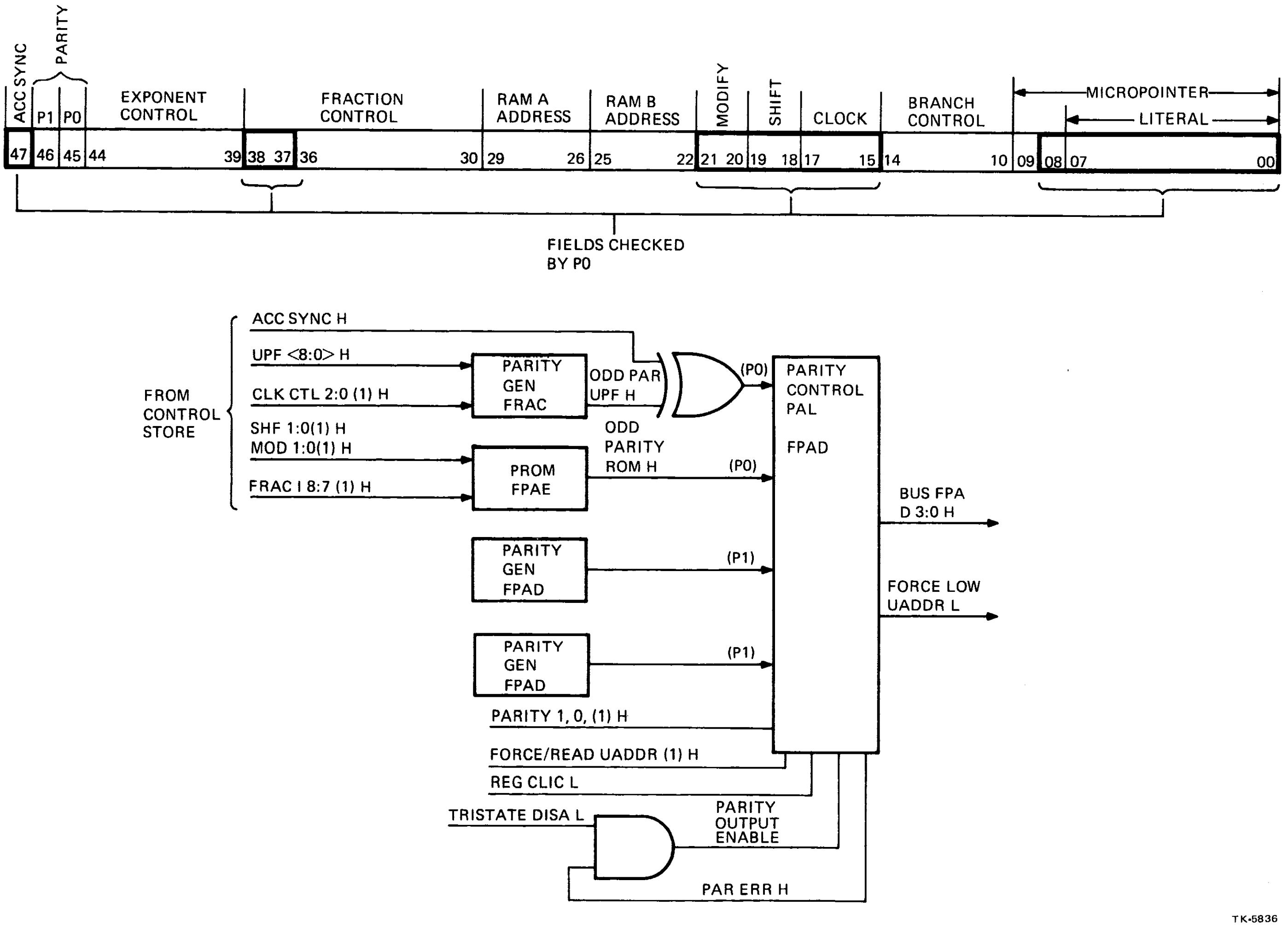


Figure 5-24 Control Store Fields Checked by Parity Bit P0

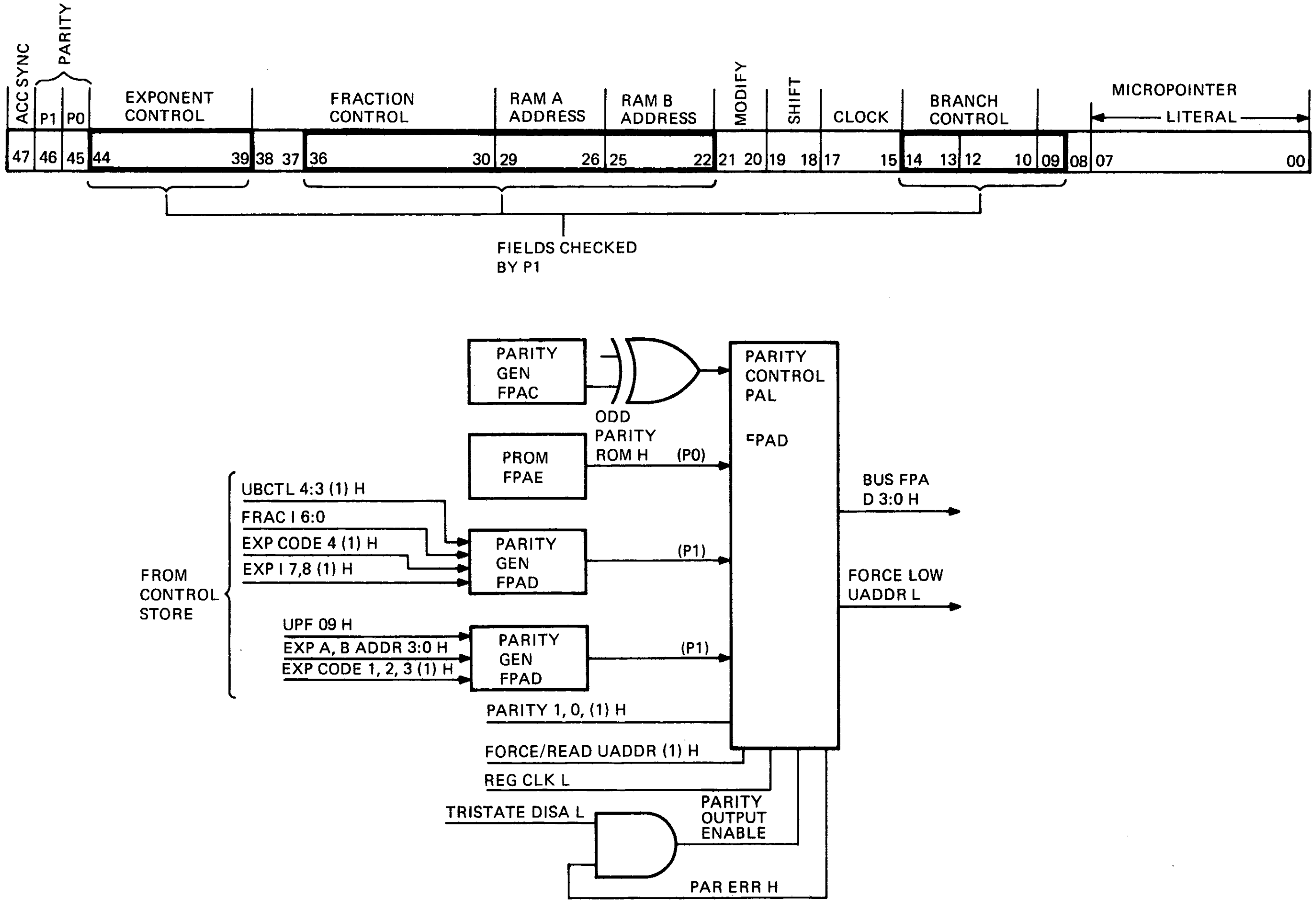


Figure 5-25 Control Store Fields Checked by Parity Bit P1

CHAPTER 6
MICROCODE DESCRIPTIONS

6.1 GENERAL

The FPA microlisting consists of a definitions file followed by microcode routines. The definitions file defines the microfield and macros. The macros equate a mnemonic statement such as ADD, with a particular set of microfields that will perform the operation specified.

6.2 FIELD DEFINITIONS

Figure 6-1 explains the first four lines of FPA microcode and illustrates field locations in the 48-bit control store microword.

Figures 6-2 through 6-19 explain the fields.

6.3 MACRODEFINITIONS

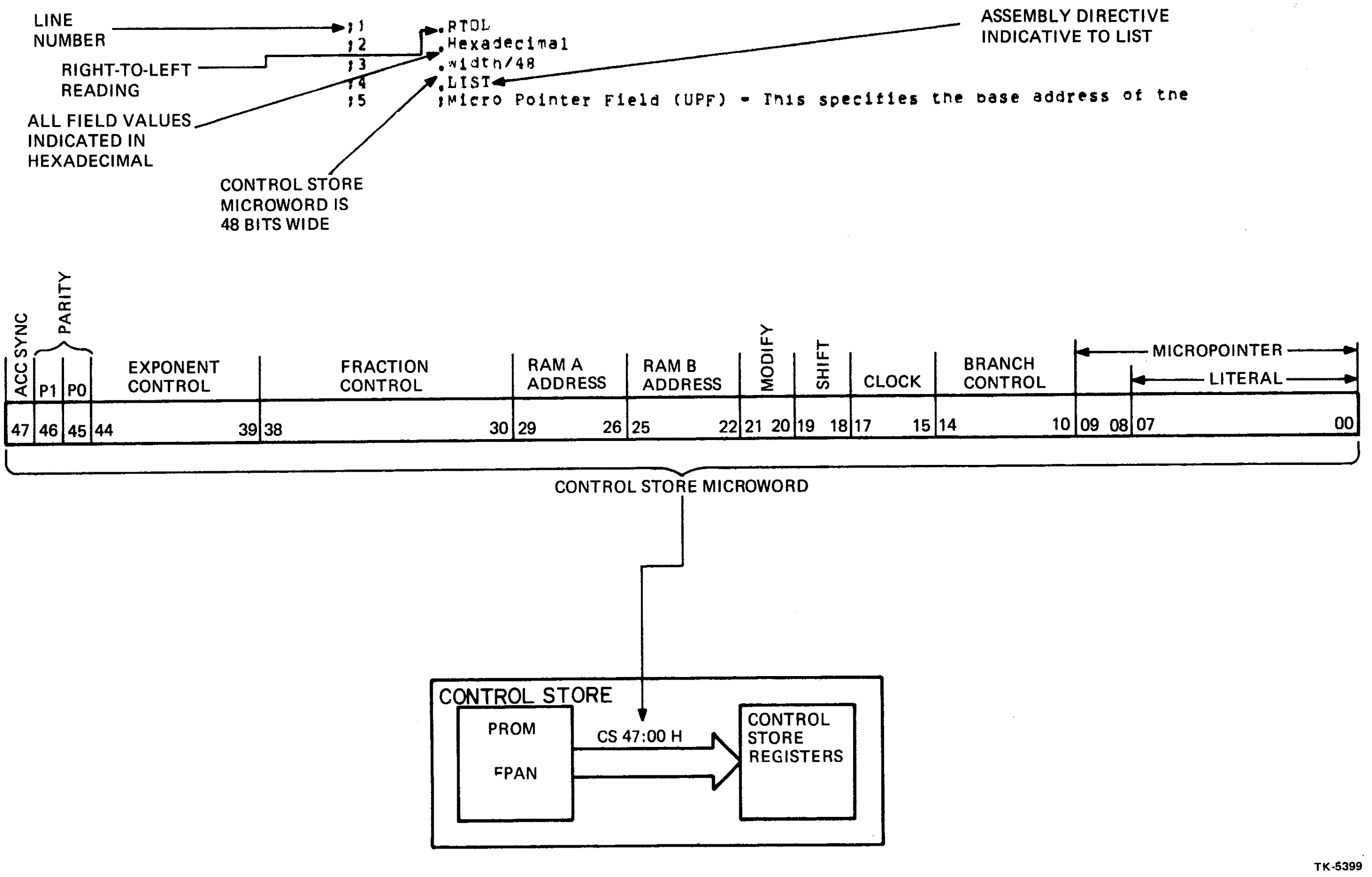
The FPA macrodefinitions consist of symbols, the value of which is one or more field value (Figure 6-2 through 6-19) and/or macros. The macrodefinitions shown consist of a line containing a macro name followed by a string in quotations which specifies the values of one or more of the microcode fields.

MNEG FWR[] to FQ "FSRC/O.A, FALU/R.MINUS, FSHF/LOADQ,FA.ADRS/@/"

Macros may include square brackets ([]) which open a microcode field but do not give it a particular value. The desired field value is inserted inside the brackets whenever this macro is used.

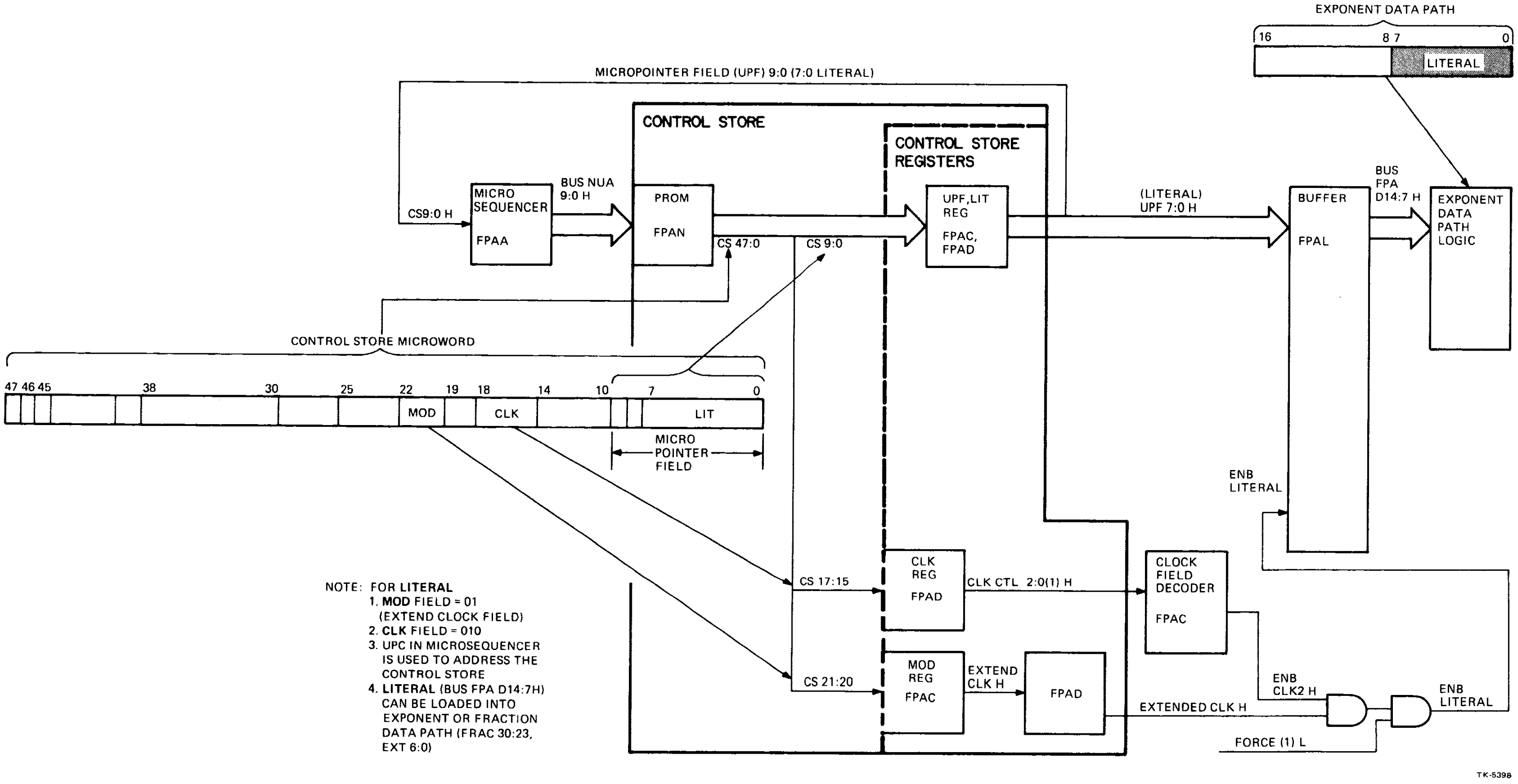
Headers generally located at the beginning of each macro describe what the macro does.

Figure 6-20 shows a section of the macrodefinitions file.



```
;1   .RTOL
;2   .Hexadecimal
;3   .width/48
;4   .LIST
;5   ;Micro Pointer Field (UPF) = This specifies the base address of the
```

Figure 6-1 Field Definitions



NOTE: FOR **LITERAL**

1. **MOD** FIELD = 01 (EXTEND CLOCK FIELD)
2. **CLK** FIELD = 010
3. UPC IN MICROSEQUENCER IS USED TO ADDRESS THE CONTROL STORE
4. **LITERAL** (BUS FPA D14:7H) CAN BE LOADED INTO EXPONENT OR FRACTION DATA PATH (FRAC 30:23, EXT 6:0)

Figure 6-2 Literal Field

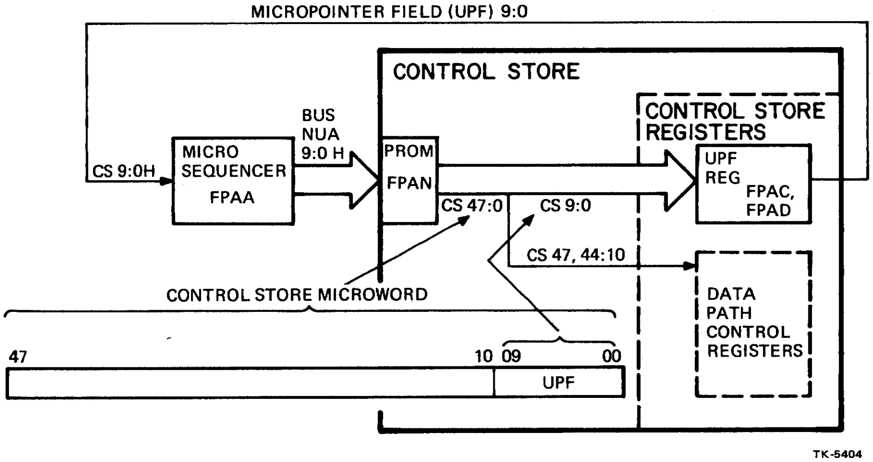
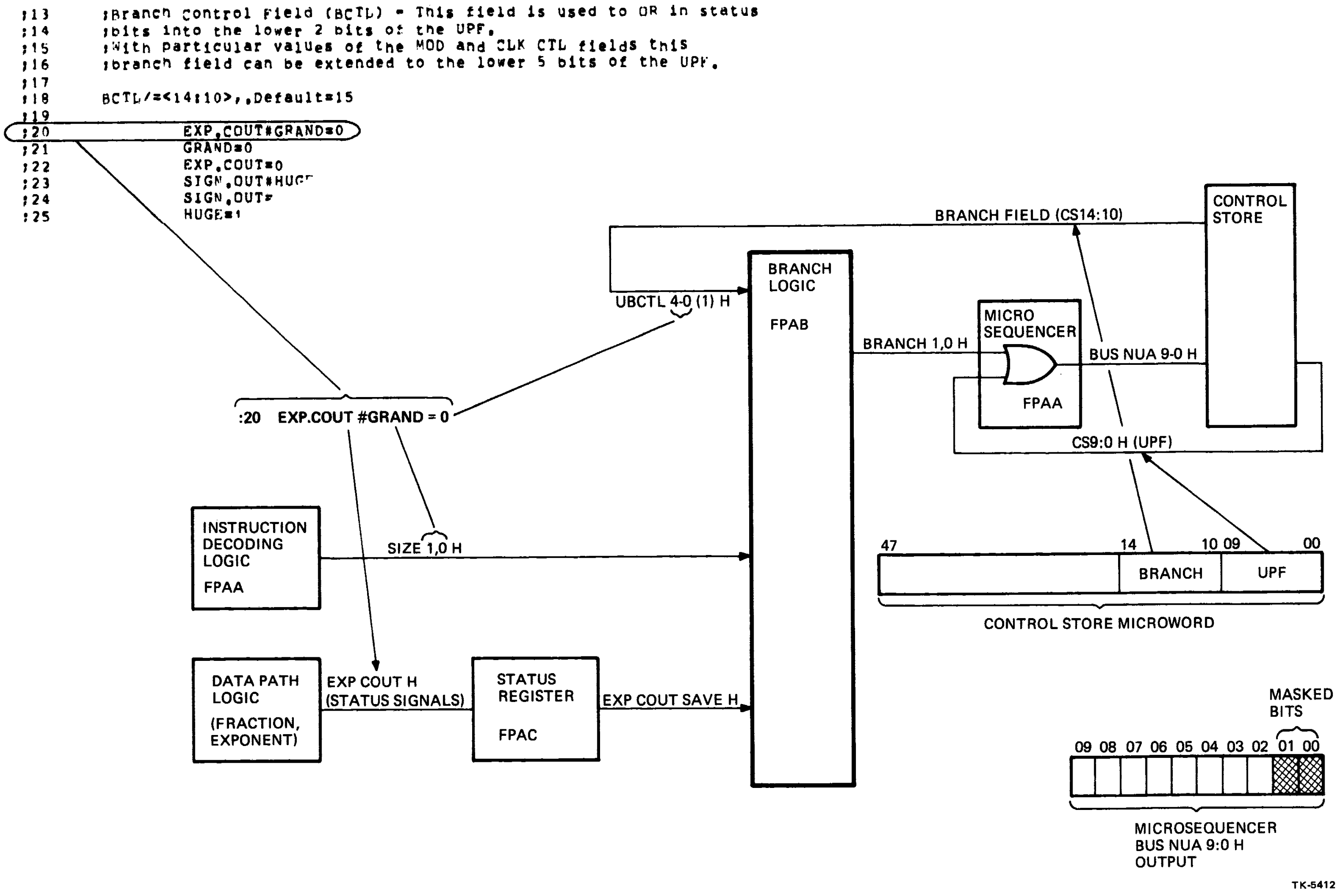
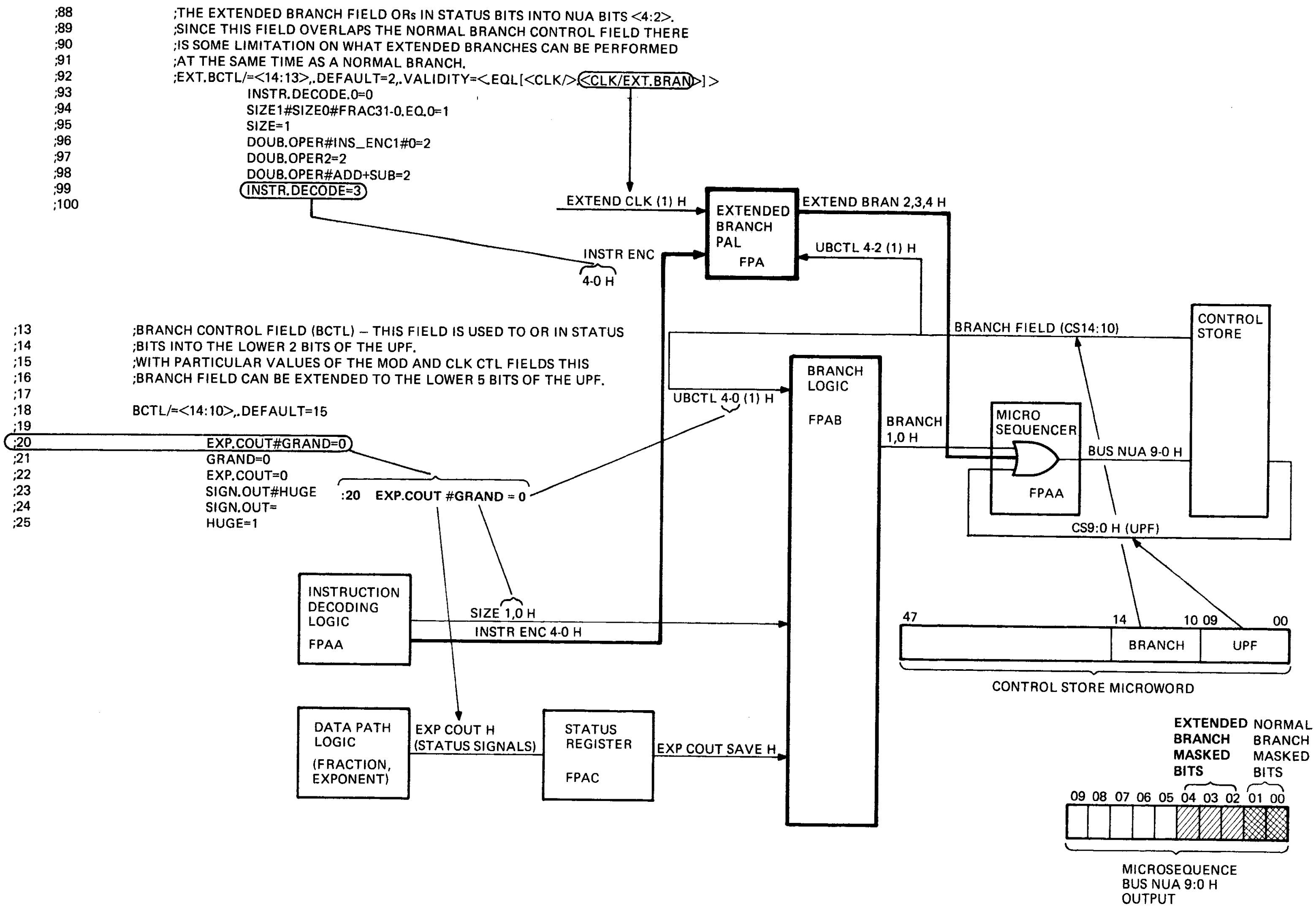


Figure 6-3 Micropointer Field



```
;13     ;Branch control Field (BCTL) - This field is used to OR in status
;14     ;bits into the lower 2 bits of the UPF,
;15     ;With particular values of the MOD and CLK CTL fields this
;16     ;branch field can be extended to the lower 5 bits of the UPF.
;17
;18     BCTL/=<14:10>,,Default=15
;19
;20             EXP,COUT#GRAND=0
;21             GRAND=0
;22             EXP,COUT=0
;23             SIGN,OUT#HUGE
;24             SIGN,OUT=
;25             HUGE=1
```

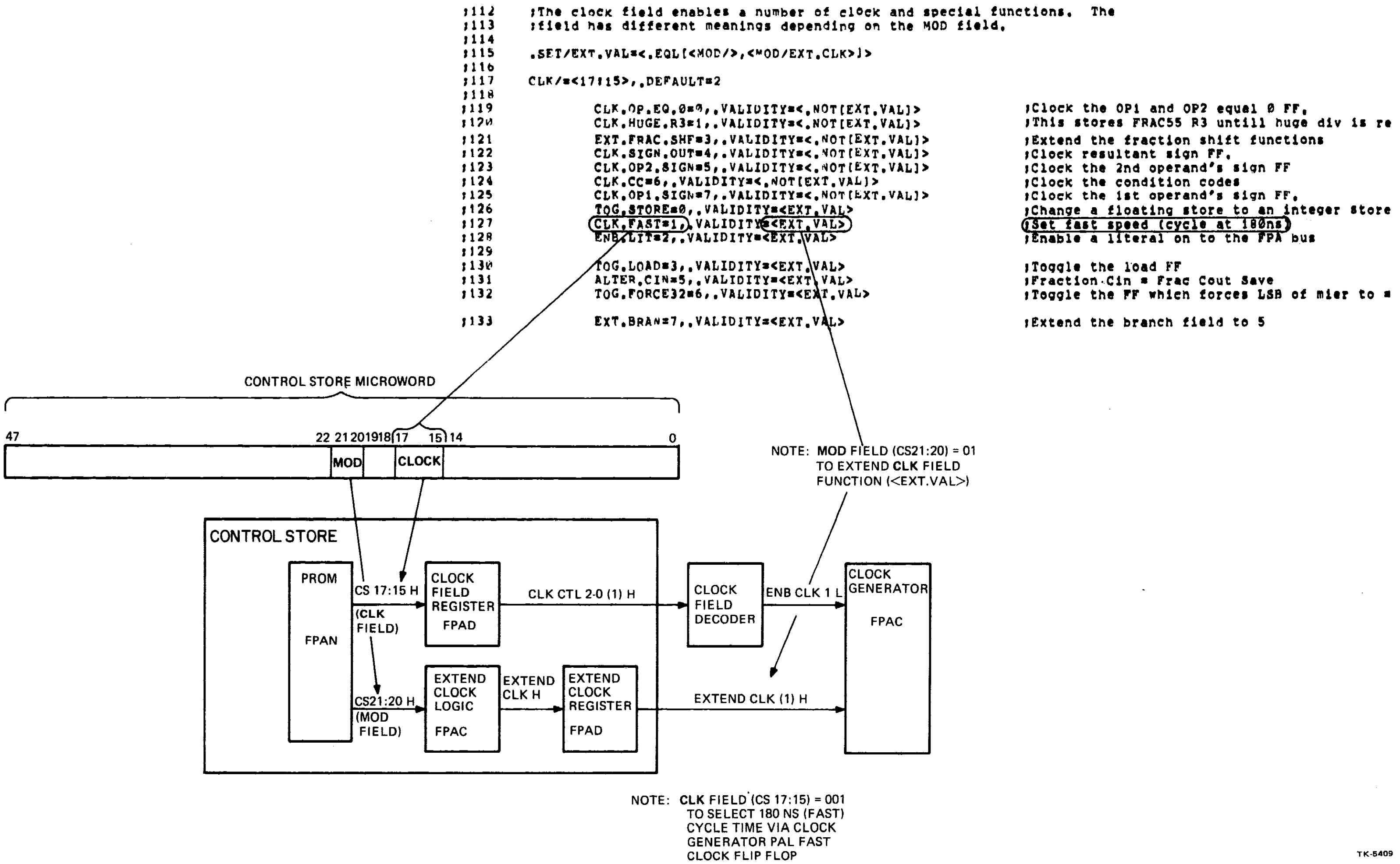
Figure 6-4 Branch Field



```
;88        ;THE EXTENDED BRANCH FIELD ORs IN STATUS BITS INTO NUA BITS <4:2>.
;89        ;SINCE THIS FIELD OVERLAPS THE NORMAL BRANCH CONTROL FIELD THERE
;90        ;IS SOME LIMITATION ON WHAT EXTENDED BRANCHES CAN BE PERFORMED
;91        ;AT THE SAME TIME AS A NORMAL BRANCH.
;92        ;EXT.BCTL/=<14:13>,.DEFAULT=2,.VALIDITY=<.EQL[<CLK/><CLK/EXT.BRAN>]>
;93                 INSTR.DECODE.0=0
;94                 SIZE1#SIZE0#FRAC31-0.EQ.0=1
;95                 SIZE=1
;96                 DOUB.OPER#INS_ENC1#0=2
;97                 DOUB.OPER2=2
;98                 DOUB.OPER#ADD+SUB=2
;99                 INSTR.DECODE=3
;100
```

```
;13        ;BRANCH CONTROL FIELD (BCTL) – THIS FIELD IS USED TO OR IN STATUS
;14        ;BITS INTO THE LOWER 2 BITS OF THE UPF.
;15        ;WITH PARTICULAR VALUES OF THE MOD AND CLK CTL FIELDS THIS
;16        ;BRANCH FIELD CAN BE EXTENDED TO THE LOWER 5 BITS OF THE UPF.
;17
;18        BCTL/=<14:10>,.DEFAULT=15
;19
;20                 EXP.COUT#GRAND=0
;21                 GRAND=0
;22                 EXP.COUT=0
;23                 SIGN.OUT#HUGE
;24                 SIGN.OUT=
;25                 HUGE=1
```

Figure 6-5 Extended Branch Field

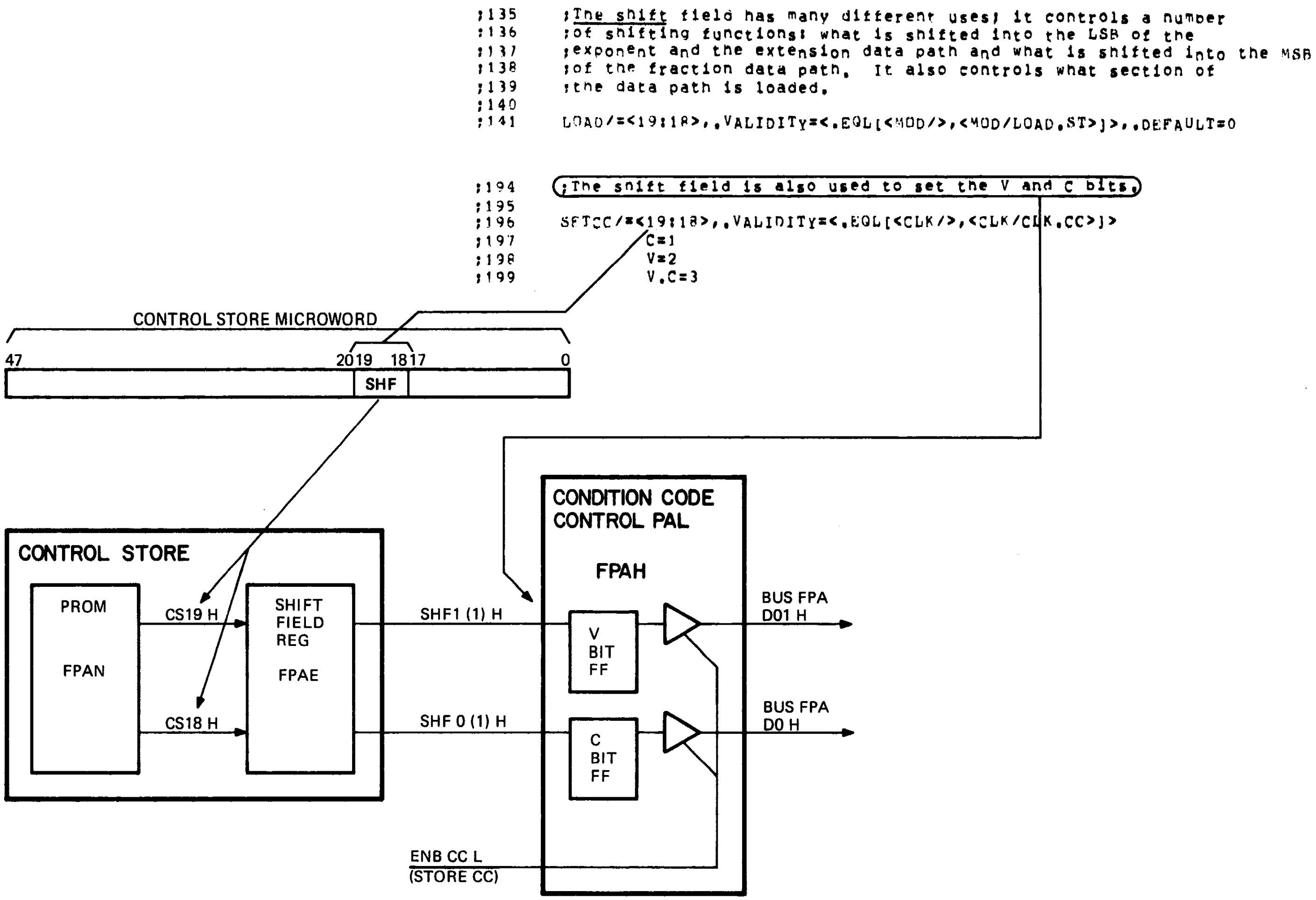


```
;The clock field enables a number of clock and special functions.  The
;field has different meanings depending on the MOD field.

.SET/EXT.VAL=<.EQL[<MOD/>,<MOD/EXT.CLK>]>

CLK/=<17:15>,.DEFAULT=2

        CLK.OP.EQ.0=0,.VALIDITY=<.NOT[EXT.VAL]>        ;Clock the OP1 and OP2 equal 0 FF.
        CLK.HUGE.R3=1,.VALIDITY=<.NOT[EXT.VAL]>        ;This stores FRAC55 R3 untill huge div is re
        EXT.FRAC.SHF=3,.VALIDITY=<.NOT[EXT.VAL]>       ;Extend the fraction shift functions
        CLK.SIGN.OUT=4,.VALIDITY=<.NOT[EXT.VAL]>       ;Clock resultant sign FF.
        CLK.OP2.SIGN=5,.VALIDITY=<.NOT[EXT.VAL]>       ;Clock the 2nd operand's sign FF
        CLK.CC=6,.VALIDITY=<.NOT[EXT.VAL]>             ;Clock the condition codes
        CLK.OP1.SIGN=7,.VALIDITY=<.NOT[EXT.VAL]>       ;Clock the 1st operand's sign FF.
        TOG.STORE=0,.VALIDITY=<EXT.VAL>                ;Change a floating store to an integer store
        CLK.FAST=1,.VALIDITY=<EXT.VAL>                 ;Set fast speed (cycle at 180ns)
        ENB.LIT=2,.VALIDITY=<EXT.VAL>                  ;Enable a literal on to the FPA bus

        TOG.LOAD=3,.VALIDITY=<EXT.VAL>                 ;Toggle the load FF
        ALTER.CIN=5,.VALIDITY=<EXT.VAL>                ;Fraction Cin = Frac Cout Save
        TOG.FORCE32=6,.VALIDITY=<EXT.VAL>              ;Toggle the FF which forces LSB of mier to =

        EXT.BRAN=7,.VALIDITY=<EXT.VAL>                 ;Extend the branch field to 5
```

NOTE: MOD FIELD (CS21:20) = 01
TO EXTEND CLK FIELD
FUNCTION (<EXT.VAL>)

NOTE: CLK FIELD (CS 17:15) = 001
TO SELECT 180 NS (FAST)
CYCLE TIME VIA CLOCK
GENERATOR PAL FAST
CLOCK FLIP FLOP

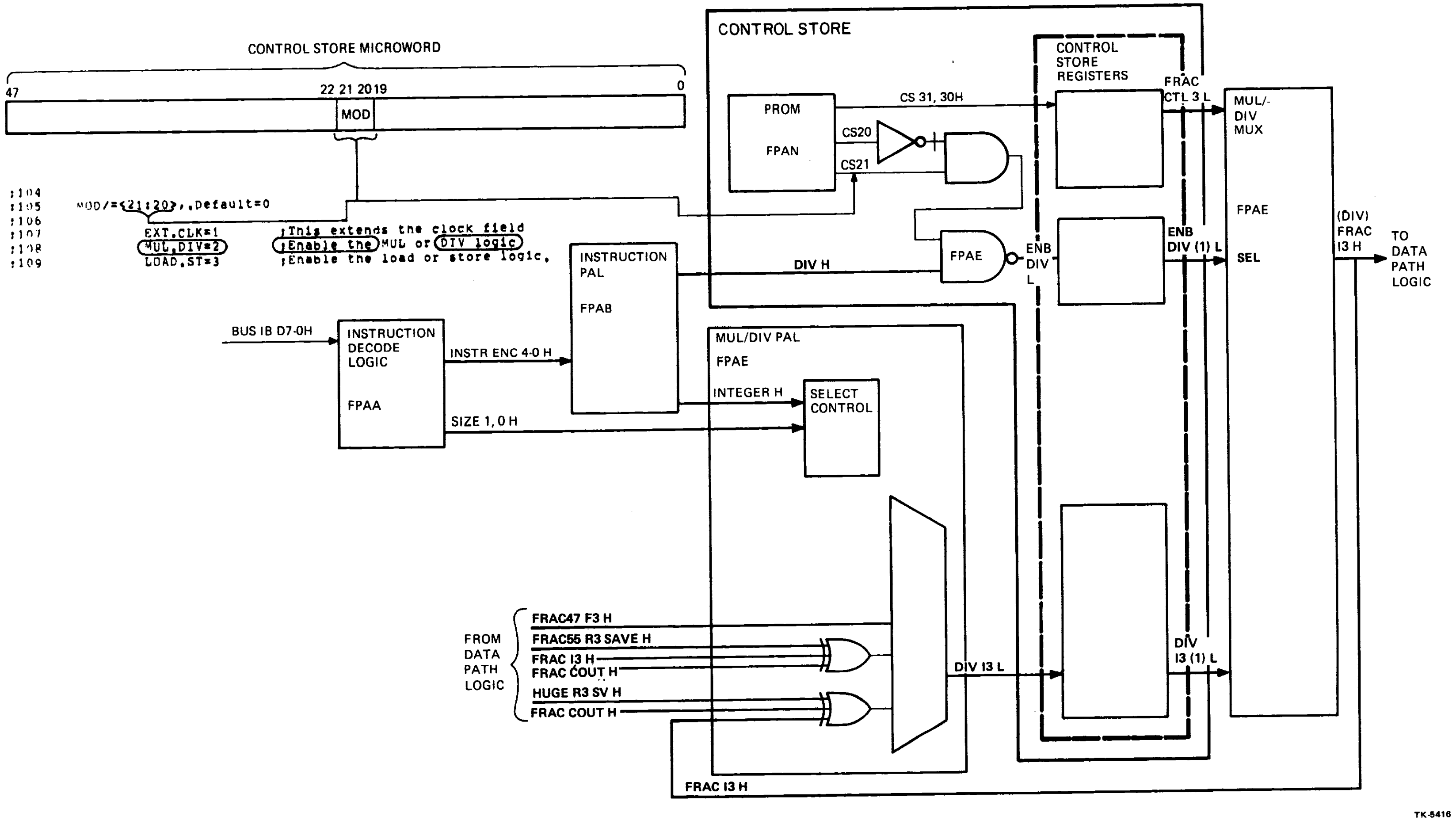
Figure 6-6 Clock Field (Used to Clock Fast Cycle)



```
;The shift field has many different uses; it controls a number
;of shifting functions: what is shifted into the LSB of the
;exponent and the extension data path and what is shifted into the MSB
;of the fraction data path.  It also controls what section of
;the data path is loaded.

LOAD/=<19:18>,,VALIDITY=<,EQL[<MOD/>,<MOD/LOAD,ST>]>,,DEFAULT=0
```

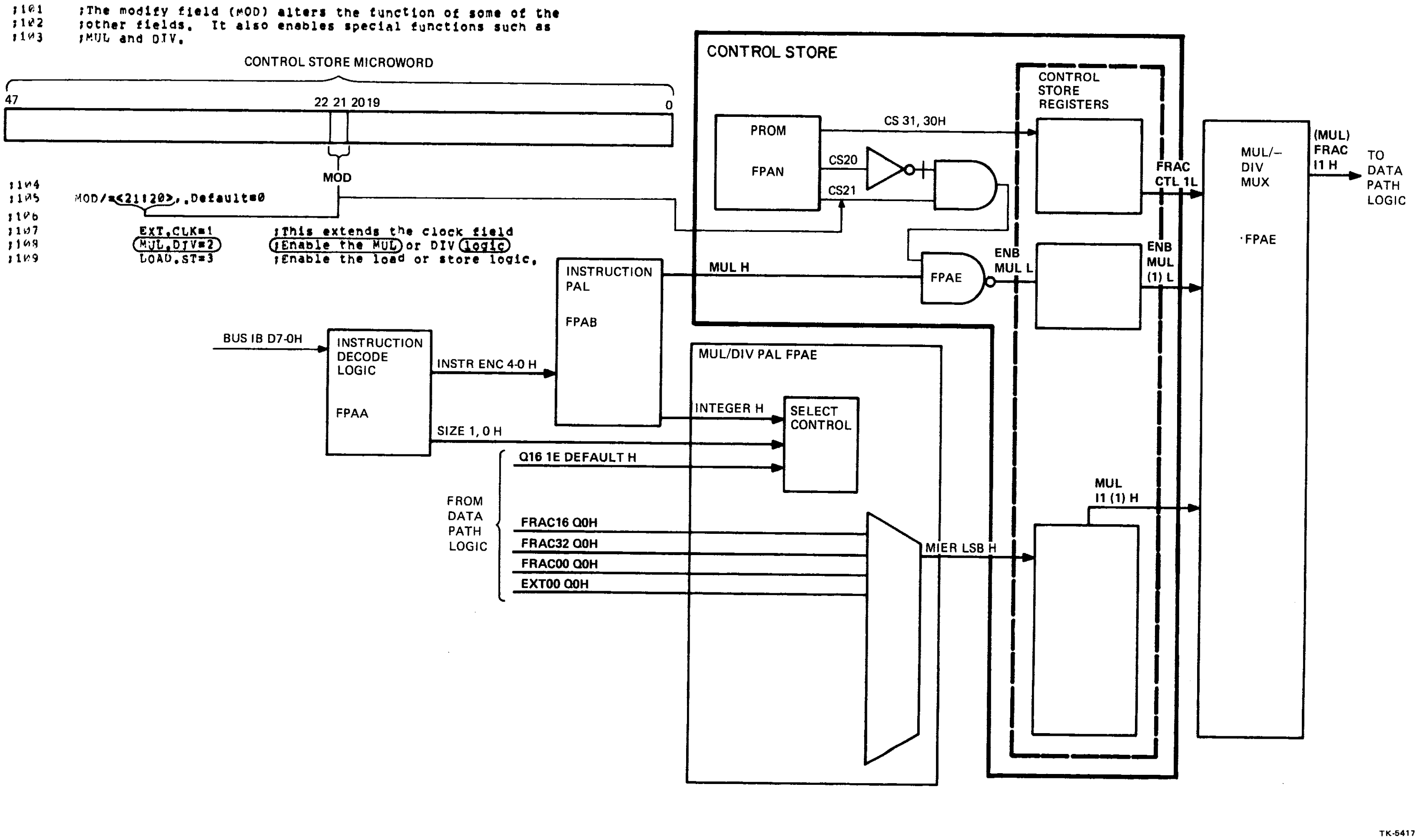
```
;The shift field is also used to set the V and C bits.

SFTCC/=<19:18>,,VALIDITY=<,EQL[<CLK/>,<CLK/CLK,CC>]>
               C=1
               V=2
               V,C=3
```

Figure 6-7 Shift Field (Used to Set V and C Bits)



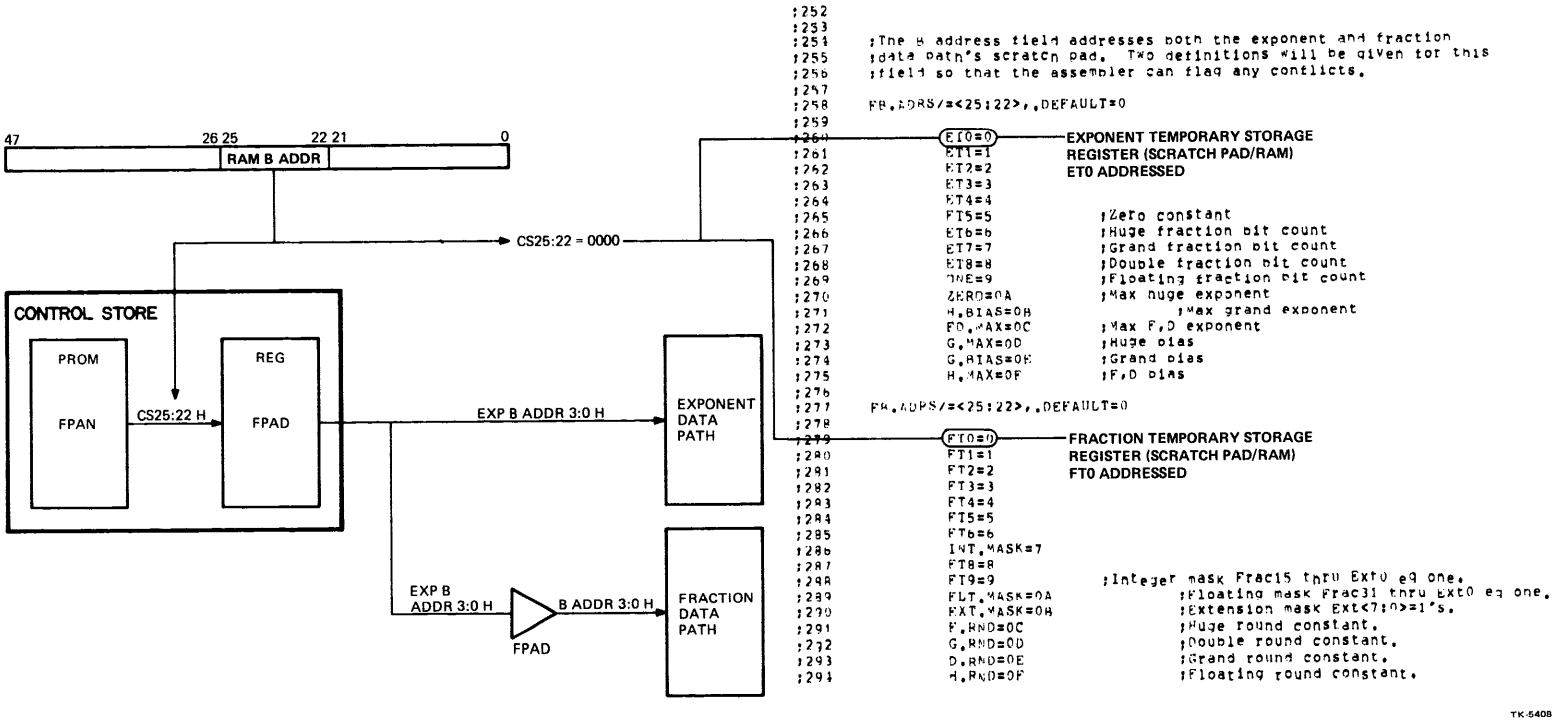
```
;104
;105     MOD/=<21:20>,.Default=0
;106
;107           EXT.CLK=1          ;This extends the clock field
;108           MUL.DIV=2          ;Enable the MUL or DIV logic
;109           LOAD.ST=3          ;Enable the load or store logic.
```

Figure 6-8 Modify Field (Used to Enable Division)



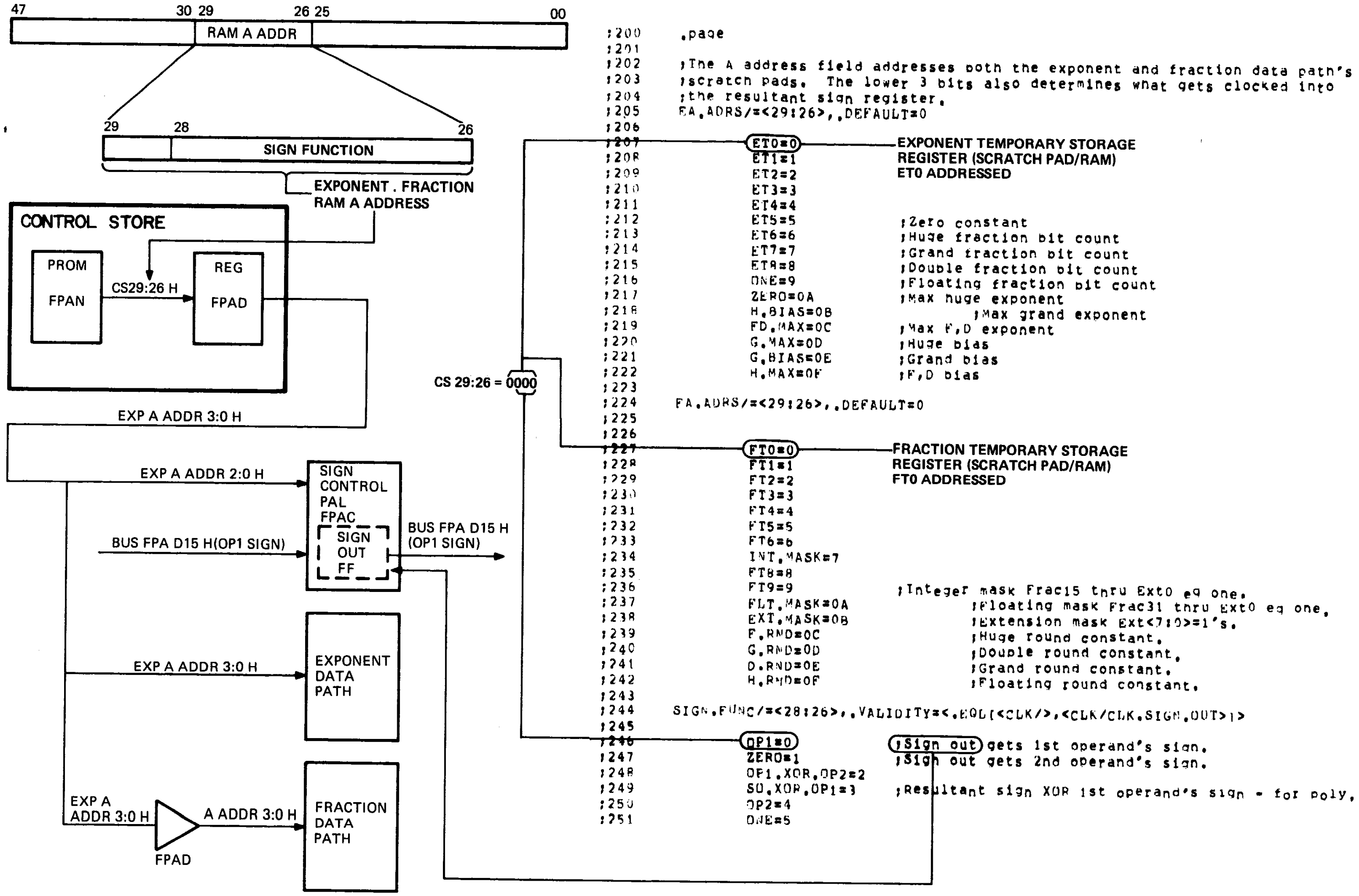
```
;The modify field (MOD) alters the function of some of the
;other fields.  It also enables special functions such as
;MUL and DIV.

MOD/=<21:20>,.Default=0

        EXT.CLK=1           ;This extends the clock field
        MUL.DIV=2           ;Enable the MUL or DIV logic
        LOAD.ST=3           ;Enable the load or store logic.
```

Figure 6-9 Modify Field (Used to Enable Multiplication)



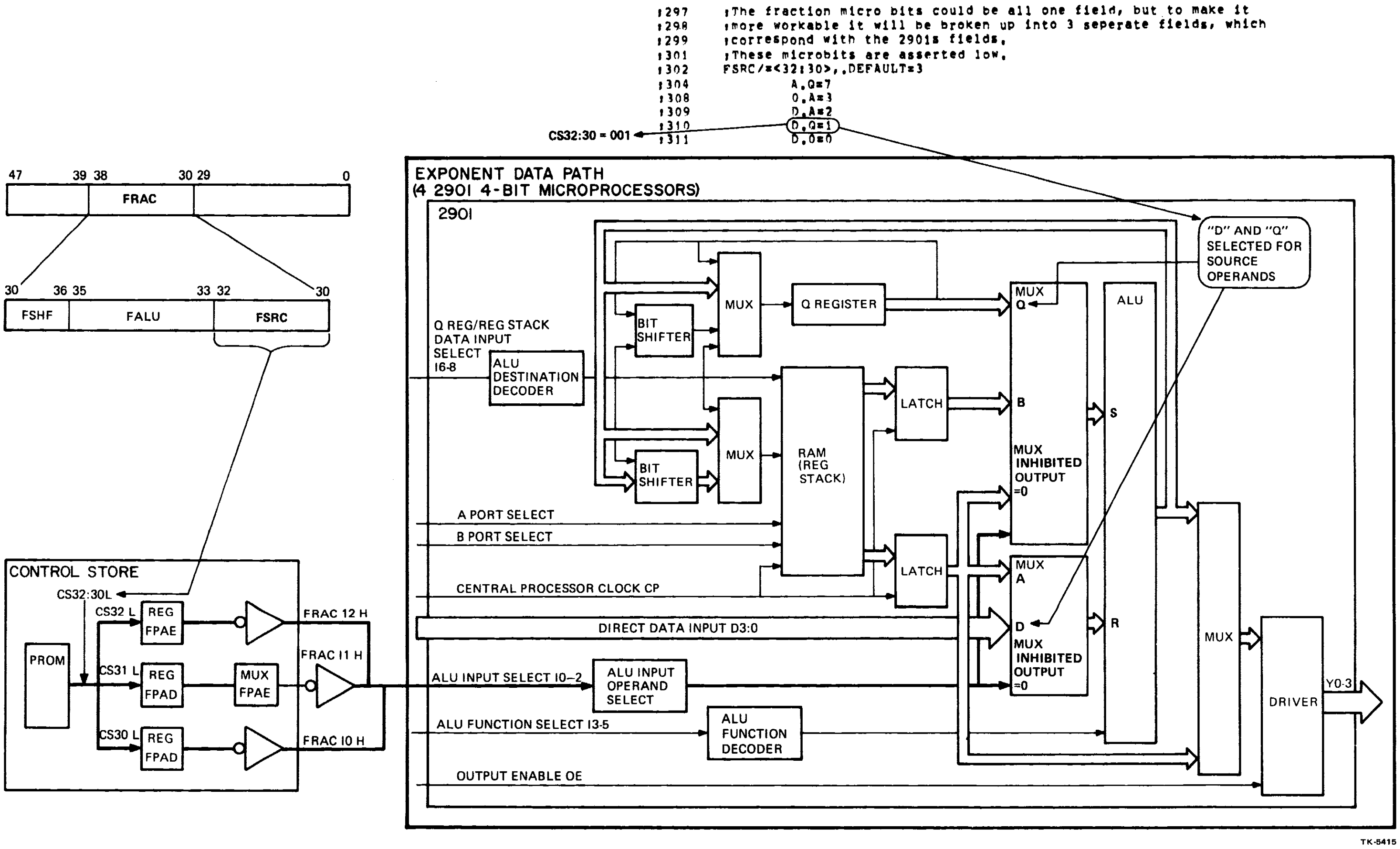
```


        ;The B address field addresses both the exponent and fraction
        ;data path's scratch pad.  Two definitions will be given for this
        ;field so that the assembler can flag any conflicts.

        FB.ADRS/=<25:22>,,DEFAULT=0

                ET0=0
                ET1=1
                ET2=2
                ET3=3
                ET4=4
                FT5=5               ;Zero constant
                ET6=6               ;Huge fraction bit count
                ET7=7               ;Grand fraction bit count
                ET8=8               ;Double fraction bit count
                ONE=9               ;Floating fraction bit count
                ZERO=0A             ;Max huge exponent
                H.BIAS=0B                   ;Max grand exponent
                FD.MAX=0C           ;Max F,D exponent
                G.MAX=0D            ;Huge bias
                G.BIAS=0E           ;Grand bias
                H.MAX=0F            ;F,D bias

        FR.ADRS/=<25:22>,,DEFAULT=0

                FT0=0
                FT1=1
                FT2=2
                FT3=3
                FT4=4
                FT5=5
                FT6=6
                INT.MASK=7
                FT8=8
                FT9=9               ;Integer mask Frac15 thru Ext0 eq one.
                FLT.MASK=0A                 ;Floating mask Frac31 thru Ext0 eq one.
                EXT.MASK=0B                 ;Extension mask Ext<7:0>=1's.
                F.RND=0C                    ;Huge round constant.
                G.RND=0D                    ;Double round constant.
                D.RND=0E                    ;Grand round constant.
                H.RND=0F                    ;Floating round constant.
```

Figure 6-10 RAM B Address Field



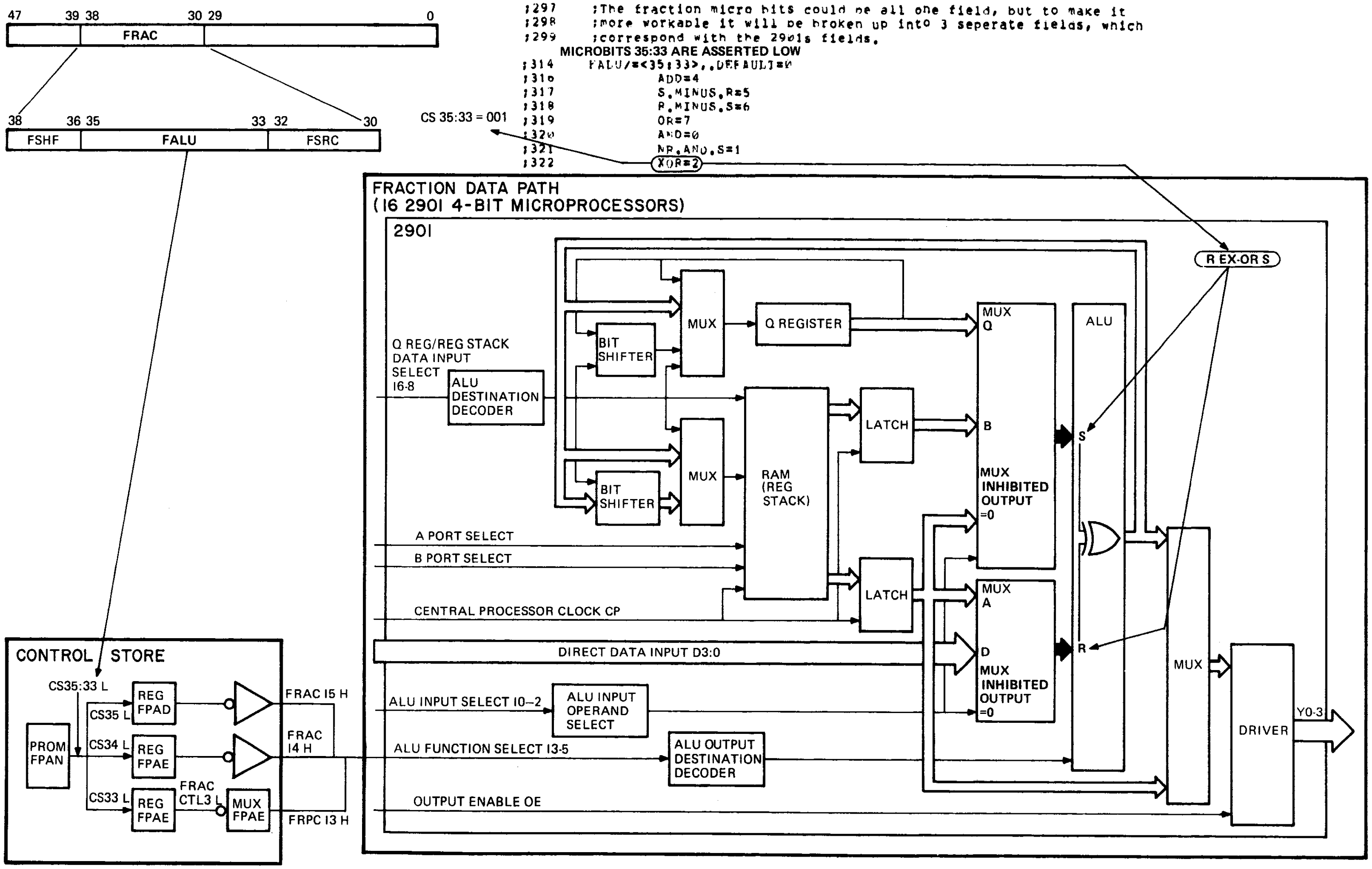
```
        .page

        ;The A address field addresses both the exponent and fraction data path's
        ;scratch pads.  The lower 3 bits also determines what gets clocked into
        ;the resultant sign register.
        FA.ADRS/=<29:26>,,DEFAULT=0

                      ET0=0
                      ET1=1
                      ET2=2
                      ET3=3
                      ET4=4
                      ET5=5              ;Zero constant
                      ET6=6              ;Huge fraction bit count
                      ET7=7              ;Grand fraction bit count
                      ET8=8              ;Double fraction bit count
                      ONE=9              ;Floating fraction bit count
                      ZERO=0A            ;Max huge exponent
                      H.BIAS=0B                ;Max grand exponent
                      FD.MAX=0C          ;Max F,D exponent
                      G.MAX=0D           ;Huge bias
                      G.BIAS=0E          ;Grand bias
                      H.MAX=0F           ;F,D bias

        FA.ADRS/=<29:26>,,DEFAULT=0


                      FT0=0
                      FT1=1
                      FT2=2
                      FT3=3
                      FT4=4
                      FT5=5
                      FT6=6
                      INT.MASK=7
                      FT8=8
                      FT9=9              ;Integer mask Frac15 thru Ext0 eq one.
                      FLT.MASK=0A                ;Floating mask Frac31 thru Ext0 eq one,
                      EXT.MASK=0B                ;Extension mask Ext<7:0>=1's.
                      F.RND=0C                   ;Huge round constant.
                      G.RND=0D                   ;Double round constant.
                      D.RND=0E                   ;Grand round constant.
                      H.RND=0F                   ;Floating round constant.

        SIGN.FUNC/=<28:26>,,VALIDITY=<.EQL[<CLK/>,<CLK/CLK.SIGN.OUT>]>

                      OP1=0              ;Sign out gets 1st operand's sign.
                      ZERO=1             ;Sign out gets 2nd operand's sign.
                      OP1.XOR.OP2=2
                      SU.XOR.OP1=3       ;Resultant sign XOR 1st operand's sign - for poly,
                      OP2=4
                      ONE=5
```

Figure 6-11 RAM A Address Field



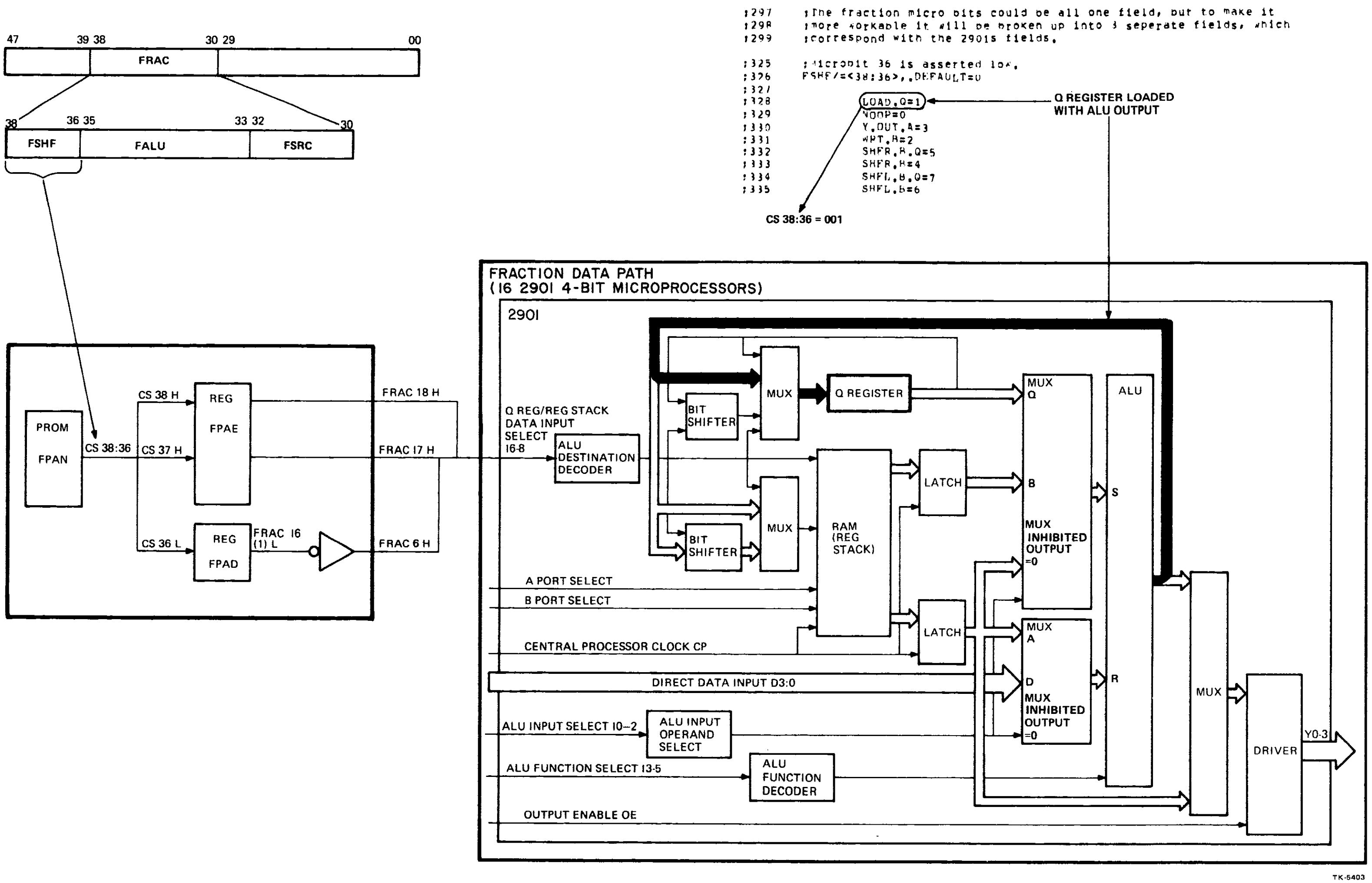
```
;The fraction micro bits could be all one field, but to make it
;more workable it will be broken up into 3 seperate fields, which
;correspond with the 2901s fields.
;These microbits are asserted low.
FSRC/=<32:30>,.DEFAULT=3
           A,Q=7
           0,A=3
           D,A=2
           D,Q=1
           D,0=0
```

Figure 6-12 Fraction ALU Source Operand (DQ) Field



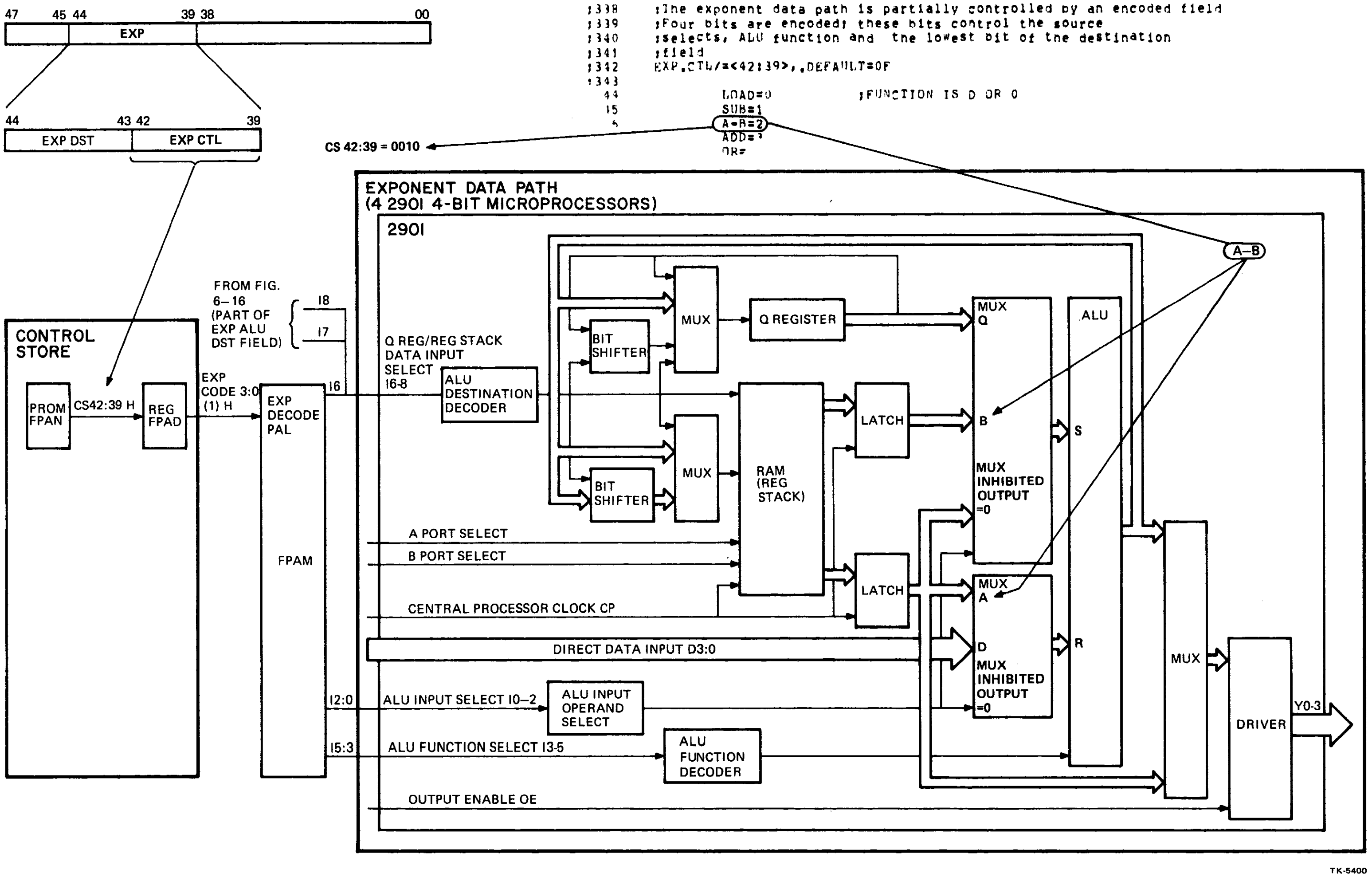
```
;The fraction micro bits could be all one field, but to make it
;more workable it will be broken up into 3 seperate fields, which
;correspond with the 2901s fields.
MICROBITS 35:33 ARE ASSERTED LOW
FALU/=<35:33>,,DEFAULT=0
          ADD=4
          S.MINUS.R=5
          R.MINUS.S=6
          OR=7
          AND=0
          NR.AND.S=1
          XOR=2
```

Figure 6-13 Fraction ALU Function (R XOR S) Field



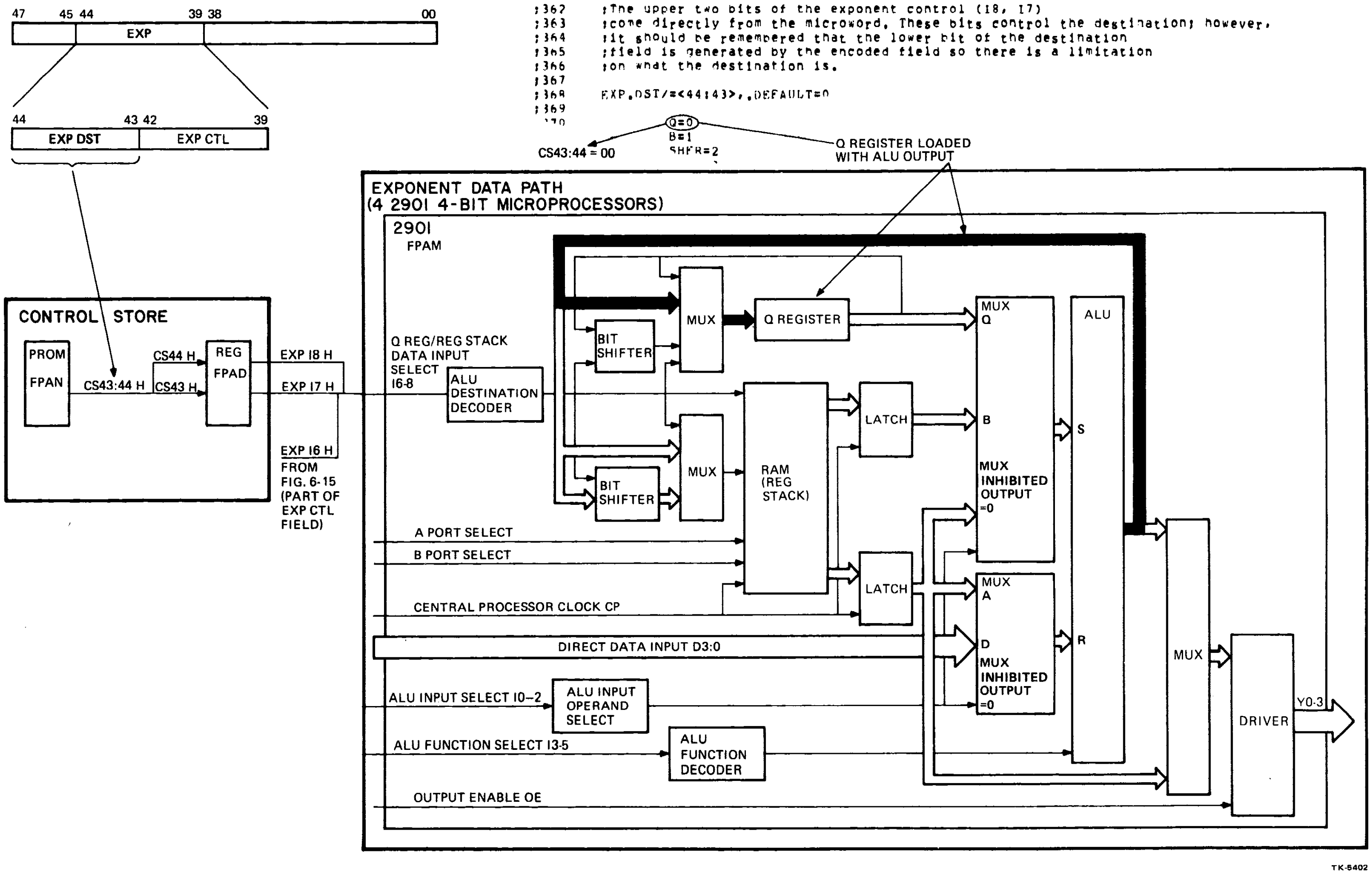
```
;The fraction micro bits could be all one field, but to make it
;more workable it will be broken up into 3 seperate fields, which
;correspond with the 2901s fields.

;Microbit 36 is asserted low.
FSHF/=<38:36>,.DEFAULT=0

                LOAD,Q=1
                NOOP=0
                Y,OUT,A=3
                WRT,B=2
                SHFR,B,Q=5
                SHFR,B=4
                SHFL,B,Q=7
                SHFL,B=6
```

Figure 6-14 Fraction ALU Destination (Q-Register) Control Field



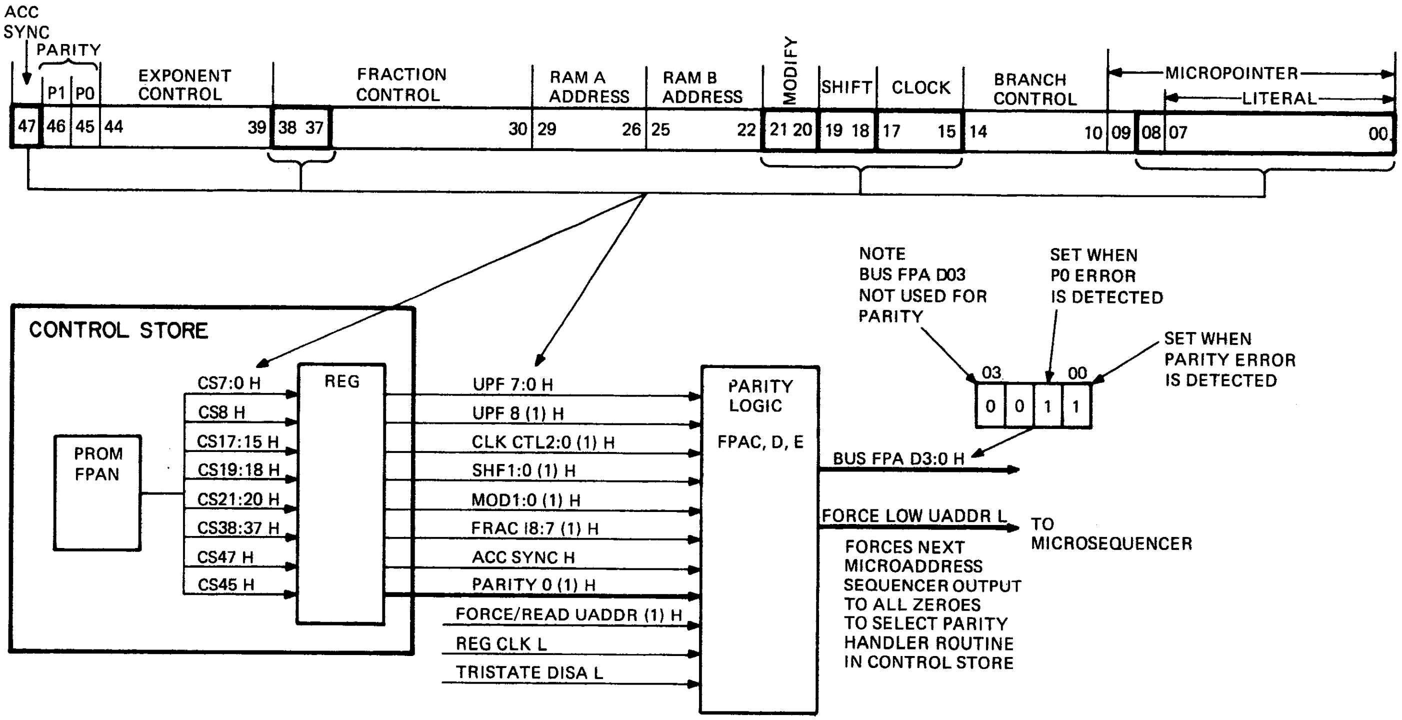
```
;The exponent data path is partially controlled by an encoded field
;Four bits are encoded; these bits control the source
;selects, ALU function and  the lowest bit of the destination
;field
EXP.CTL/=<42:39>,.DEFAULT=0F

        LOAD=0              ;FUNCTION IS D OR 0
        SUB=1
        A-B=2
        ADD=3
        OR=
```

Figure 6-15 Exponent Control (A-B) Field



```
;The upper two bits of the exponent control (18, 17)
;come directly from the microword. These bits control the destination; however,
;it should be remembered that the lower bit of the destination
;field is generated by the encoded field so there is a limitation
;on what the destination is.

EXP.DST/=<44:43>,,DEFAULT=0

        Q=0
        B=1
        SHFR=2
```

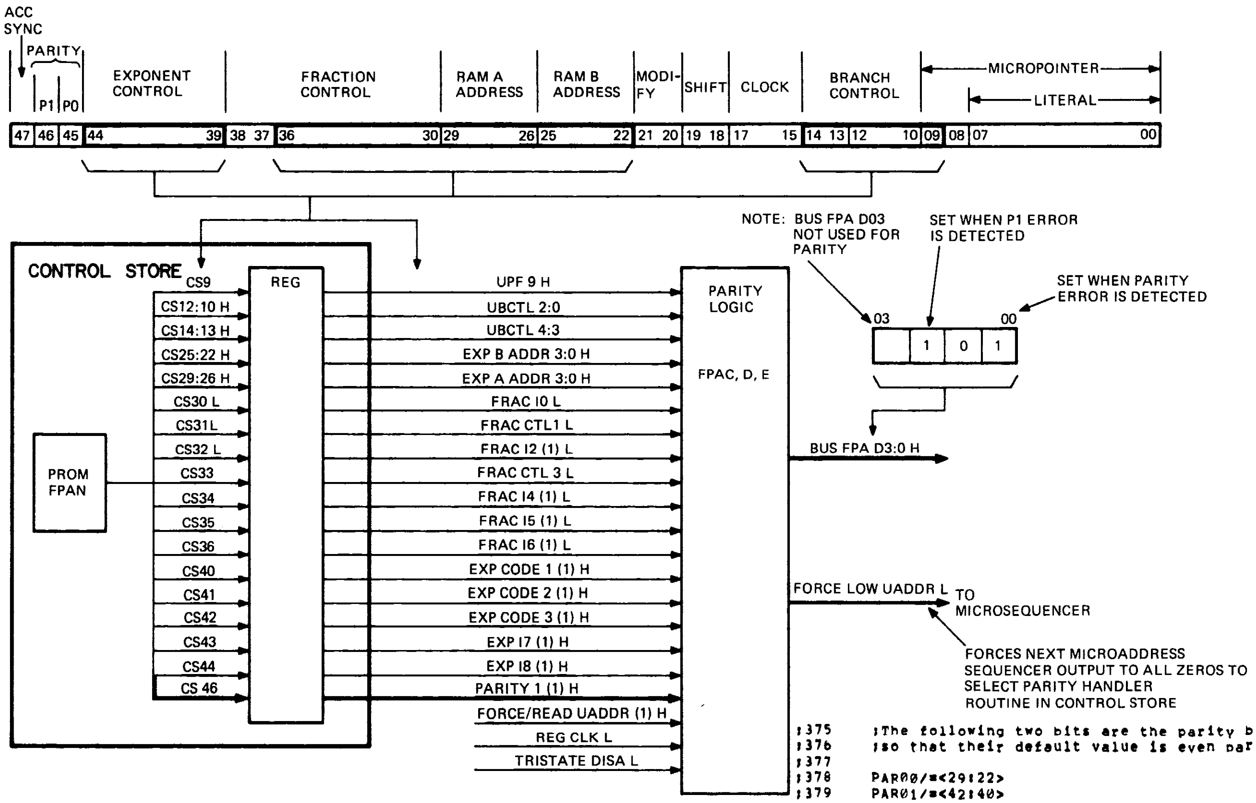
Figure 6-16 Exponent ALU Destination (Q-Register) Control Field



```
;The following two bits are the parity bits; they are defined
;so that their default value is even parity for their given fields.

PAR00/=<29:22>
PAR01/=<42:40>
PAR02/=<9>
.SET/PAR.CK2=<.PARITY[<PAR00/>,<PAR01/>,<PAR02/>]>
PAR10/=<14:13>
PAR11/=<36:30>
PAR12/=<39>
PAR13/=<44:43>
PAR14/=<12:10>
.SET/PAR.CK1=<.PARITY[<PAR10/>,<PAR11/>,<PAR12/>,<PAR13/>,<PAR14/>]>
P1/=<46>,.DEFAULT=<.XOR[PAR.CK2,PAR.CK1]>
PAR20/=<8:0>
PAR21/=<17:15>
PAR22/=<21:18>
PAR23/=<38:37>
PAR24/=<47>
.SET/PARITY0=<.PARITY[<PAR20/>,<PAR21/>,<PAR22/>,<PAR23/>,<PAR24/>]>
P0/=<45>,.DEFAULT=<.NOT[PARITY0]>
```

TK-5401

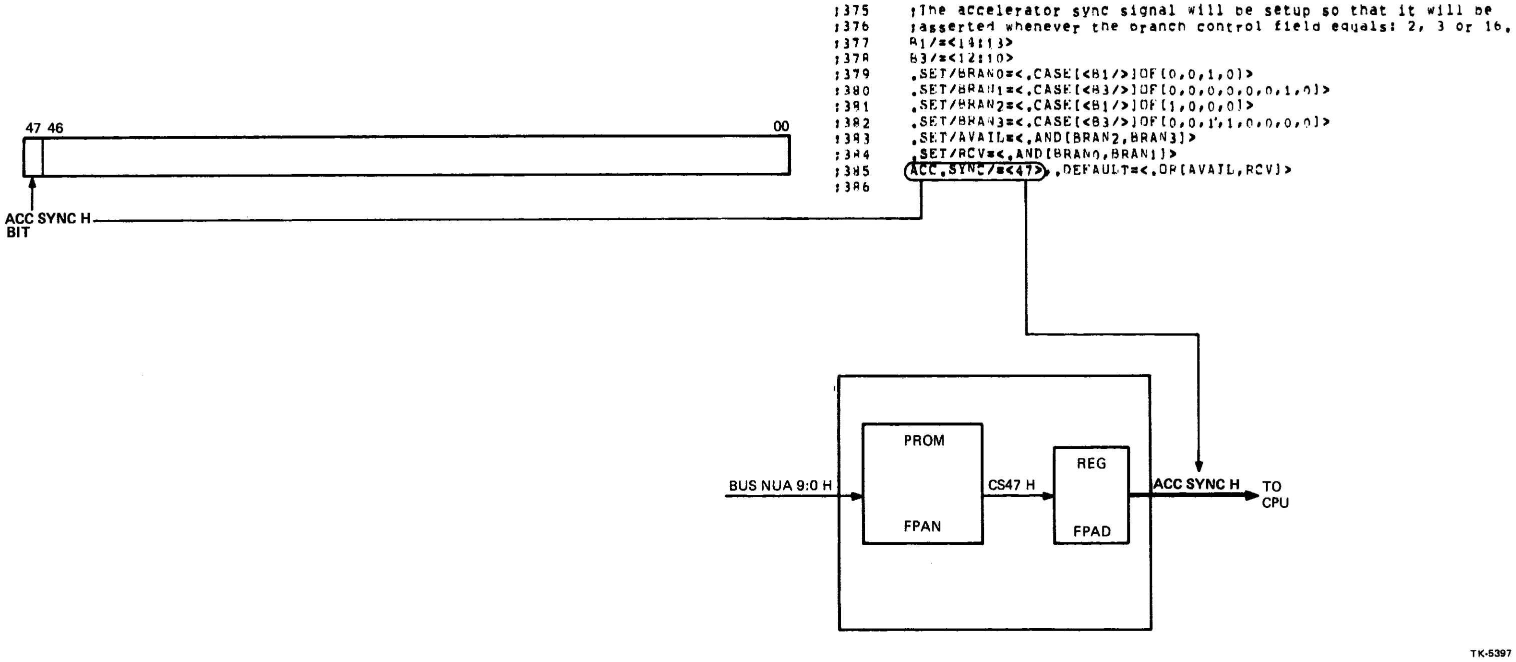
Figure 6-17 Parity Field P0



```
;The following two bits are the parity bits; they are defined
;so that their default value is even parity for their given fields.

PAR00/=<29:22>
PAR01/=<42:40>
PAR02/=<9>
.SET/PAR.CK2=<.PARITY[<PAR00/>,<PAR01/>,<PAR02/>]>
PAR10/=<14:13>
PAR11/=<36:30>
PAR12/=<39>
PAR13/=<44:43>
PAR14/=<12:10>
.SET/PAR.CK1=<.PARITY[<PAR10/>,<PAR11/>,<PAR12/>,<PAR13/>,<PAR14/>]>
P1/=<46>,.DEFAULT=<.XOR[PAR.CK2,PAR.CK1]>
PAR20/=<8:0>
PAR21/=<17:15>
PAR22/=<21:18>
PAR23/=<38:37>
P0/=<45>,.DEFAULT=<.NOT[.PARITY[<PAR20/>,<PAR21/>,<PAR22/>,<PAR23/>]]>
```

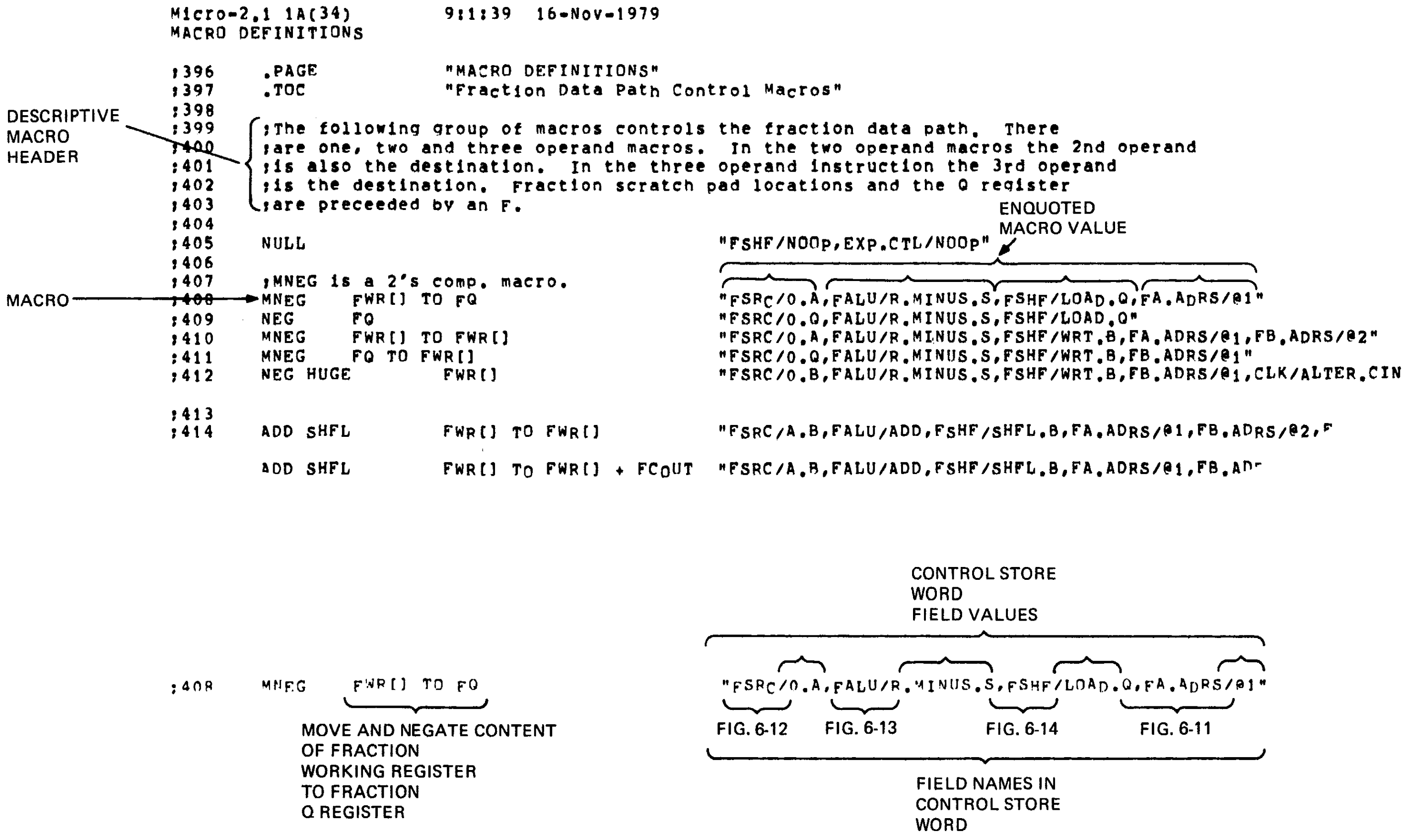
TK-5405

Figure 6-18 Parity Field P1



```
;The accelerator sync signal will be setup so that it will be
;asserted whenever the branch control field equals: 2, 3 or 16,
B1/=<14:13>
B3/=<12:10>
.SET/BRAN0=<.CASE[<B1/>]OF[0,0,1,0]>
.SET/BRAN1=<.CASE[<B3/>]OF[0,0,0,0,0,0,1,0]>
.SET/BRAN2=<.CASE[<B1/>]OF[1,0,0,0]>
.SET/BRAN3=<.CASE[<B3/>]OF[0,0,1,1,0,0,0,0]>
.SET/AVAIL=<.AND[BRAN2,BRAN3]>
.SET/RCV=<.AND[BRAN0,BRAN1]>
ACC.SYNC/=<47>,.DEFAULT=<.OR[AVAIL,RCV]>
```

Figure 6-19 Accelerator Sync Field



```
Micro-2.1 1A(34)        9:1:39  16-Nov-1979
MACRO DEFINITIONS

;396    .PAGE           "MACRO DEFINITIONS"
;397    .TOC            "Fraction Data Path Control Macros"
;398
;399    ;The following group of macros controls the fraction data path.  There
;400    ;are one, two and three operand macros.  In the two operand macros the 2nd operand
;401    ;is also the destination.  In the three operand instruction the 3rd operand
;402    ;is the destination.  Fraction scratch pad locations and the Q register
;403    ;are preceeded by an F.
;404
;405    NULL                                    "FSHF/NOOp,EXp.CTL/NOOp"
;406
;407    ;MNEG is a 2's comp. macro.
;408    MNEG    FWR[] TO FQ                     "FSRC/0.A,FALU/R.MINUS.S,FSHF/LOAD.Q,FA.ADRS/@1"
;409    NEG     FQ                              "FSRC/0.Q,FALU/R.MINUS.S,FSHF/LOAD.Q"
;410    MNEG    FWR[] TO FWR[]                  "FSRC/0.A,FALU/R.MINUS.S,FSHF/WRT.B,FA.ADRS/@1,FB.ADRS/@2"
;411    MNEG    FQ TO FWR[]                     "FSRC/0.Q,FALU/R.MINUS.S,FSHF/WRT.B,FB.ADRS/@1"
;412    NEG HUGE        FWR[]                   "FSRC/0.B,FALU/R.MINUS.S,FSHF/WRT.B,FB.ADRS/@1,CLK/ALTER.CIN

;413
;414    ADD SHFL        FWR[] TO FWR[]          "FSRC/A.B,FALU/ADD,FSHF/SHFL.B,FA.ADRS/@1,FB.ADRS/@2,F

        ADD SHFL        FWR[] TO FWR[] + FCOUT  "FSRC/A.B,FALU/ADD,FSHF/SHFL.B,FA.ADRS/@1,FB.AD
```

```
;408    MNEG    FWR[] TO FQ                     "FSRC/0.A,FALU/R.MINUS.S,FSHF/LOAD.Q,FA.ADRS/@1"
```

DESCRIPTIVE MACRO HEADER
MACRO
ENQUOTED MACRO VALUE
CONTROL STORE WORD FIELD VALUES
MOVE AND NEGATE CONTENT OF FRACTION WORKING REGISTER TO FRACTION Q REGISTER
FIG. 6-12
FIG. 6-13
FIG. 6-14
FIG. 6-11
FIELD NAMES IN CONTROL STORE WORD
TK-5833

Figure 6-20 MACRO Definitions

6.4 MICROROUTINE

Figure 6-21 illustrates an overview of the FPA microcode. The NULL task for the FPA is the wait loop. This microword does nothing except jump to itself. When an IRD signal is issued by the CPU, the FPA will jump to an IRD target as determined by the op code on the IB-Bus and the IRD ROM. The IRD target for instructions not executed by the FPA is the wait loop.

Each instruction class calls either an integer or floating fetch routine, depending on the data type of the operand(s).

After the operand(s) is fetched the instruction will execute. For the floating-point instruction, each instruction class has more than one instruction; the data type and instruction class determine the specific instruction being executed. For each instruction class there is usually one common flow with separate branches for individual data types. For example, ADD F, D, and G have a common flow; ADD H branches away from this common flow because it requires two cycles to add a huge (H) word.

At the end of the execution a store routine is jumped to; the store routine jumped to depends on what data type is being stored.

There are two routines that the CPU forces via the TRAP ACC signal: the initialization and abort routines.

The initialization routine generates a number of constants which are stored permanently in some of the FPA's WRs. This routine is forced upon power up.

The abort routine is forced by the CPU when the CPU must stop execution of the current instruction. The abort sequence sets up some constants for the next instruction and goes to the wait loop.

Figure 6-22 illustrates an ADDition instruction; the ADD flow illustrates the basic flow for all floating arithmetic instructions. The IRD target for ADDX is 201, as shown in the figure. The FET.FLT routine is called from this IRD target. The FET.FLT routine determines the data type, and fetch and appropriate operands. It also sets up some data type depended constants.

Whenever the exponent is loaded in the FET.FLT routine, a flag is set if the exponent is zero; there are two exponent = 0 flags (one for each operand). When the FET.FLT routine is through, it branches on the signal (OP1.AND.OP2) .NE.0.. This branch will OR a one into the LSB of the return address if neither operand is zero. In the case of the ADD instructions, the calling address is 201, the normal return address is 202, and the return address for the case where neither operand is zero, is 203.

If one or both of the operands are zero, a reserved operand check is performed. If neither are reserved operands, then the nonzero operand (or a zero, if both are zero) is moved to the output WR, and the store routine is jumped to.

If neither operand is zero, an execution routine is called; this routine performs all the necessary prealignment shifts, additions and normalization shifts. Then the RND.TST routine is called, (in the case of ADD it is actually jumped to, to save a state) and will round the result and check for overflow or underflow. The RND.TST routine has two return addresses: one address indicates that no exception occurred; the other indicates that an exception did occur.

The two return addresses are generated by ORing a particular status condition into the two LSBs of the return address. In the case of ADDX, the two return addresses are 207 and 204.

The exception return jumps to an exception handler. This routine determines what exception occurred, generates the proper error code, and passes the code to the CPU.

The no exception return sets the condition codes and jumps to the store routine.

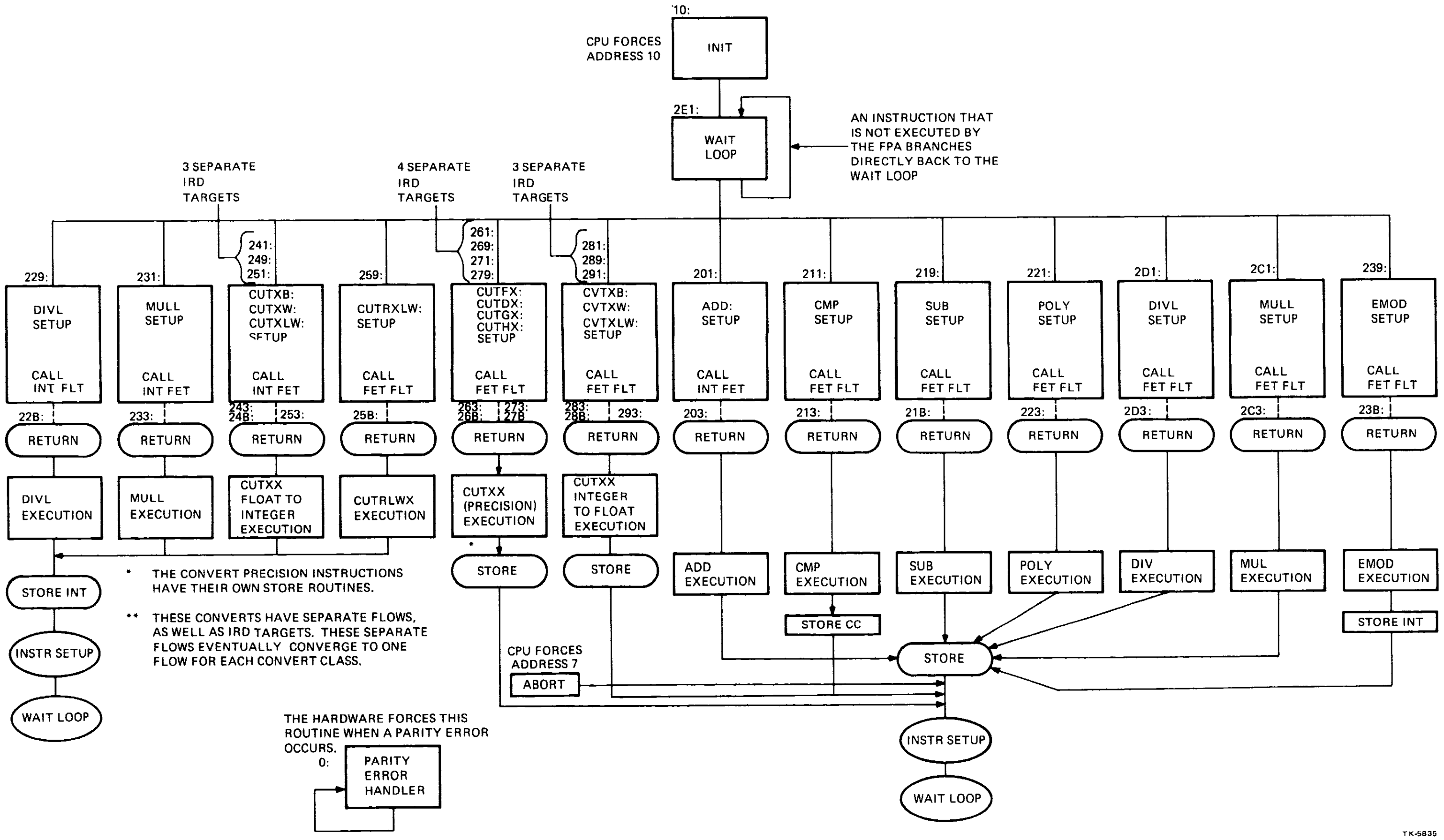


Figure 6-21 Microcode Overview

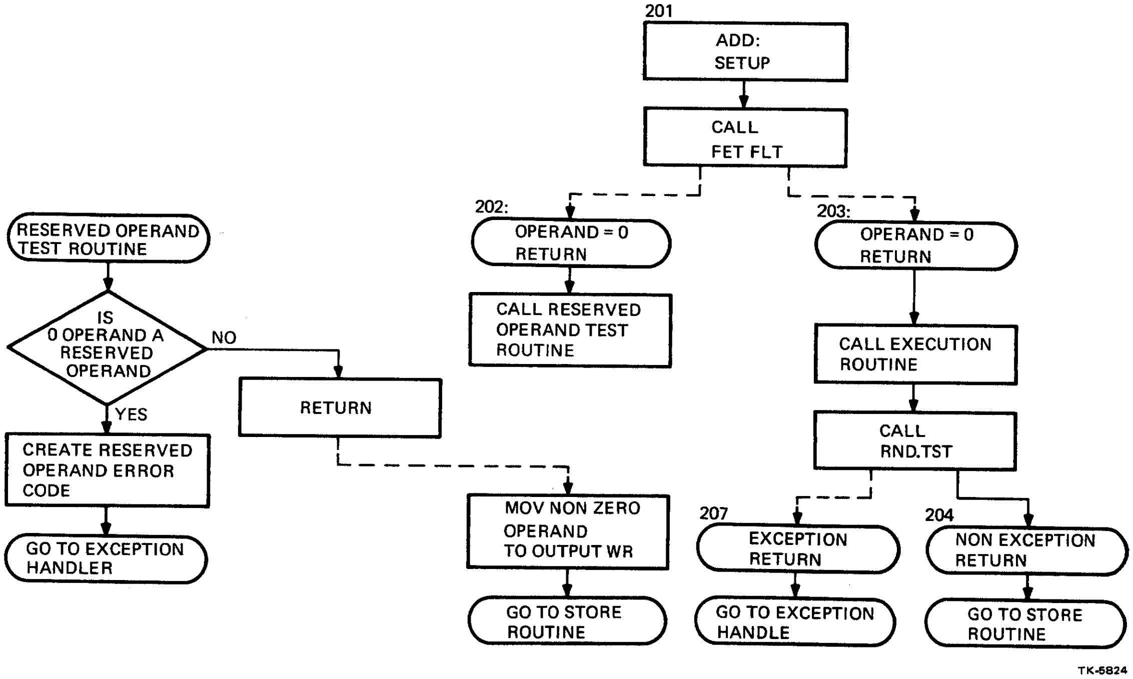


Figure 6-22 Microcode ADD Flow

APPENDIX A
PROGRAMMED ARRAY LOGIC

A.1 INTRODUCTION

Programmed array logic (PAL) devices used in the FPA are logic arrays that contain a programmable AND OR GATE ARRAY comprised of fusable links. Before a PAL is used in the FPA, it is electrically configured and inserted in a PAL programmer that modifies it for particular circuit functions. The programming burns certain links in the array.

Figure A-1 shows the three FPA PAL types and explains the PAL type designator. All three PAL types contain an output circuit (register or inventer) connected to an AND OR GATE ARRAY. The arrays are identical before programming.

NOTE
Additional information on all PALs described in this section can be obtained on microfiche.

Figure A-2 shows AND OR GATE ARRAY details. Figure A-3 shows how fusable links (F1 through F4) in an array can be programmed for a particular function. Figure A-4 illustrates how a particular function (integer division) is enabled for the data shift in control PAL.

A.2 PIN DESIGNATIONS

Figure A-5 illustrates PAL designated (D), input/output (I/O pins are dashed), and register pin (R) designations.

NOTES

1. **A slash (/) indicates signal is asserted low.**

2. **A dash (-) indicates pin has I/O function.**

A.3 PAL FUNCTIONS

Figures A-5 through A-23 illustrate the FPA PALs. The Boolean equations for the PALs can be found on microfiche.

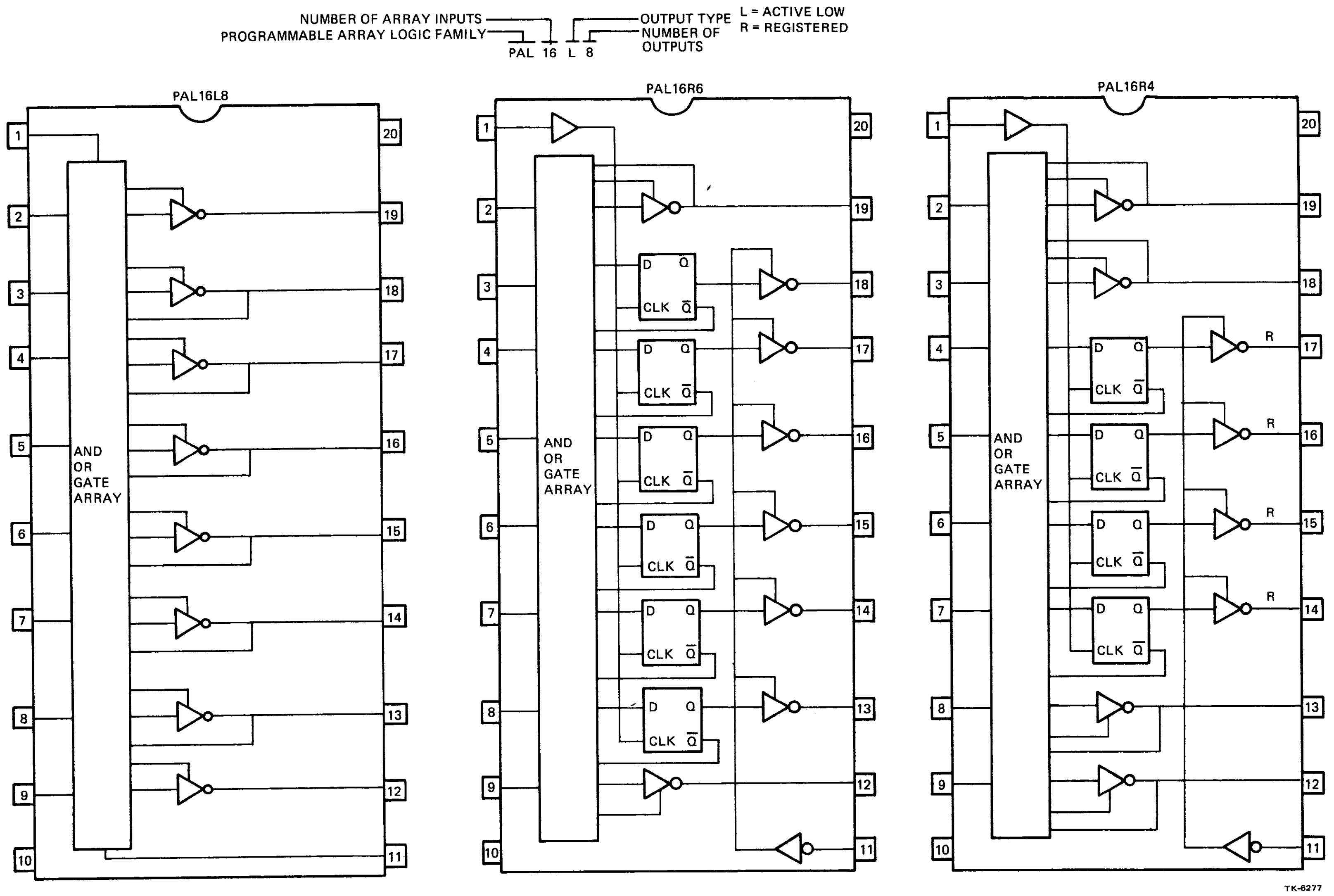


Figure A-1 FPA PAL Types

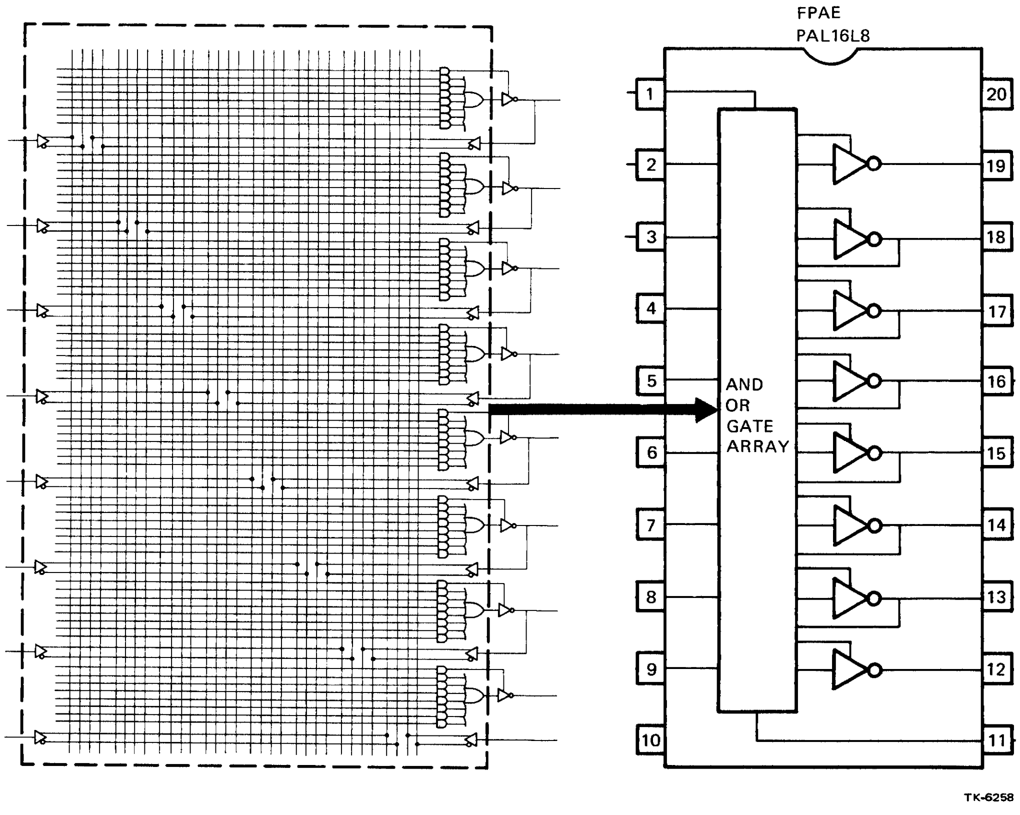
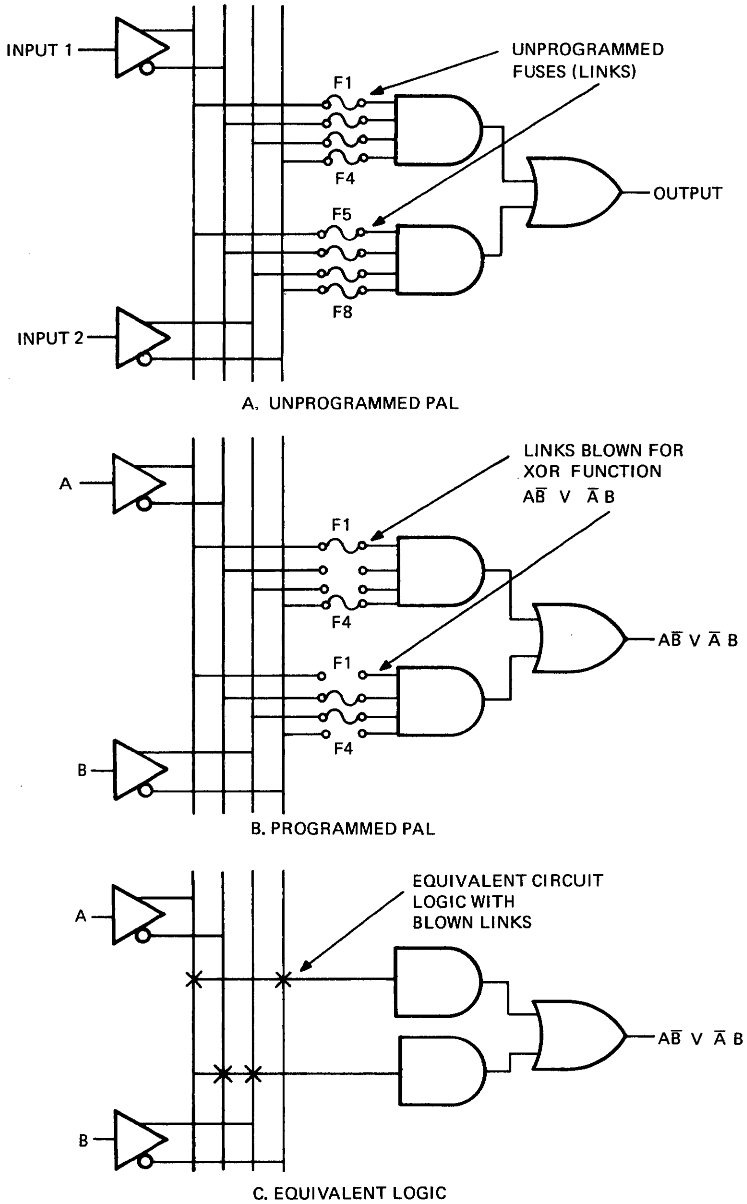


Figure A-2 AND OR GATE ARRAY Details


INPUT 1
UNPROGRAMMED FUSES (LINKS)
F1
F4
OUTPUT
F5
F8
INPUT 2
A
LINKS BLOWN FOR XOR FUNCTION
$A\overline{B}$ V $\overline{A}B$
F1
F4
$A\overline{B}$ V $\overline{A}$ B
F1
F4
B
A
EQUIVALENT CIRCUIT LOGIC WITH BLOWN LINKS
$A\overline{B}$ V $\overline{A}$ B
B

A. UNPROGRAMMED PAL

B. PROGRAMMED PAL

C. EQUIVALENT LOGIC

TK-6255

Figure A-3 Fusable Link Programming

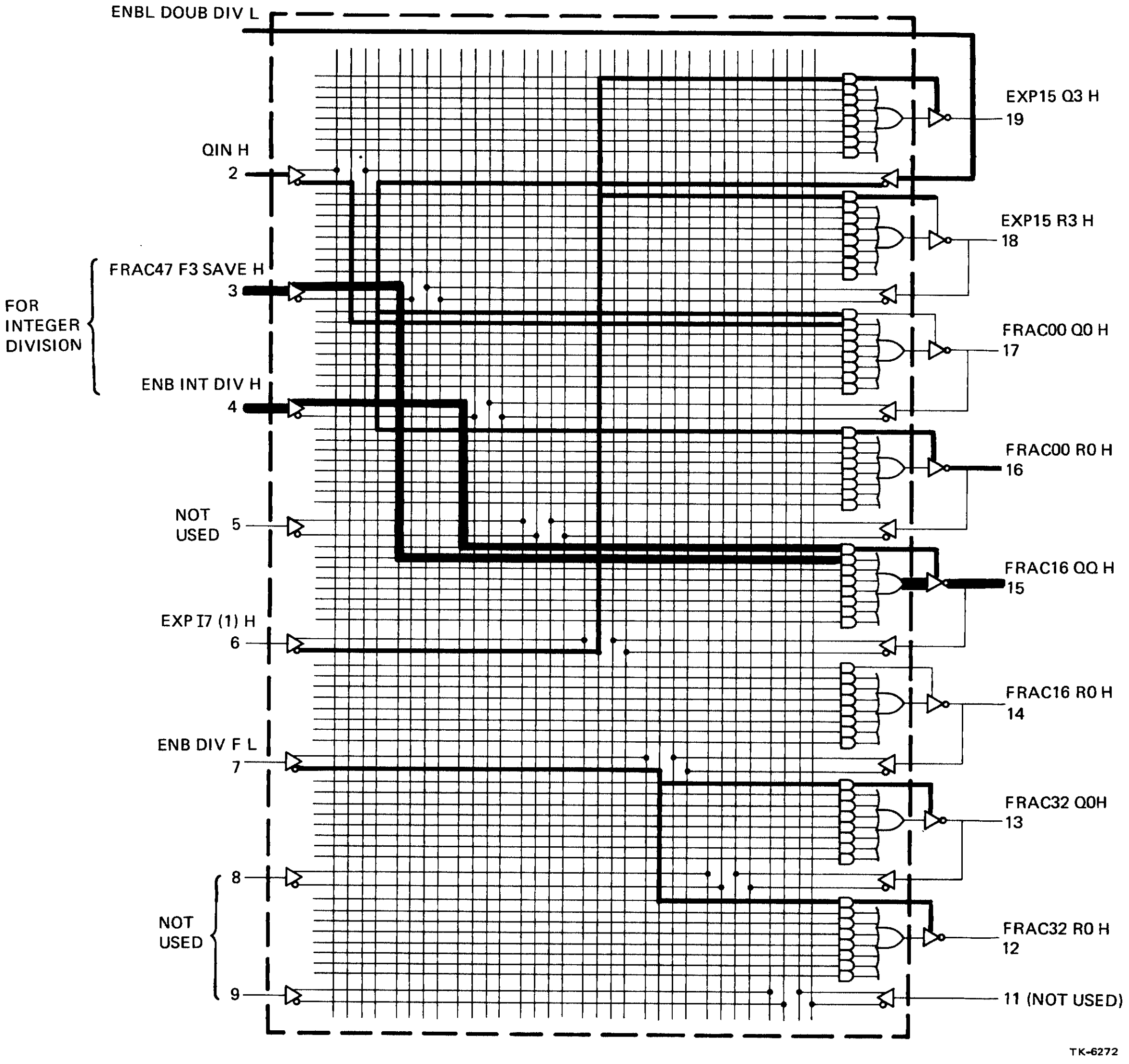
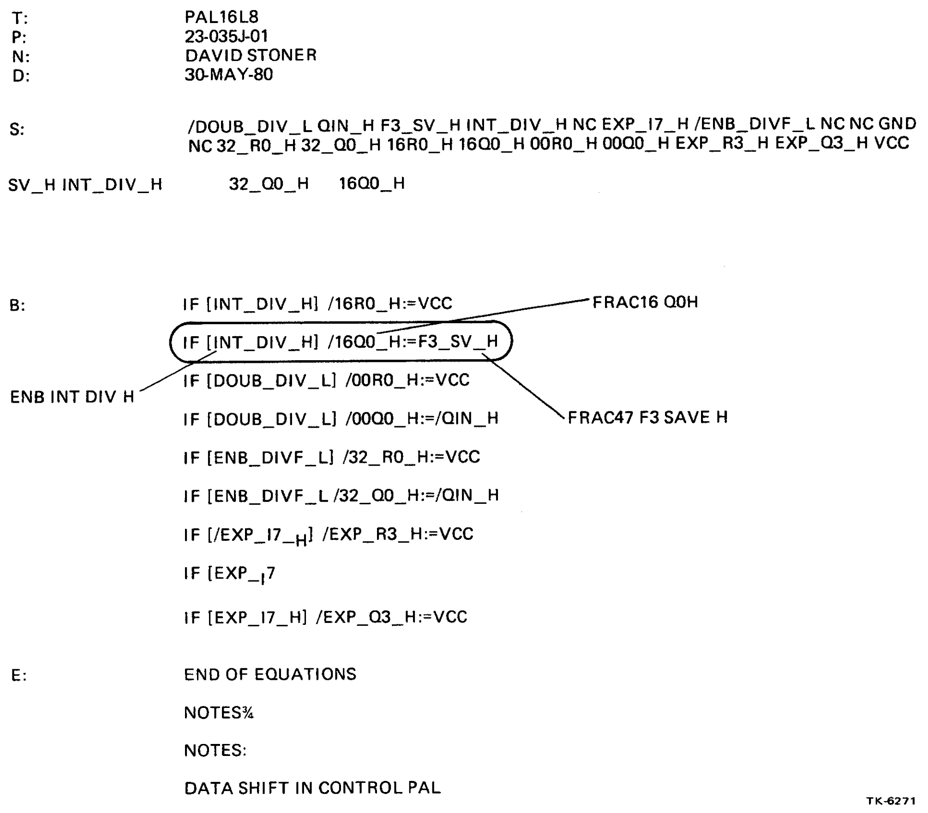
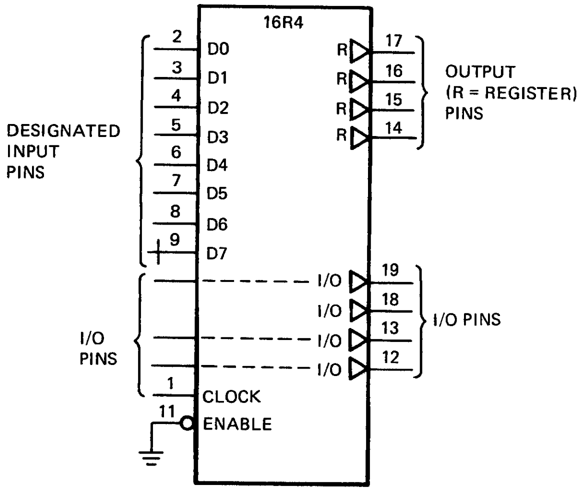


Figure A-4 Integer Division Enabled for Data Shift in PAL
(Sheet 1 of 2)



```
T:          PAL16L8
P:          23-035J-01
N:          DAVID STONER
D:          30-MAY-80

S:          /DOUB_DIV_L QIN_H F3_SV_H INT_DIV_H NC EXP_I7_H /ENB_DIVF_L NC NC GND
            NC 32_R0_H 32_Q0_H 16R0_H 16Q0_H 00R0_H 00Q0_H EXP_R3_H EXP_Q3_H VCC

SV_H INT_DIV_H     32_Q0_H    16Q0_H


B:          IF [INT_DIV_H] /16R0_H:=VCC
            IF [INT_DIV_H] /16Q0_H:=F3_SV_H
            IF [DOUB_DIV_L] /00R0_H:=VCC
            IF [DOUB_DIV_L] /00Q0_H:=/QIN_H
            IF [ENB_DIVF_L] /32_R0_H:=VCC
            IF [ENB_DIVF_L /32_Q0_H:=/QIN_H
            IF [/EXP_I7_H] /EXP_R3_H:=VCC
            IF [EXP_I7
            IF [EXP_I7_H] /EXP_Q3_H:=VCC

E:          END OF EQUATIONS
            NOTES¾
            NOTES:
            DATA SHIFT IN CONTROL PAL
```

Figure A-4 Integer Division Enabled for Data Shift in PAL (Sheet 2 of 2)

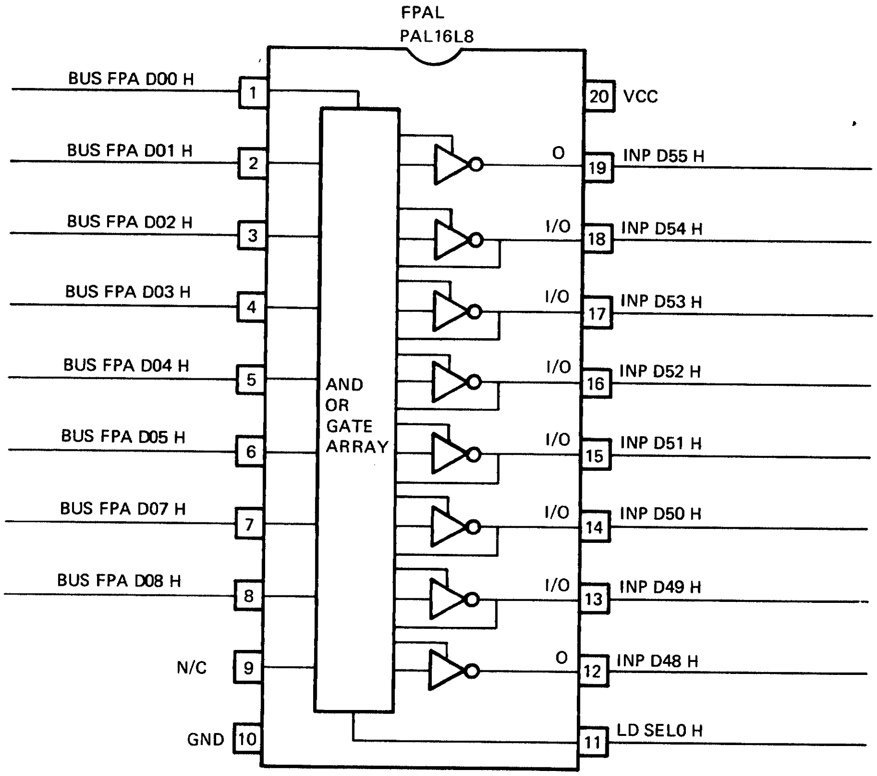


NOTES: 1. SLASH(1) INDICATES SIGNAL IS ASSERTED LOW

2. DASH (–) INDICATES PIN HAS I/O FUNCTION

TK-6254

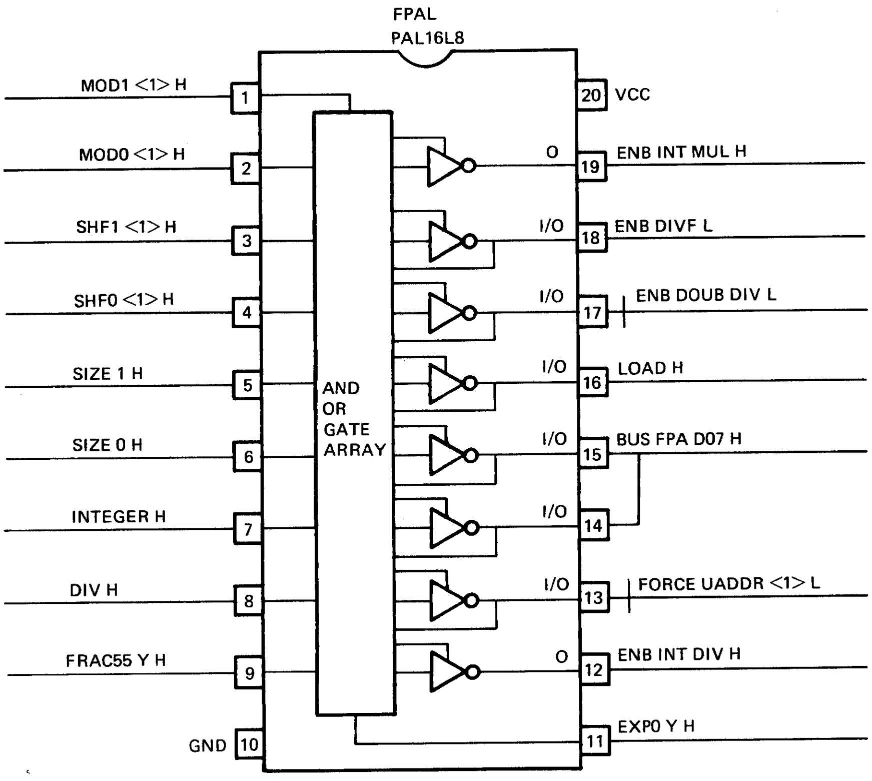
Figure A-5 Pin Designations



THIS PAL SERVES AS A MUX TO DIRECT THE HIDDEN BIT TO THE CORRECT BIT POSITION AS DETERMINED BY THE DATA SIZE.

TK-6264

Figure A-6 Hidden Bit PAL



THIS PAL ENABLES VARIOUS DRIVERS WHICH DRIVE SOME OF THE RAM3–RAM0 AND Q3–Q0 BUSES FOR MULTIPLY AND DIVIDE.

TK-6263

Figure A-7 Input Enable PAL

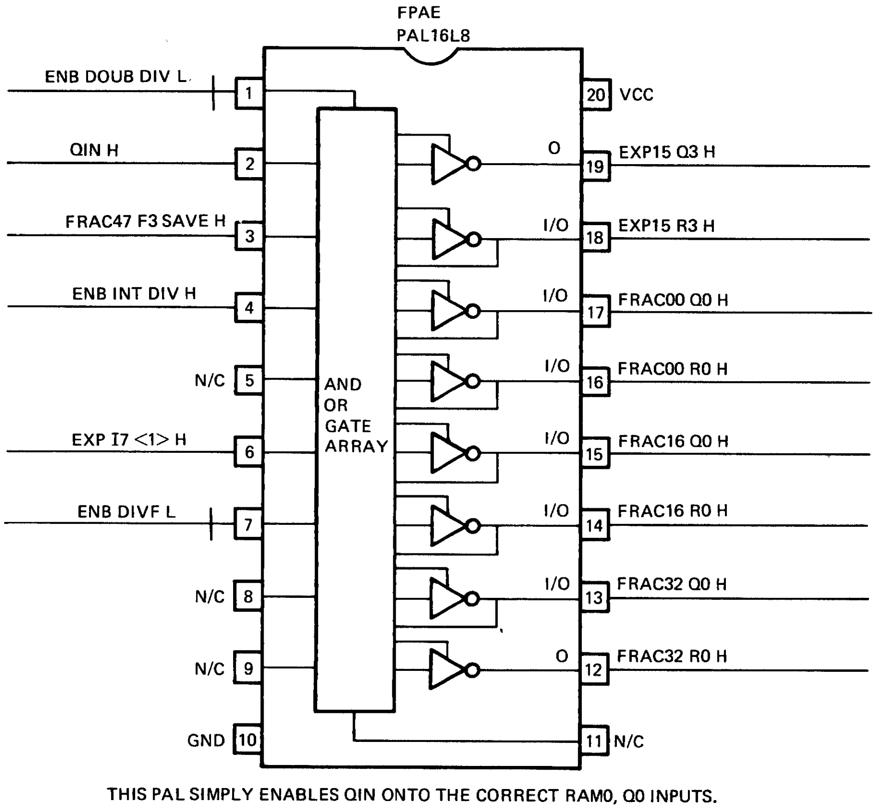
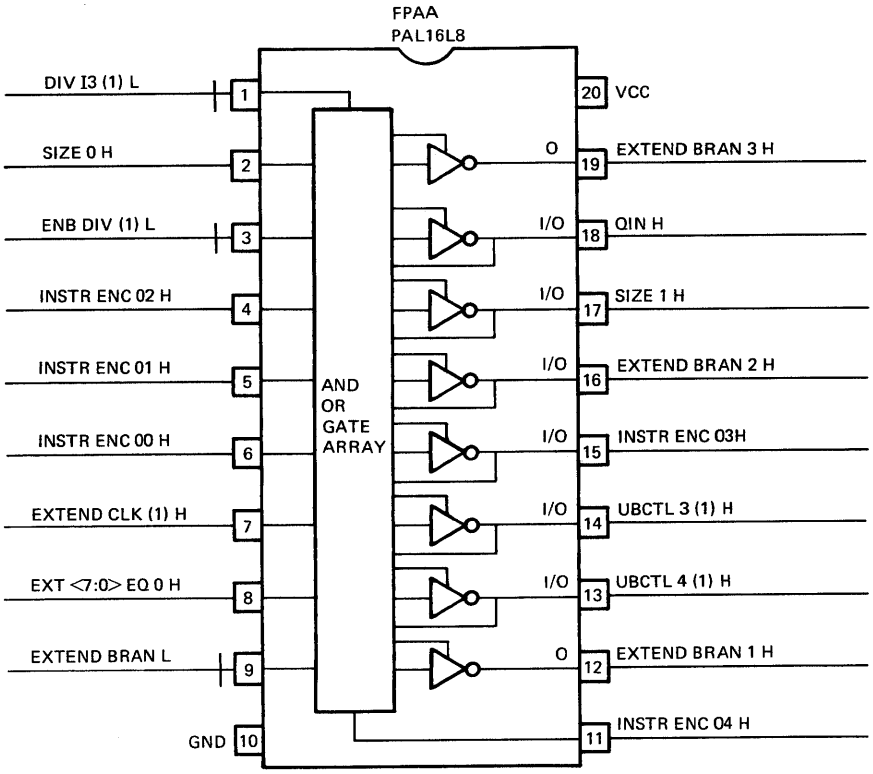


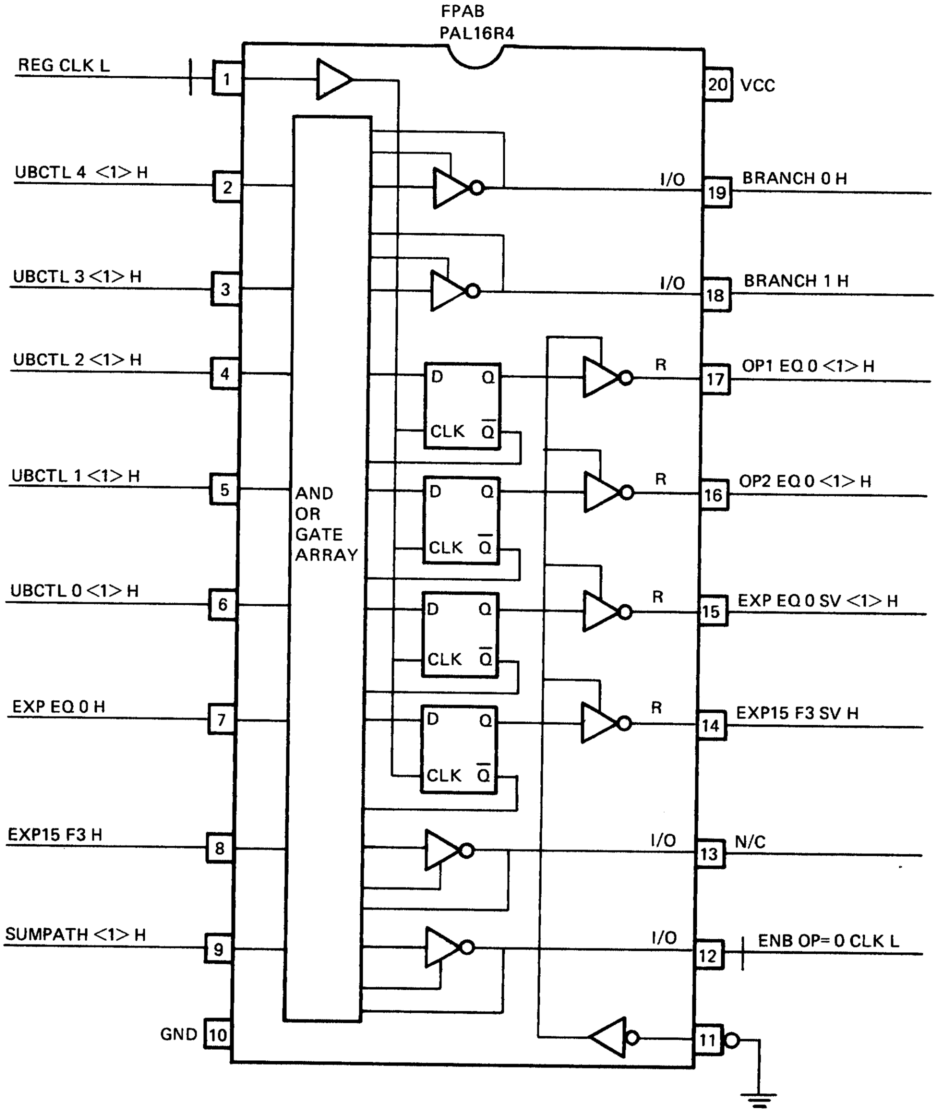
Figure A-8 Data Shift in PAL



THIS PAL GENERATES THREE OUTPUT SIGNALS.

TK-6270

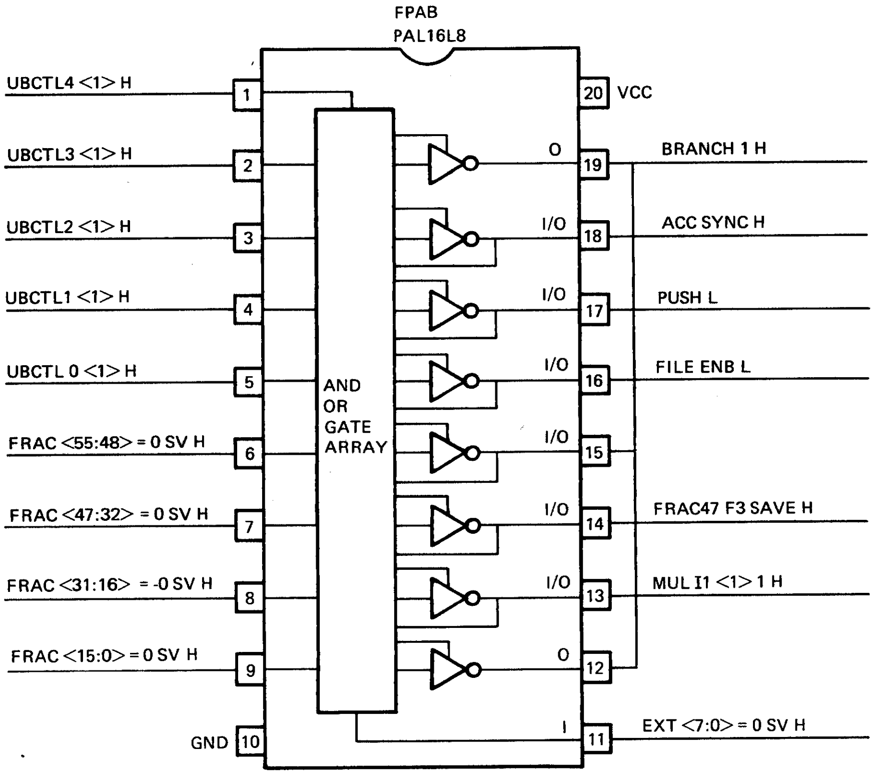
Figure A-9 Extended Branch PAL



THIS PAL GENERATES BOTH LOWER BRANCH BITS; IT ALSO LATCHES
A NUMBER OF STATUS SIGNALS.

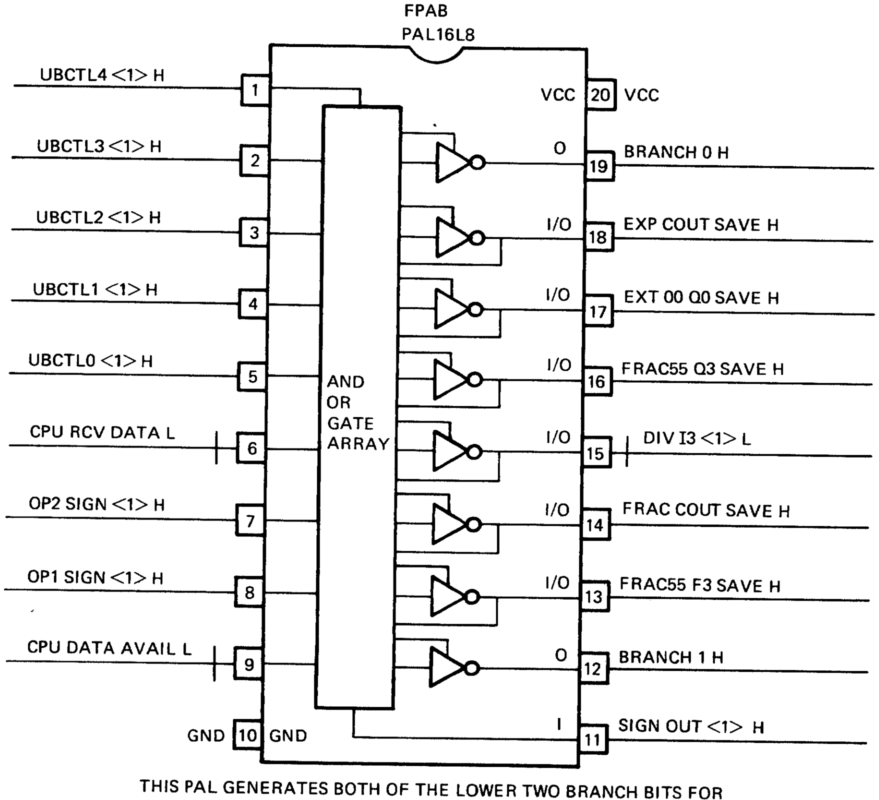
TK-6275

Figure A-10 Branch 3 PAL



THIS PAL GENERATES THE BRANCH 1 SIGNAL.

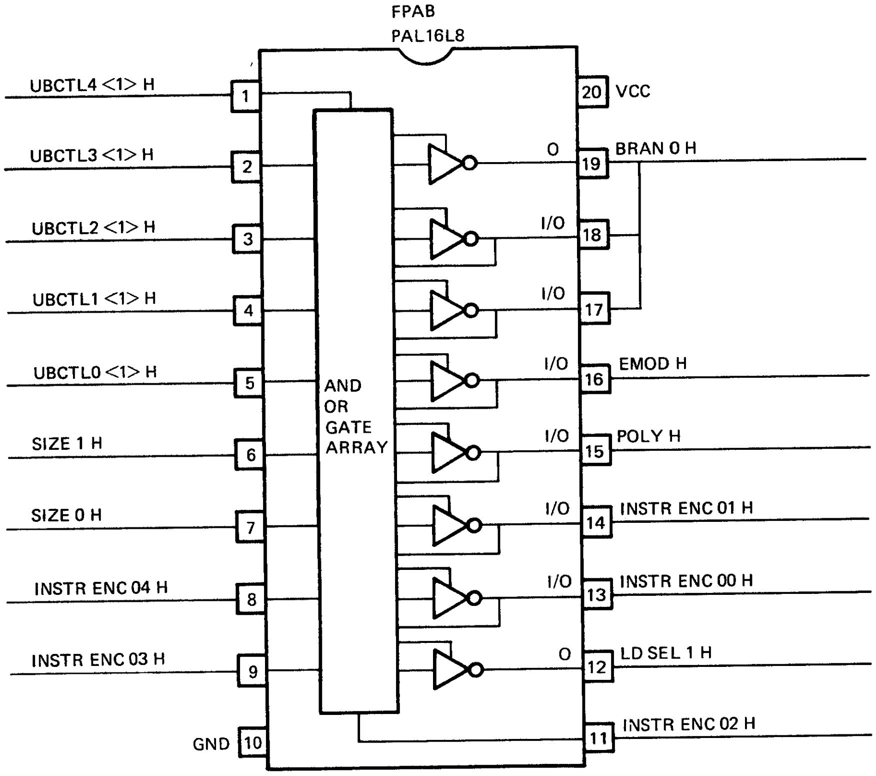
Figure A-11 Branch 2 PAL



THIS PAL GENERATES BOTH OF THE LOWER TWO BRANCH BITS FOR CERTAIN UBCTL VALUES.

TK-6262

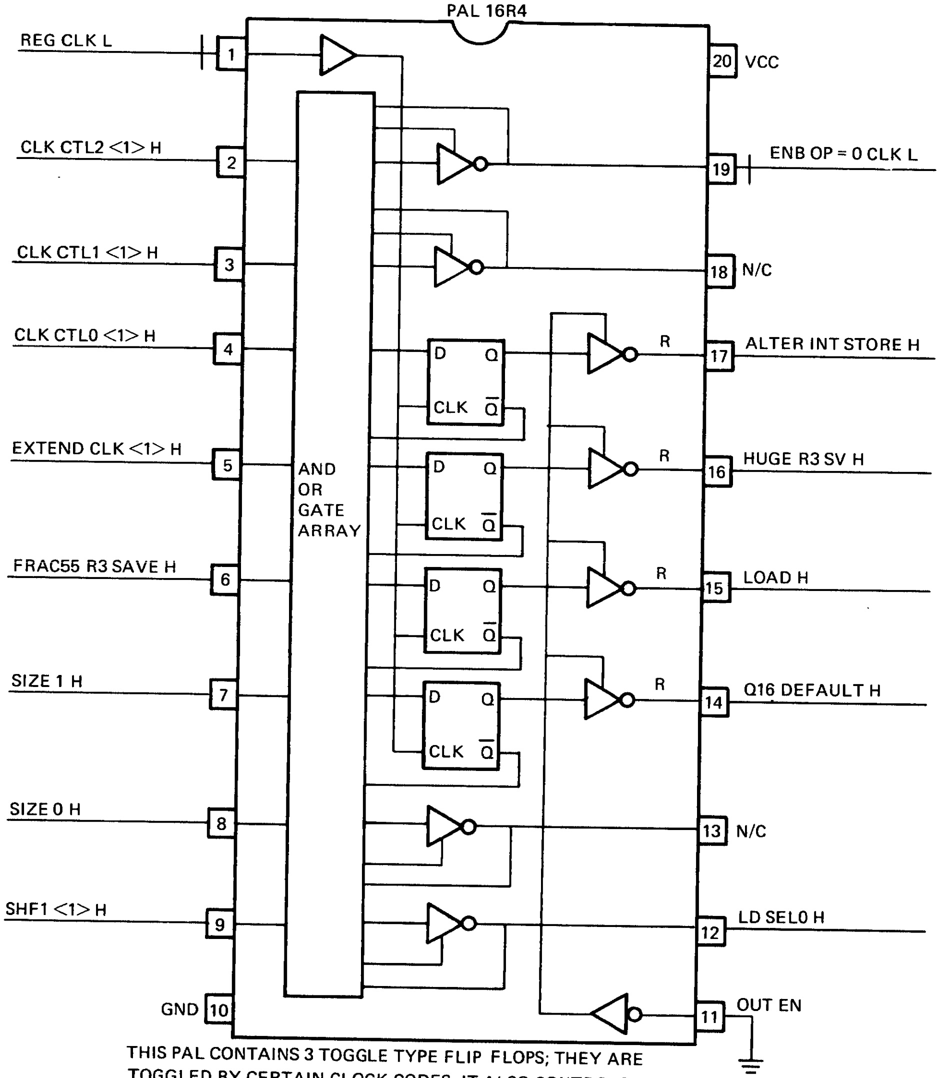
Figure A-12 Branch 1 PAL



THIS PAL WILL GENERATE THE LOWEST BRANCH BIT FOR THOSE UBCTL FIELD WHOSE UPPER TWO BITS ARE 0.

TK-6265

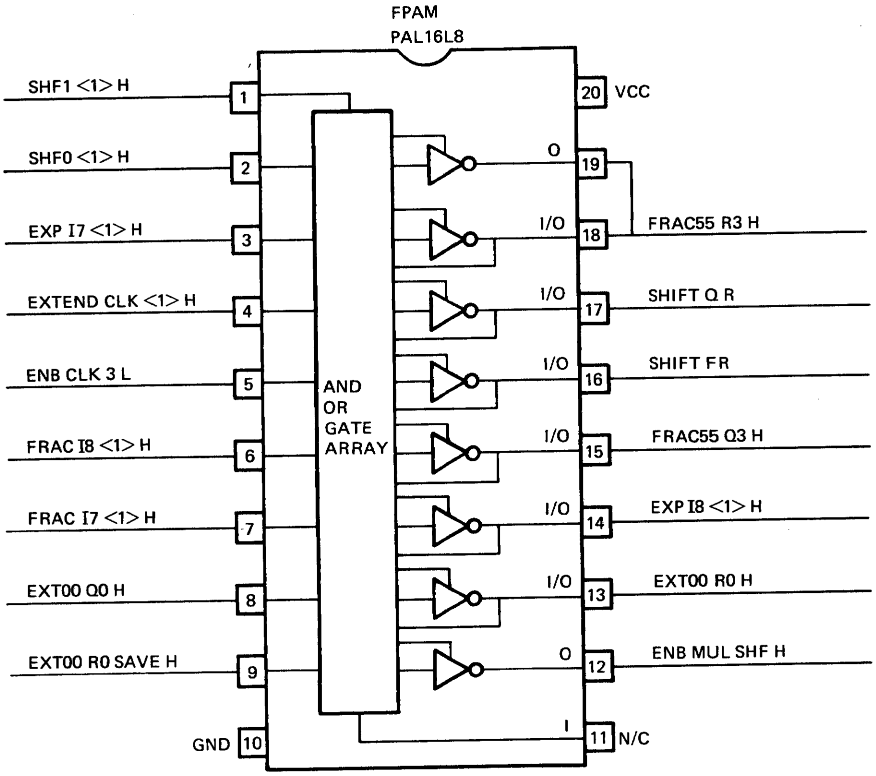
Figure A-13 Branch 0 PAL



THIS PAL CONTAINS 3 TOGGLE TYPE FLIP FLOPS; THEY ARE TOGGLED BY CERTAIN CLOCK CODES. IT ALSO CONTROLS THE DATA IN PAL AND THE CLOCK OF THE OP1=0 AND OP2=0 FLAGS.

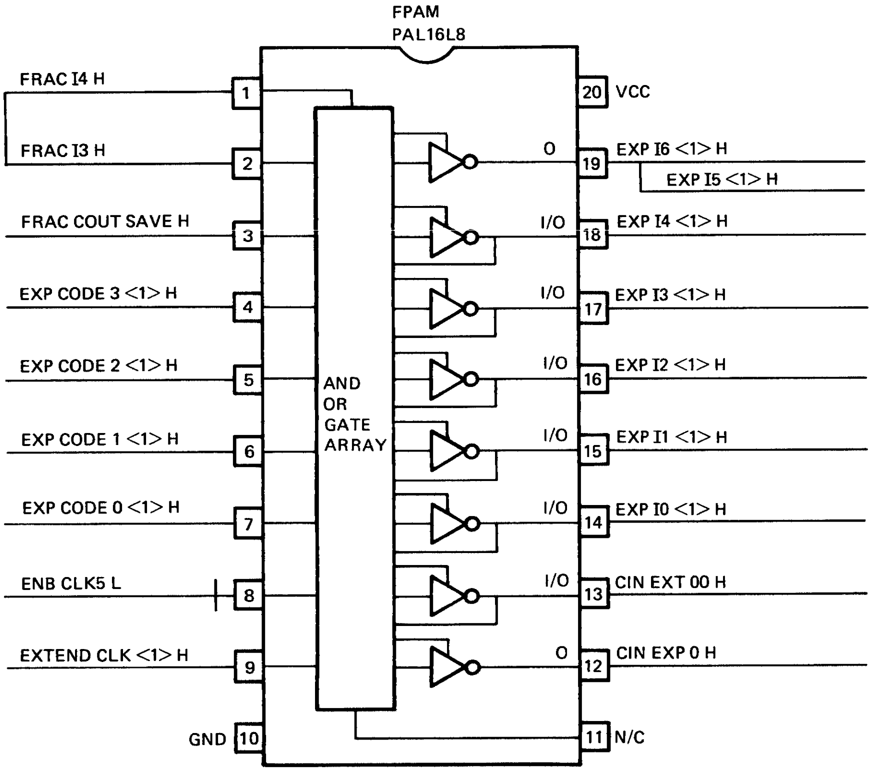
TK-6274

Figure A-14 Extended Function PAL



THIS PAL CONTROLS WHAT IS SHIFTED INTO THE MSBs OF THE FRACTION DATA PATH (RAM 3, Q3), AND WHAT IS SHIFTED INTO THE LSBs OF THE EXPONENT DATA PATH (FAM0, Q0).

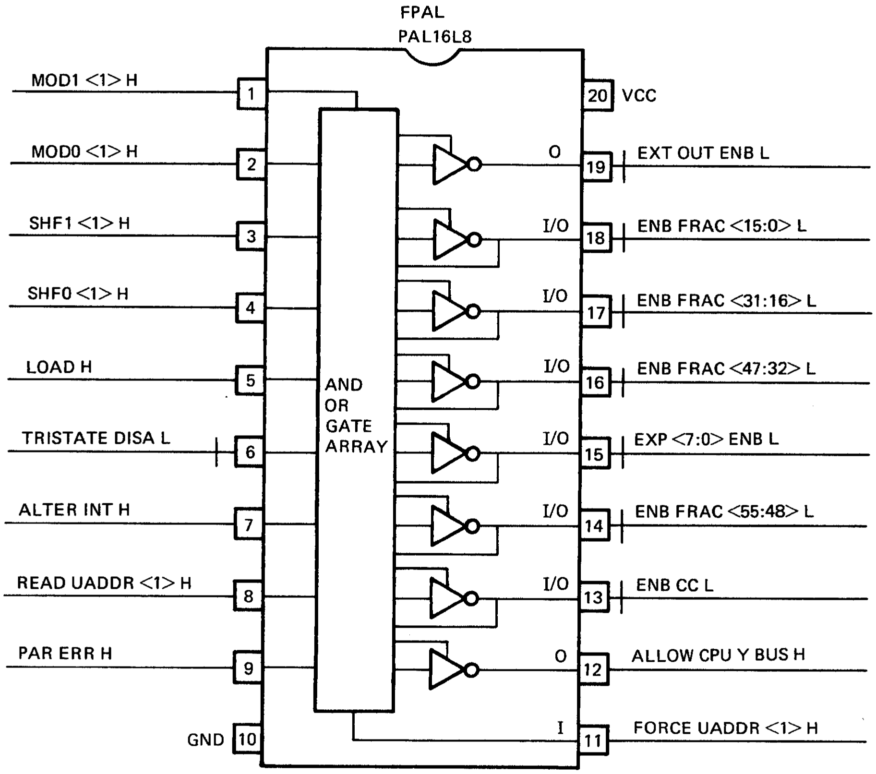
Figure A-15 Fraction Shift Control PAL



THE EXPONENT CONTROL PAL DECODES A MICROFIELD 4 BITS WIDE TO CONTROL EXP 16–0. THE PAL MAPS THE 4 BIT FIELD INTO A 7 BIT FIELD.

TK-6267

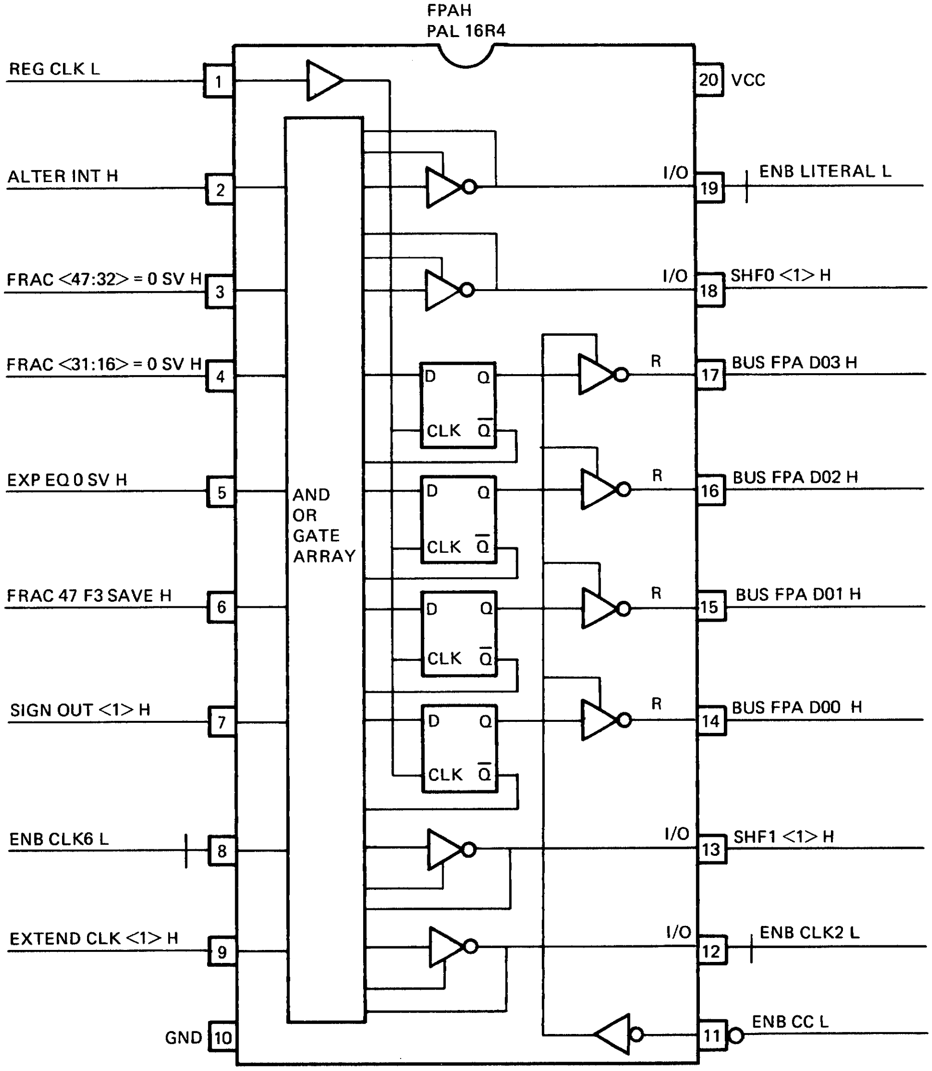
Figure A-16 Exponent Control PAL



THIS PAL ENABLES THE SELECTED BIT SLICE GROUP ONTO THE BUS FPA DURING A STORE OPERATION.

TK-6257

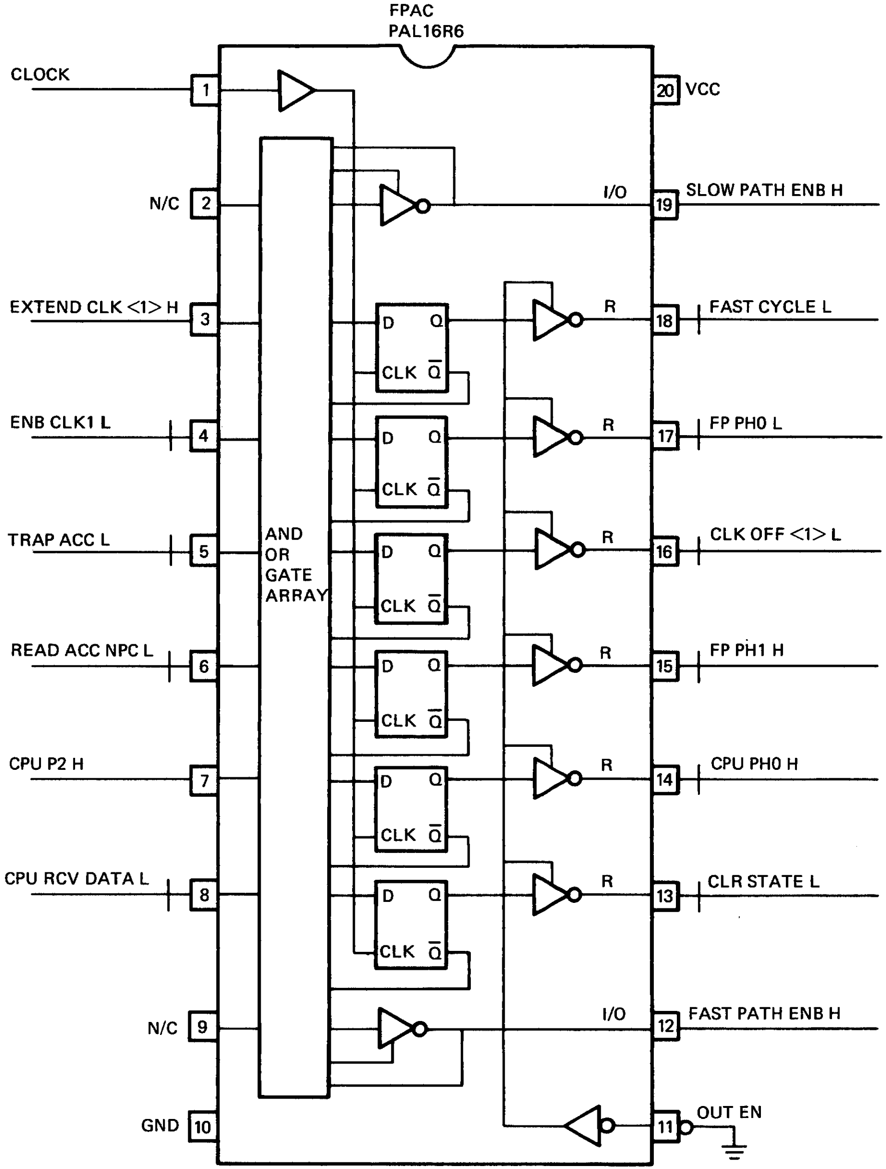
Figure A-17 Store Control PAL



THIS PAL STORES THE CONDITION CODES, WHICH WILL BE PASSED TO THE CPU. CC BITS N AND Z ARE SET ACCORDING TO VARIOUS STATUS CONDITIONS; CC BITS C AND V ARE EXPLICITLY SET BY THE MICROCODE AS ERROR FLAGS TO THE CPU. THE PAL ALSO GENERATES THE LITERAL ENABLE.

TK-6276

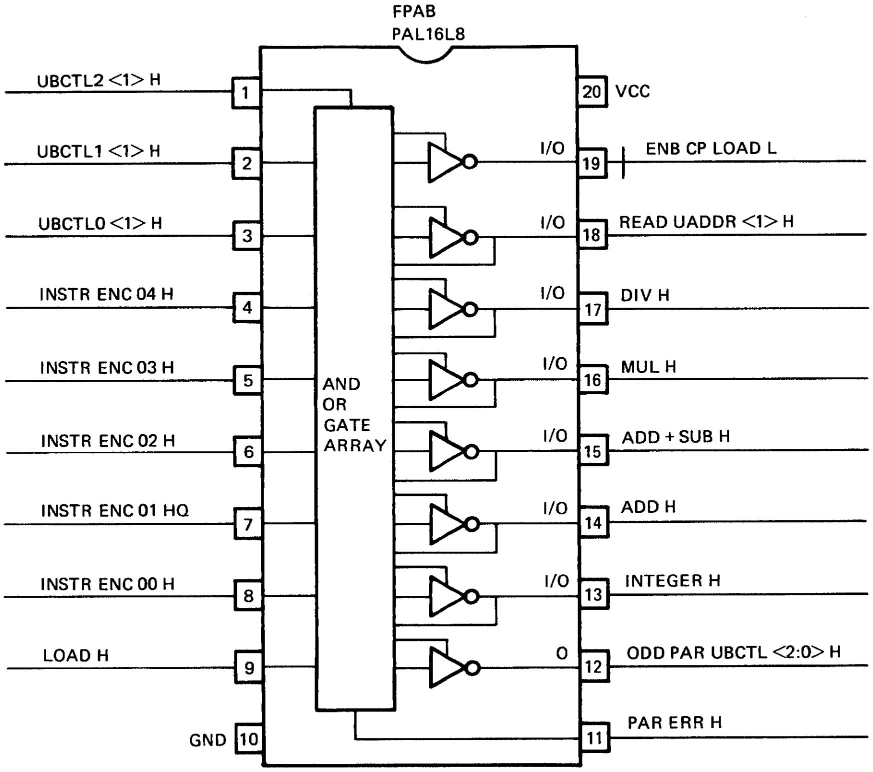
Figure A-18 Condition Code PAL



THE CLOCK PAL CONTROLS THE CLOCKS FOR THE FPA; IT WILL ENABLE THE CPU TO CLOCK THE FPA IF FAST IS NOT SET, OTHERWISE THE FPA WILL GENERATE ITS OWN CLOCKS.

TK-6253

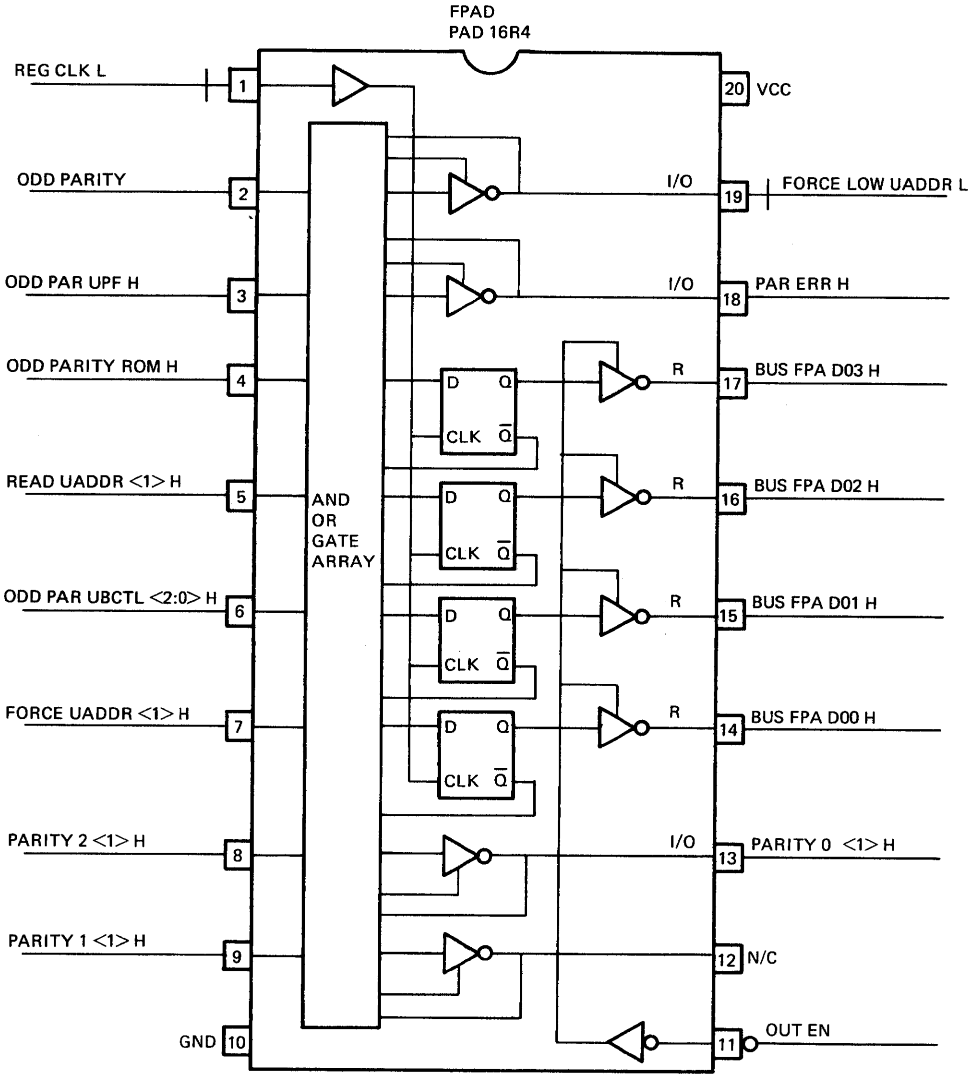
Figure A-19 Clock Control PAL



THIS INSTRUCTION PAL GENERATES A NUMBER OF INSTRUCTION SPECIFIC SIGNALS NEEDED FOR CONTROL AND BRANCHES.

TK-6256

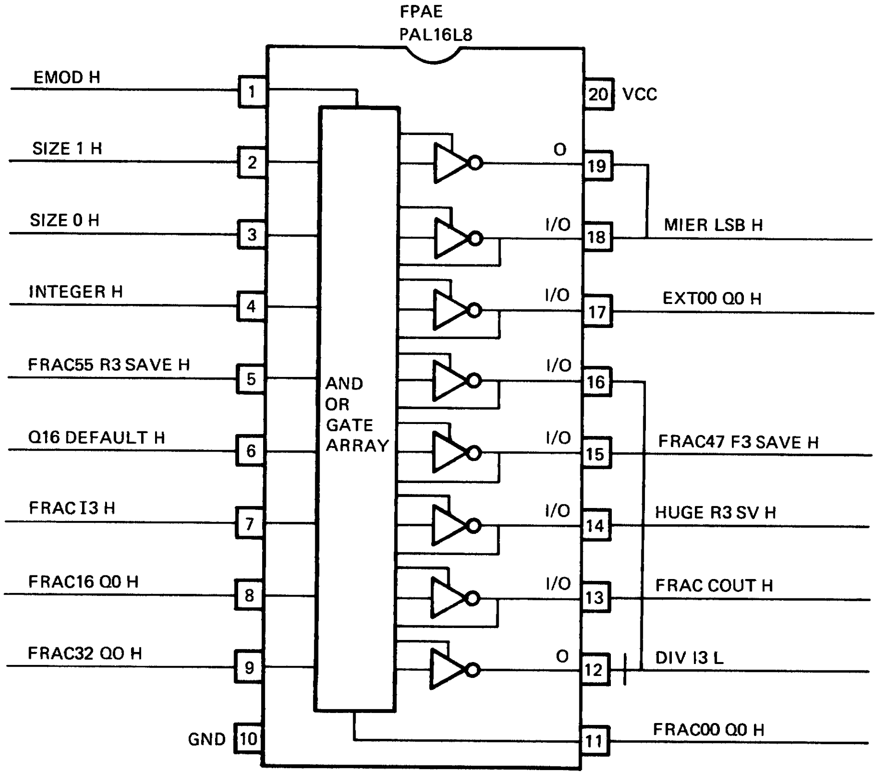
Figure A-20 Instruction PAL



THE PARITY PAL CHECKS THE 2 GROUPS OF MICROBITS FOR A PARITY ERROR. IF ONE IS FOUND, A FLAG IS SET TO INDICATE WHAT PARITY ERROR OCCURED. ONCE THIS IS DONE MICROADDRESS ZERO IS FORCED. THIS MICROWORD WILL LOOP ON ITSELF, CONSTANTLY PLACING THE PARITY ERROR ON THE BUS FPA; BUS FPA D00 IS THE OR OF THE THREE PARITY BITS.

TK-6261

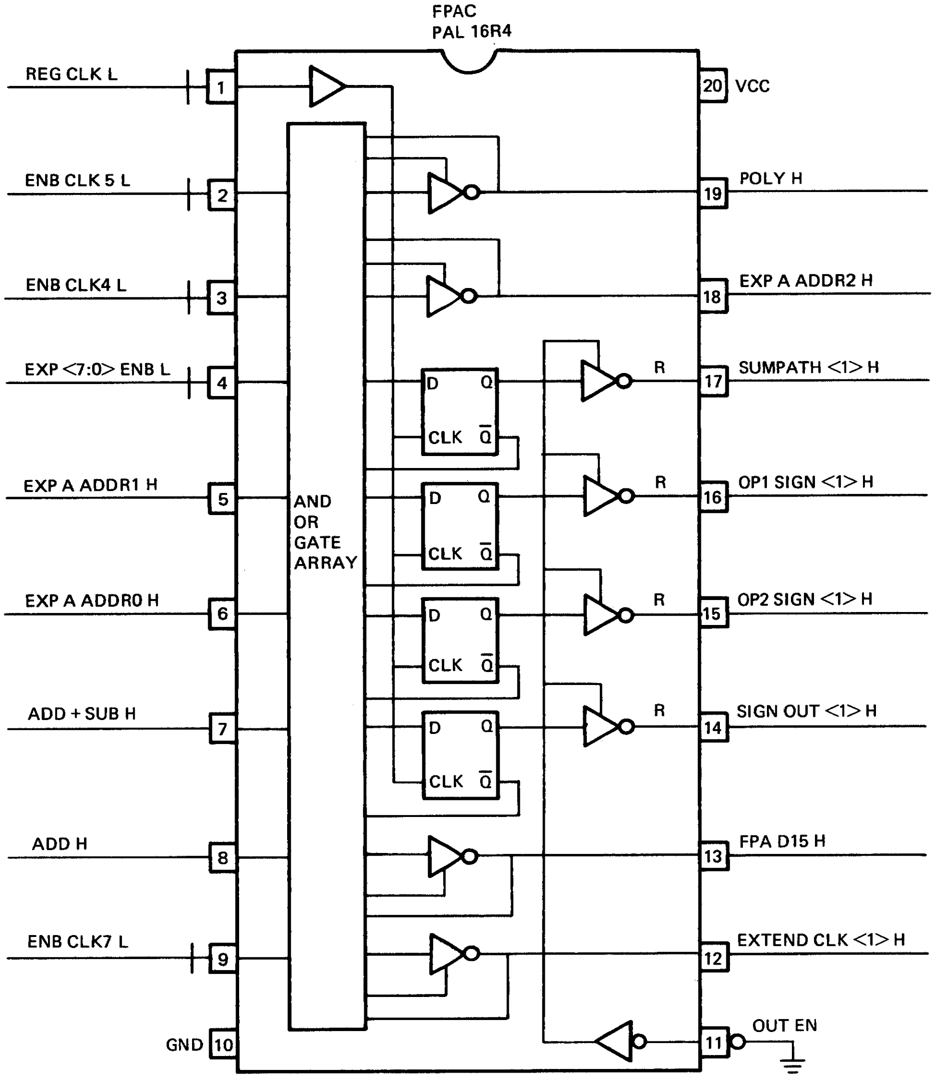
Figure A-21 Parity PAL



THIS PAL PERFORMS THE CONDITIONAL CONTROL FOR BOTH MULTIPLY AND DIVIDE.

TK-6259

Figure A-22 Multiply/Divide PAL



THIS PAL STORES THE SIGN OF BOTH OPERANDS, THE RESULTANT SIGN AND A SIGNAL CALLED SUMPATH, WHICH INDICATES WHETHER A SUM OR DIFFERENCE IS TO BE EXECUTED FOR THE ADD AND SUBTRACT INSTRUCTIONS.

TK-6260

Figure A-23 Sign PAL

APPENDIX B
GLOSSARY

Algorithm

Set of processes (procedure) FPA performs to solve a floating-point problem in a finite number of steps.

ACC

Accelerator.

ACC SYNC

Accelerator synchronization bit (CS47, Figure 6-19) asserted whenever branch control field (CS14:10, Figure 6-4) equals 2, 3, or 16. ACC SYNC H indicates to CPU that FPA is ready.

ALU

Arithmetic logic unit contained in data path logic and in microaddress sequencer.

Bias

Excess notation.

Branch Control
Field

Five-bit field (CS14:10, Figure 6-4) used to OR in status bits into the lower 2 bits of the micropointer field (UPF). With particular values of the MOD and CLK CTL fields, the branch control field can be extended to the lower 5 bits of the UPF.

BUS FPA

Internal 32-bit wide FPA bus.

BUS NUA

Next microaddress bus. Located at output of microaddress sequencer.

Clock

Normally 180 ns when FPA is processing operands; 270 ns when FPA is synchronized with CPU.

Clock Field

Three-bit field (CS17:15, Figure 6-6) used to enable a number of clock and special functions.

CMP

Compare instruction (Figure 6-21).

CSR

Control store register.

CVT

Conversion instruction (Figure 6-21) used to convert one data type to another.

D	64-bit double format.
Divide-by-Zero	Exception (error) condition that occurs when the divisor is a zero. For this condition the destination is unaffected and the condition codes are unpredictable.
DIVL	Longword division instruction (Figure 6-23).
EMOD	Extended precision multiply and integerize (Figure 6-21).
Exception	Error condition that occurs during operand processing; reported to the CPU via the Y-Bus.
Excess Notation	Bias (80,400,4000) used to store and handle the exponent portion of floating-point numbers.
Exponent	Contains power of 2 in a bias format. Is an 8-bit value for single (F) and double (D), 11-bit value for grand (G), and a 15-bit value for huge (H) data formats.
EXP CTL Field	CS 44:39 (Figure 6-15).
EXP DST Field	Exponent destination control field (Figure 6-16).
Exponent Data Path	16-bit wide data path.
Extended Op Code	Op code equal to FD; used to extend the VAX instruction code beyond the normal 8-bits of the IB-Bus.
FALU Field	Fraction ALU function field (Figure 6-13).
Force	CPU inhibits operation of FPA microaddress sequencer and then writes (forces) a microaddress into control store via the Y-Bus.
F	32-bit long single format.
FPA	Floating-point accelerator.
FPAA through FPAN	FPA schematic logic diagrams.
Fraction Data Path	64-bit wide data path.
FRAC Field	Fraction control field (Figure 6-1).
Fraction	Normalized, magnitude binary representation with sign and magnitude notation.
FRSC Field	Fraction ALU source operand field (Figure 6-12).
FSHF Field	Fraction ALU destination control field (Figure 6-14).

G	Grand format.
Grand Format	64-bit longword format.
Guard Bits	Bits used to save the LSBs of an operand that have been shifted out of the fraction and are required for precision reasons.
Hidden Bit	Because MSB of fractions stored in memory is always a logical one, CPU does not send this bit. Therefore, FPA inserts a one into this bit into MSB of every fraction whenever it receives an operand from the CPU.
H	Huge.
Huge Format	128-bit longword.
IB-Bus	Instruction bus used for transfer of op codes to FPA.
Integer Data Path	Fraction data path 47:16.
IRD	Instruction decoding state.
Literal (LIT) Field	8-bit field (CS7:0, Figure 6-2) control store applies to microaddress sequencer.
Load	CPU sends FPA operands.
LSB	Least significant bit.
Microaddress	10-bit field normally generated by FPA microaddress sequencer (or forced by CPU) to select required data path setup signals during operand processing.
Micropointer Field (UPF)	10-bit field (CS9:0, Figure 6-3) that specifies the base of the next microaddress of the microaddress sequencer.
Microword	10-bit microaddress word applied to control store.
MIER	Multiplier.
MOD Field	Two-bit modify field (CS21:20, Figure 6-8) used to extend use of other fields and also enable special functions.
MSB	Most significant bit.
MUL	Shortword multiplication instruction (Figure 6-21).
Normalization	Alignment of fraction resultant with fraction data path MSB.
Op Code	Eight-bit operation code field that indicates what operation (instruction) must be performed on operands received on the Y-Bus.

Operand	Data received on the Y-Bus that is to be operated on.
Overflow	Exception (error) that occurs when exponent of floating-point number is larger than the largest representable exponent for the data type after normalization and rounding have been performed.
PAL	Programmable array logic.
Parity Field	Two-bit field (CS46:45, Figures 6-17, 6-18) used to check for control store errors.
POLY	Polynomial instruction (Figure 6-23).
Prealignment	Exponents are made equal (prealigned) prior to addition or subtraction of two floating-point numbers.
Probing	Process of determining if address is accessible.
PROM	Programmable read-only memory.
RAM A Field	Four-bit field (CS29:26, Figure 6-11) used to address the scratch pad of both the exponent and fraction data paths.
RAM B Field	Four-bit field (CS25:22, Figure 6-10) used to address scratch pad of both the exponent and fraction data paths.
Range Test	Test performed on exponents prior to addition or subtraction of two floating-point numbers to determine if prealignment/addition is required.
ROM	Read-only memory.
Rounding	Adding a one to the most significant guard bit.
RTOL	Right-to-left-reading (Figure 6-1).
Save	Signal name suffix that indicates signal name in question (e.g., EXT R0 SAVE H) was generated in the previous cycle.
SHF (Shift) Field	Two-bit field (CS19:18, Figure 6-7) that controls a number of shifting functions.
Size Field	Two-bit field output of instruction decoding logic. Field value indicates size (F, D, G, or H) of operand to be received from CPU on Y-Bus.
Status Register	Branch logic register that receives status signals from data path logic.
Store	FPA result sent to CPU.
SUB	Subtract instruction (Figure 6-21).
Summation	Addition of two numbers when sign of both operands are the same.

Trap

CPU traps (halts) FPA at current microaddress so that it can be read out to the Y-Bus.

Underflow

Exception (error) condition that occurs when the exponent of a floating-point number is smaller than the smallest representable exponent for the data type, after normalization and rounding have been performed.

UPF

Micropointer field.

Y-Bus

32-bit wide FPA-CPU operand interface bus.

VAX-11/730 FP730 FPA
Technical Description
EK–FP730–TD–001

Reader's Comments

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our publications.

What is your general reaction to this manual? In your judgment is it complete, accurate, well organized, well written, etc? Is it easy to use? ____________________

What features are most useful? ____________________

What faults or errors have you found in the manual? ____________________

Does this manual satisfy the need you think it was intended to satisfy? ____________________

Does it satisfy *your* needs? __________ Why? ____________________

Please send me the current copy of the *Documentation Products Directory*, which contains information on the remainder of DIGITAL's technical documentation.

Name ____________________ Street ____________________
Title ____________________ City ____________________
Company ____________________ State/Country ____________________
Department ____________________ Zip ____________________

Additional copies of this document are available from:

Digital Equipment Corporation
Accessories and Supplies Group
P.O. Box CS2008
Nashua, New Hampshire 03061

Attention: Documentation Products
Telephone: 1-800-258-1710

Order No. EK–FP730–TD–001

Fold Here

Do Not Tear – Fold Here and Staple

digital

No Postage
Necessary
if Mailed in the
United States

Digital Equipment Corporation
Educational Services/Quality Assurance
12 Crosby Drive, BU/E08
Bedford, MA 01730