

# Datapath Design

Book of David A. Patterson

Pp: 203-211

Book of P. Hayes

PP: 266-273

# Example of Floating point Addition

$9.999 \times 10^1 + 1.610 \times 10^{-1}$ . we can store four decimal digits of the significand and two decimal digits of the exponent.

Step 1: Align the decimal point of the number that has the smaller exponent.

$$1.610_{\text{ten}} \times 10^{-1} = 0.1610_{\text{ten}} \times 10^0 = \underline{0.01610_{\text{ten}} \times 10^1}$$

After shifting, the number is  $0.016_{\text{ten}} \times 10^1$

Step 2: Add the significands.

$$\begin{array}{r} 9.999_{\text{ten}} \\ + 0.016_{\text{ten}} \\ \hline 10.015_{\text{ten}} \end{array}$$

The sum is  $10.015_{\text{ten}} \times 10^1$ .

# Example of Floating point Addition

## Step 3:

Normalize the result.

$$10.015_{\text{ten}} \times 10^1 = 1.0015_{\text{ten}} \times 10^2$$

Check for underflow and overflow.  $+127 > 2 > -126$

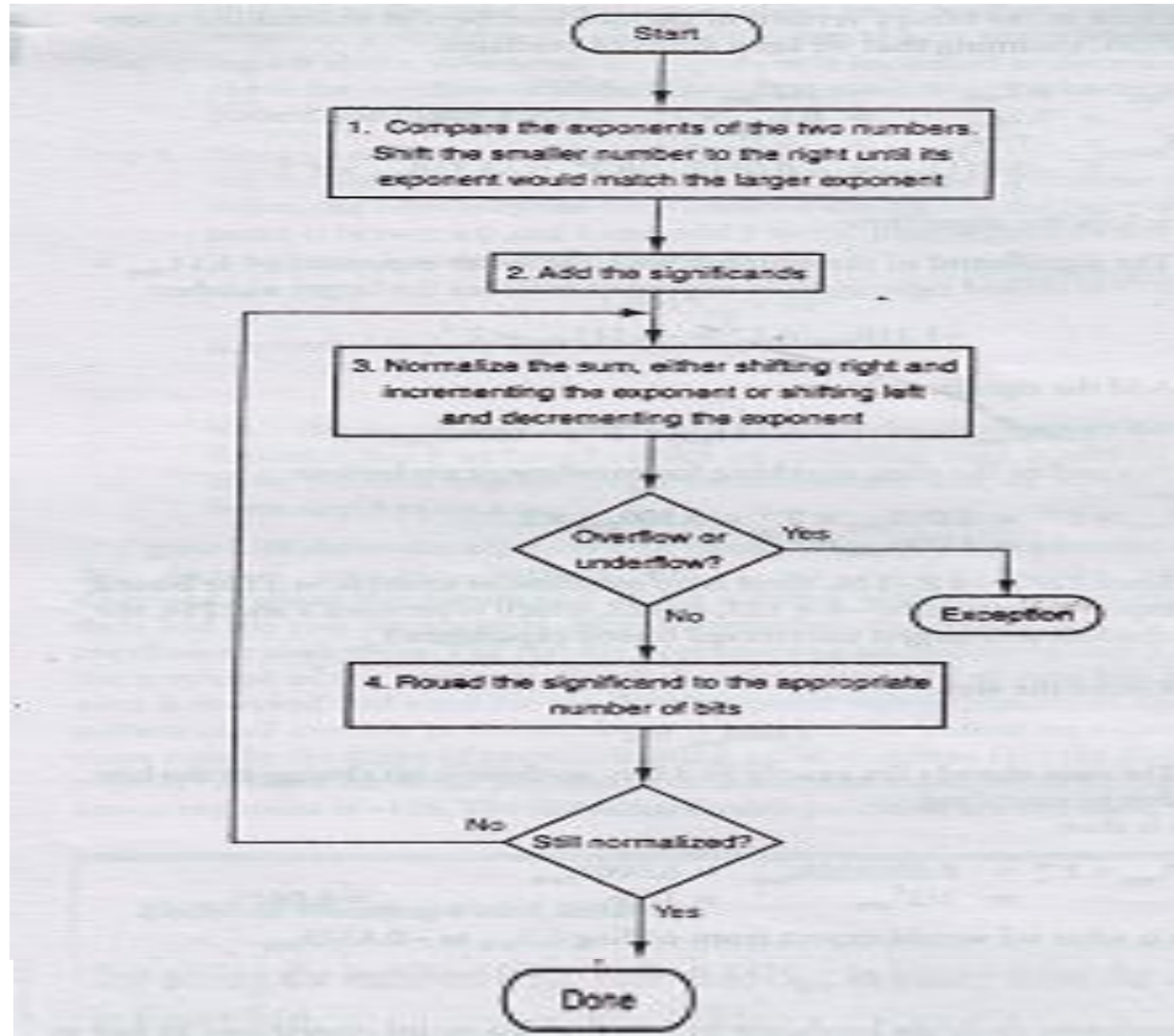
## Step 4:

Round the result.

Rules: truncate the number if the digit to the right of the desired point is between 0 and 4. Add 1 to the digit if the number to the right is between 5 and 9.

$$1.002_{\text{ten}} \times 10^2$$

# Floating point Addition

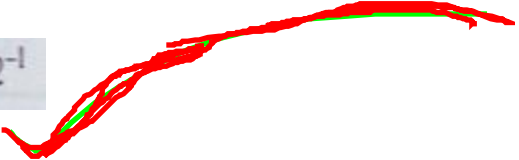


# Example

Add the numbers 0.5 and -0.4375 in binary.

$$\begin{aligned} 0.5_{\text{ten}} &= 1/2_{\text{ten}} = 1/2^1_{\text{ten}} \\ &= 0.1_{\text{two}} = 0.1_{\text{two}} \times 2^0 = 1.000_{\text{two}} \times 2^{-1} \\ -0.4375_{\text{ten}} &= -7/16_{\text{ten}} = -7/2^4_{\text{ten}} \\ &= -0.0111_{\text{two}} = -0.0111_{\text{two}} \times 2^0 = -1.110_{\text{two}} \times 2^{-2} \end{aligned}$$

Step 1:  $-1.110_{\text{two}} \times 2^{-2} = -0.111_{\text{two}} \times 2^{-1}$



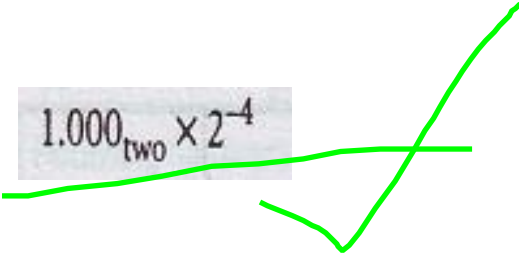
Step 2:  $1.000_{\text{two}} \times 2^{-1} + (-0.111_{\text{two}} \times 2^{-1}) = 0.001_{\text{two}} \times 2^{-1}$

Step 3:  $0.001_{\text{two}} \times 2^{-1} = 0.010_{\text{two}} \times 2^{-2} = 0.100_{\text{two}} \times 2^{-3} = 1.000_{\text{two}} \times 2^{-4}$

Since  $127 \geq -4 \geq -126$ , there is no overflow and underflow.

# Example

Step 4: Round the sum

$$1.000_{\text{two}} \times 2^{-4}$$


The sum is

$$\begin{aligned} 1.000_{\text{two}} \times 2^{-4} &= 0.0001000_{\text{two}} = 0.0001_{\text{two}} \\ &= 1/2^4_{\text{ten}} = 1/16_{\text{ten}} = 0.0625_{\text{ten}} \end{aligned}$$

# Example of Floating-Point Multiplication

$1.110 \times 10^{10} \times 9.200 \times 10^{-5}$ . we can store only four digits of the significand and two digits of the exponent.

Step 1:

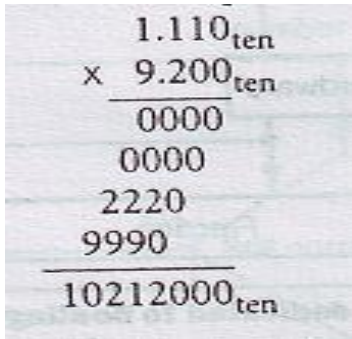
Calculate the exponent of the product by simply adding the exponents of the operands together: `New exponent = 10 + (-5) = 5`

Using the biased notation,

$$\begin{aligned}\text{New exponent} &= (10+127)+(-5+127) - 127 \\ &= 137 + 122 - 127 \\ &= 132 = 5 + 127\end{aligned}$$

# Example of Floating Point Multiplication

Step 2: Multiply the significands.



A handwritten multiplication problem in base 10. The first number is 1.110<sub>ten</sub> and the second is 9.200<sub>ten</sub>. The multiplication is shown with a horizontal line under the second number. The partial products are 0000, 0000, 2220, and 9990. The final product is 10212000<sub>ten</sub>.

The product is 10.212000<sub>ten</sub> and the final product is 10.212  $\times 10^5$

Step 3:

Normalize the product and check for underflow and overflow.

$$10.212_{\text{ten}} \times 10^5 = 1.0212_{\text{ten}} \times 10^6$$



# Example of Floating Point Multiplication

## Step 4:

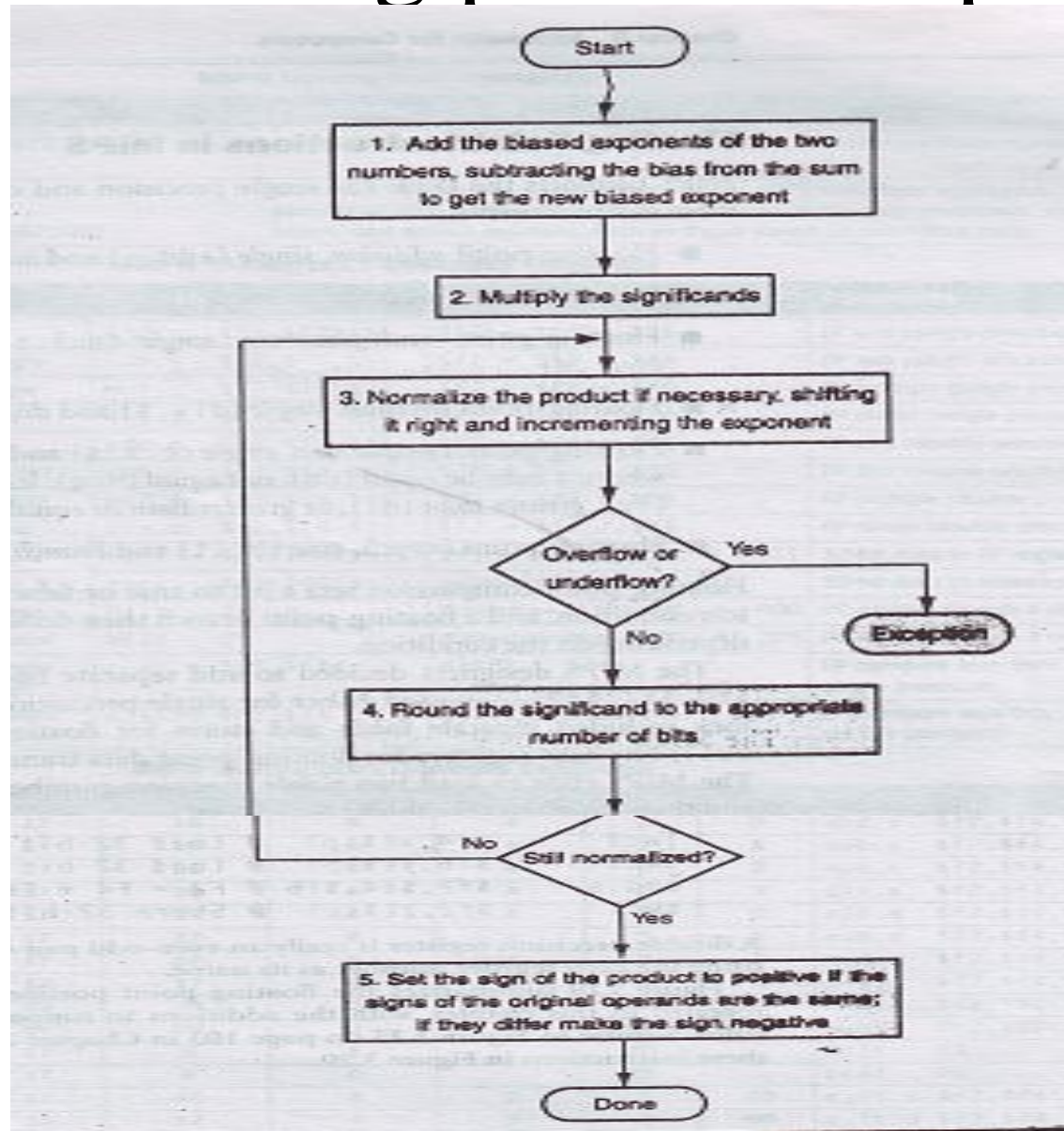
The number  $1.0212 \times 10^6$  is rounded to  $1.021 \times 10^6$

## Step 5:

Set the sign of the product. The sign of the product depends on the sign of the original operands. If they are both the same, the sign is positive. Otherwise it is negative.

$+ 1.021 \times 10^6$

# Floating point Multiplication



# Example

Multiply the numbers  $0.5_{\text{ten}}$  and  $-0.4375_{\text{ten}}$ .

In binary,  $1.000_{\text{two}} \times 2^{-1}$  by  $-1.110_{\text{two}} \times 2^{-2}$ .

Step 1:  $-1 + (-2) = -3$

In biased representation:

$$\begin{aligned} (-1 + 127) + (-2 + 127) - 127 &= (-1 - 2) + (127 + 127 - 127) \\ &= -3 + 127 = 124 \end{aligned}$$

Step 2: Multiply the significands.

$$\begin{array}{r} 1.000_{\text{two}} \\ \times 1.110_{\text{two}} \\ \hline 0000 \\ 1000 \\ 1000 \\ 1000 \\ \hline 1110000_{\text{two}} \end{array}$$

# Example

The product is  $1.110000 \times 2^{-3}$ . Final product is  $1.110 \times 2^{-3}$ .

Step 3: The product is already normalized and there is no overflow and underflow since ,  $127 \geq -3 \geq -126$

Step 4: Rounding the product makes no change.

Step 5: The final product is -  $1.110 \times 2^{-3}$

Converting to decimal,

$$\begin{aligned} -1.110_{\text{two}} \times 2^{-3} &= -0.001110_{\text{two}} = -0.00111_{\text{two}} \\ &= -7/2^5_{\text{ten}} = -7/32_{\text{ten}} = -0.21875_{\text{ten}} \end{aligned}$$

# Floating Point Arithmetic

Let  $(X_M, X_E)$  be a floating point representation of a number  $X$ , that has the numerical value  $X_M \times B^{X_E}$ , where  $X_M$  and  $X_E$  are fixed point numbers.

## Basic Arithmetic Operations on Floating-point Number:

Addition	$X + Y = (X_M 2^{X_E - Y_E} + Y_M) \times 2^{Y_E}$	} where $X_E \leq Y_E$
Subtraction	$X - Y = (X_M 2^{X_E - Y_E} - Y_M) \times 2^{Y_E}$	
Multiplication	$X \times Y = (X_M \times Y_M) \times 2^{X_E + Y_E}$	
Division	$X/Y = (X_M / Y_M) \times 2^{X_E - Y_E}$	

# Floating-point Addition and Subtraction

Floating-point addition and subtraction have 3 main steps:

1. Compute  $Y_E - X_E$ , a fixed point subtraction.
2. Shift  $Y_E - X_E$  places to the right to form  $X_M 2^{X_E - Y_E}$ .
3. Compute  $X_M 2^{X_E - Y_E} \pm Y_M$ , a fixed point addition and subtraction.

Example:

$$\underline{X = 1.32400111 \times 10^{17}} \text{ and } \underline{Y = 1.04799245 \times 10^{21}}$$

$$Y_E - X_E = 21 - 17 = 4$$

$$X_M 2^{-4} = 0.00013240$$

$$X_M 2^{-4} + Y_M = 0.00013240 + 1.04799245 = 1.04812485$$

The final result is  $1.04812485 \times 10^{21}$

# Difficulties

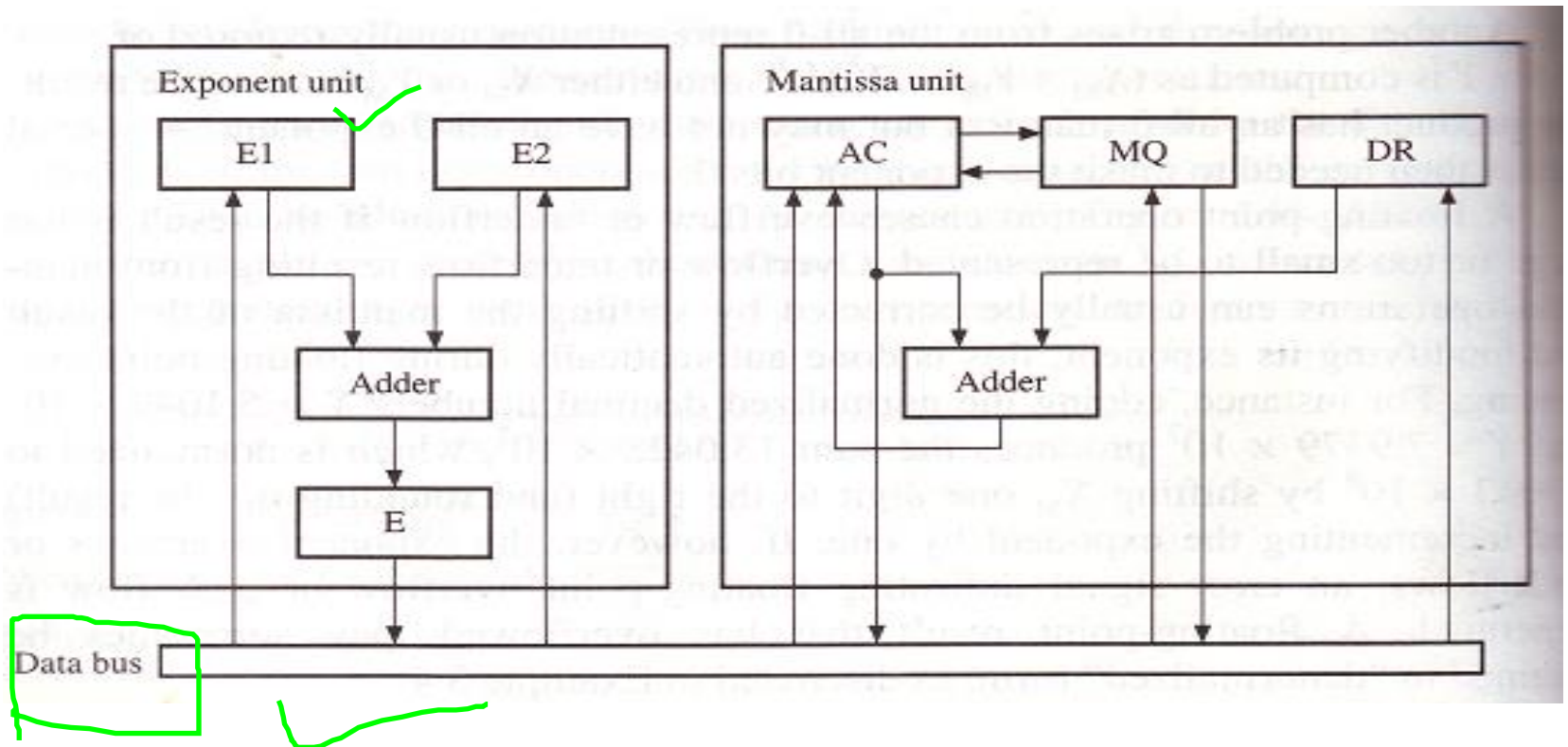
1. If biased exponents are added and subtracted using fixed-point arithmetic in the course of a floating-point calculation, the resulting exponent is doubly biased and must be corrected by subtracting the bias.
2. If  $X \times Y$  is computed as  $(X_M \times Y_M) \times 2^{X_E+Y_E}$  and either  $X_M$  or  $Y_M$  is 0, the resulting product has an all-0 mantissa but may not have all-0 exponent. A special step is then needed to make the exponent bits 0.
3. Overflow and underflow may arise in both mantissa and exponents manipulation. Mantissa overflow and underflow can be adjusted by shifting exponent. If the exponents overflow or underflow then error must be detected.

# Difficulties

4. We need two guard bits to preserve the accuracy of floating-point calculations: 1. for rounding purpose. 2. to maintain precision during normalization.



# Floating-point Unit



# Algorithm for Floating-point Addition

```

register AC [ $n_M-1:0$ ], DR [ $n_M-1:0$ ], E [ $n_E-1:0$ ], E1 [ $n_E-1:0$ ], E2 [ $n_E-1:0$ ];
AC_OVERFLOW, ERROR;

BEGIN:      AC_OVERFLOW := 0, ERROR := 0;
LOAD:      E1 := XE, AC := XM;
           E2 := YE, DR := YM;

{Compare and equalize exponents}
COMPARE:    E := E1 - E2;
EQUALIZE:   if E < 0 then AC := right-shift(AC), E := E + 1,
           go to EQUALIZE; else
           if E > 0 then DR := right-shift(DR), E := E - 1,
           go to EQUALIZE;

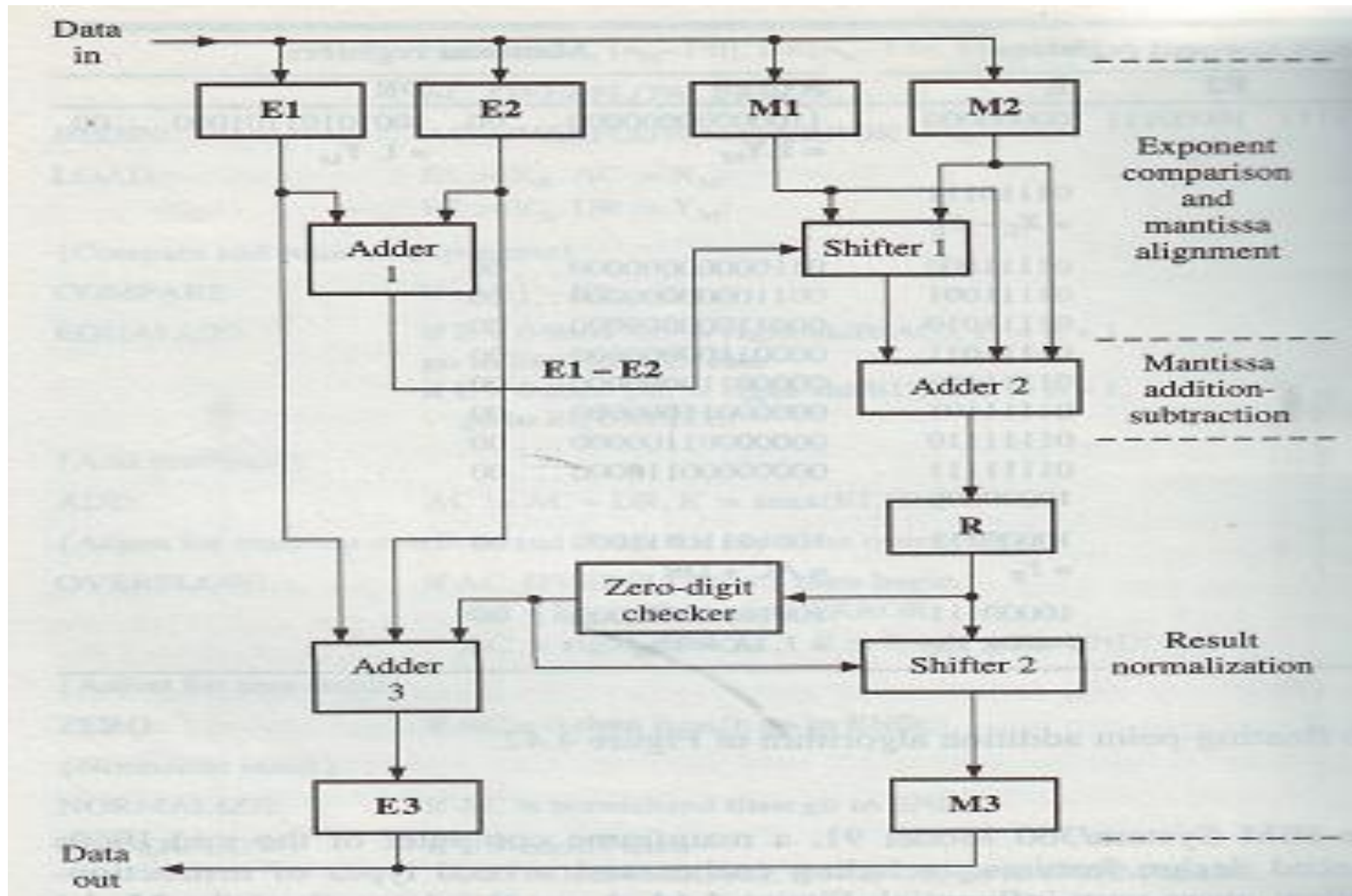
{Add mantissas}
ADD:        AC := AC + DR, E := max(E1, E2);
{Adjust for mantissa overflow and check for exponent overflow}
OVERFLOW:   if AC_OVERFLOW = 1 then begin
           if E = EMAX then go to ERROR;
           AC := right-shift(AC), E := E + 1, go to END; end
{Adjust for zero result}
ZERO:       if AC = 0 then E := 0, go to END;
{Normalize result}
NORMALIZE:  if AC is normalized then go to END;
UNDERFLOW:  if E > EMIN then
           AC := left-shift(AC), E := E - 1, go to NORMALIZE;
{Set error flag indicating overflow or underflow}
ERROR:      ERROR := 1;
END:

```

# Example

Exponent registers			Mantissa registers	
E1	E2	E	AC	DR
01111111 $= X_E$	10000111 $= Y_E$	00000000	11000000000000 ... 00 $= 1.X_M$	10010101101000 ... 00 $= 1.Y_M$
		01110111 $= X_E - Y_E$		
		01111000	01100000000000 ... 00	
		01111001	00110000000000 ... 00	
		01111010	00011000000000 ... 00	
		01111011	00001100000000 ... 00	
		01111100	00000110000000 ... 00	
		01111101	00000011000000 ... 00	
		01111110	00000001100000 ... 00	
		01111111	00000000110000 ... 00	
		10000000		
		10000111 $= Y_E$	10010110011000 ... 00 $= AC + DR$	
		10000111 $= (X + Y)_E$	10010110011000 ... 00 $= 1.(X + Y)_M$	

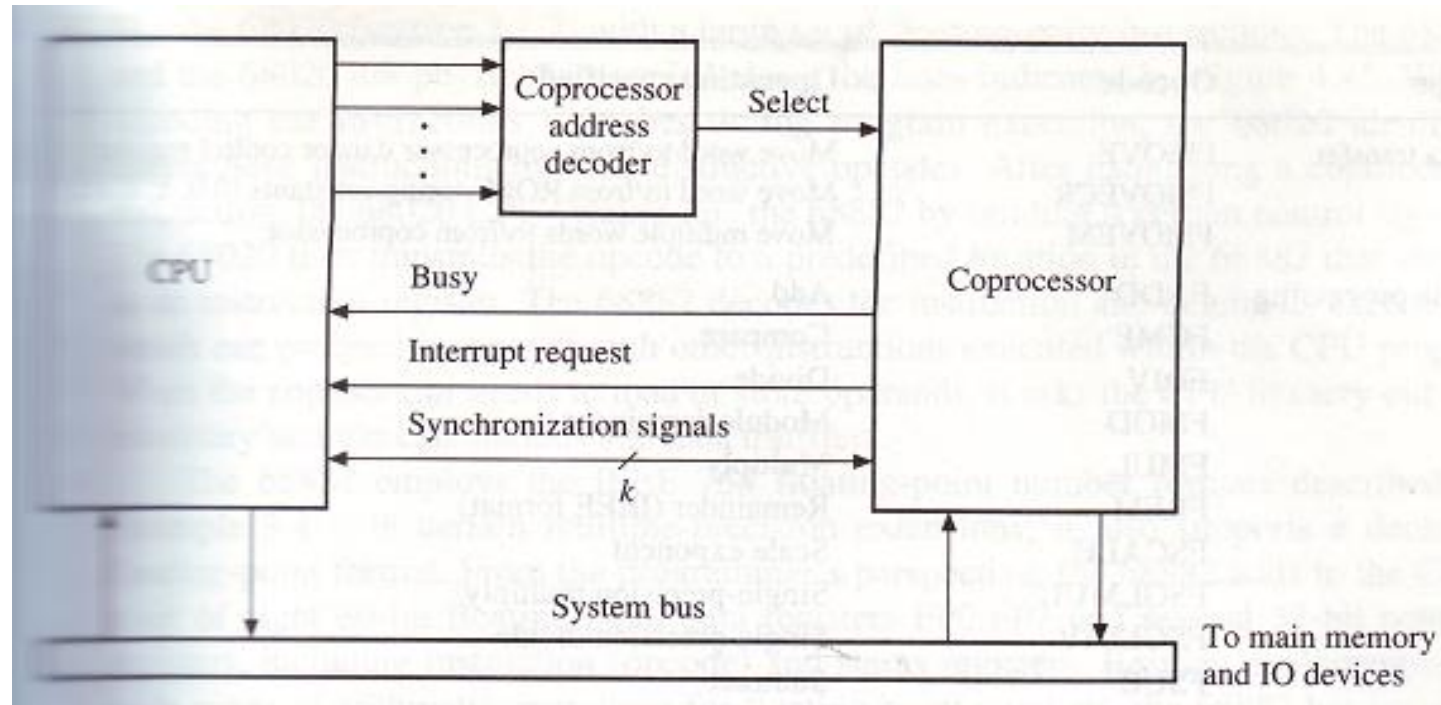
# Floating Unit of IBM System



# Coprocessor

- ✓ An auxiliary processors to provide fast, low-cost hardware implementation of some special functions.
- ✓ A separate instruction set processor that is closely coupled with to the CPU and whose instructions and registers are direct extensions of CPU.
- ✓ It requires specialized control logic to link the CPU with the coprocessor and to handle the instructions that are executed by the coprocessor.

# Coprocessor



# Coprocessor

- ✓ A coprocessor instruction contains three fields:
  1. Opcode
  2. The address of the particular coprocessor.
  3. Particular operation to be executed by the coprocessor.