UNIT II

Arithmetic Unit

Addition and subtraction of signed numbers, Design of fast adders, Multiplication of positive Numbers, Signed operand multiplication and fast multiplication, Integer division , Floating point numbers and operations.

2.1 Arithmetic Unit

2.1.1 Addition and Subtraction of Signed Numbers

Q1. Explain the procedure for Addition and Subtraction with signed-magnitude data with the help of flowchart.

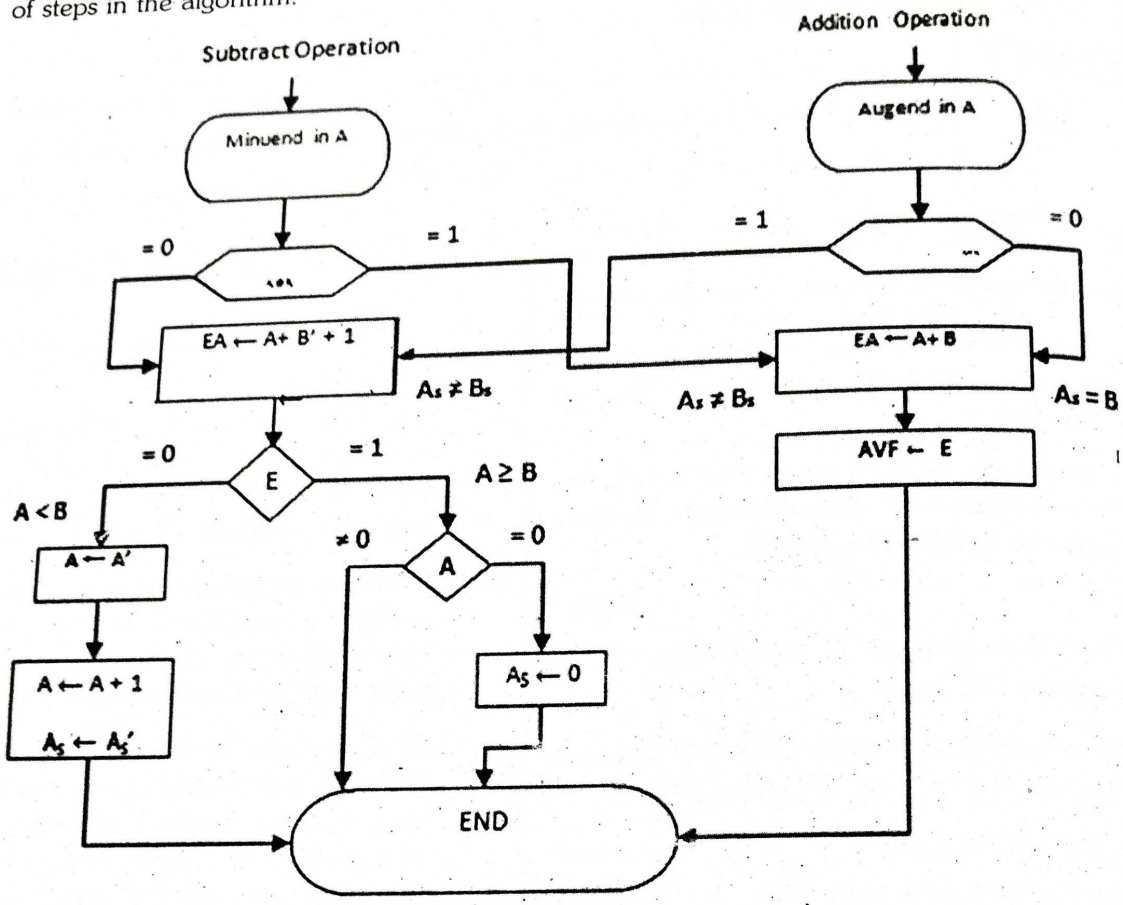
(Imp.)

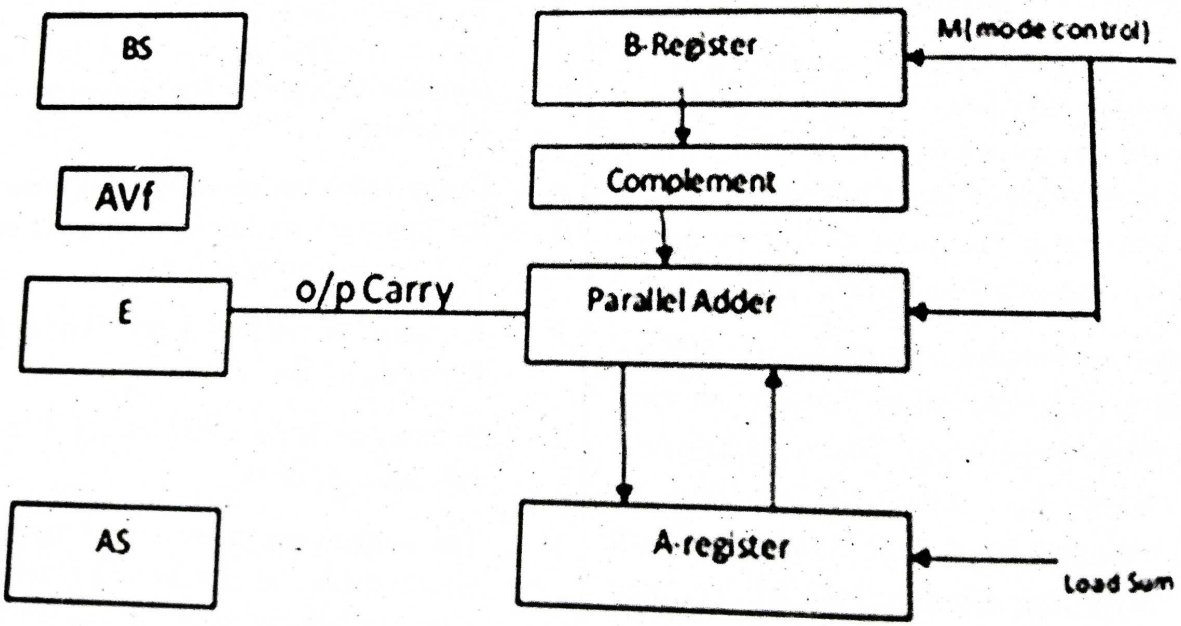
Ans :

- The two signs A, and B, are compared by an exclusive-OR gate.
- If the output of the gate is 0 the signs are identical; If it is 1, the signs are different.
- For an add operation, identical signs dictate that the magnitudes be added. For a subtract operation, different signs dictate that the magnitudes be added.
- The magnitudes are added with a microoperation EAU A + B, where EA is a register that combines E and A. The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred into the add-overflow flip-flop AVF.
- The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. The magnitudes are subtracted by adding A to the 2's complemented B. No overflow can occur if the numbers are subtracted so AVF is cleared to 0.
- 1 in E indicates that A >= B and the number in A is the correct result. If this numbs is zero, the sign A must be made positive to avoid a negative zero.
- 0 in E indicates that A < B. For this case it is necessary to take the 2's complement of the value in A. The operation can be done with one microoperation $A^{U}A' + 1$.
- However, we assume that the A register has circuits for microoperations complement and increment, so the 2's complement is obtained from these two microoperations.
- In other paths of the flowchart, the sign of the result is the same as the sign of A. so no change in A is required. However, when A < B, the sign of the result is the complement of the original sign of A. It is then necessary to complement A, to obtain the correct sign.
- The final result is found in register A and its sign in As. The value in AVF provides an overflow indication. The final value of E is immaterial.
- Figure following shows a block diagram of the hardware for implementing the addition and subtraction operations.
- It consists of registers A and B and sign flip-flops As and Bs.

  Subtraction is done by adding A to the 2's complement of B.
- The output carry is transferred to flip-flop E , where it can be checked to determine the relative magnitudes of two numbers.
- The add-overflow flip-flop AVF holds the overflow bit when A and B are added.

➢ The A register provides other microoperations that may be needed when we specify the sequence of steps in the algorithm.





Hardware for signed-magnitude addition and subtraction

2.1.2 Design of F

Q2. Explain abo adders or ca help of circu

Ans :

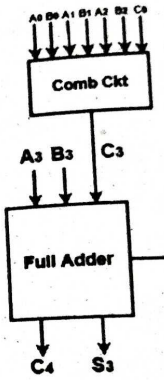
➢ Carry Look version of th

➢ It generates simultaneous

➢ The time con = Θ (logn)

Logic Diagram

The logic di is as shown belo



Carry Look A

The carr only on the fo

➢ Bits bei

➢ Carry-in

Now,

➢ The ab from th

➢ So, the be eva

➢ Thus, carry-i adder.

2.1.2 Design of Fast Adders

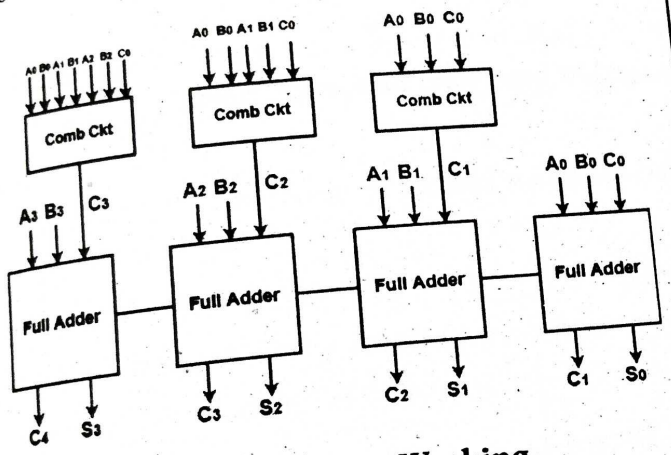
Q2. Explain about implementation of Fast adders or carry look ahead adders with help of circuit diagram.

Ans :

- Carry Look Ahead Adder is an improved version of the ripple carry adder.
- It generates the carry-in of each full adder simultaneously without causing any delay.
- The time complexity of carry look ahead adder $= \Theta$ (logn).

Logic Diagram

The logic diagram for carry look ahead adder is as shown below.



Carry Look Ahead Adder Working

The carry-in of any stage full adder depends only on the following two parameters-

- Bits being added in the previous stages
- Carry-in provided in the beginning

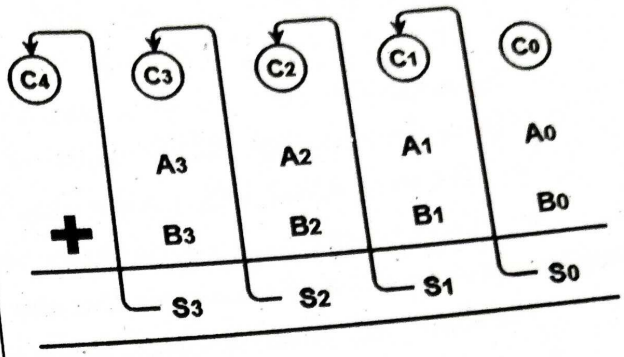
Now,

- The above two parameters are always known from the beginning.
- So, the carry-in of any stage full adder can be evaluated at any instant of time.
- Thus, any full adder need not wait until its carry-in is generated by its previous stage full adder.

4-Bit Carry Look Ahead Adder

Consider two 4-bit binary numbers $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are to be added.

Mathematically, the two numbers will be added as,



Adding two 4-bit Numbers

From here, we have,

$$C_1 = C_0 (A_0 \oplus B_0) + A_0B_0$$
$$C_2 = C_1 (A_1 \oplus B_1) + A_1B_1$$
$$C_3 = C_2 (A_2 \oplus B_2) + A_2B_2$$
$$C_4 = C_3 (A_3 \oplus B_3) + A_3B_3$$

For simplicity, Let,

- $G_i = A_iB_i$ where G is called carry generator
- $P_i = A_i \oplus B_i$ where P is called carry propagator

Then, re-writing the above equations, we have,

$$C_1 = C_0P_0 + G_0 \quad \ldots (1)$$
$$C_2 = C_1P_1 + G_1 \quad \ldots (2)$$
$$C_3 = C_2P_2 + G_2 \quad \ldots (3)$$
$$C_4 = C_3P_3 + G_3 \quad \ldots (4)$$

Now,

- Clearly, C_1, C_2 and C_3 are intermediate carry bits.
- So, let's remove C_1, C_2 and C_3 from RHS of every equation.

- Substituting (1) in (2), we get C_2 in terms of C_0.
- Then, substituting (2) in (3), we get C_3 in terms of C_0 and so on.

Finally, we have the following equations,

- $C_1 = C_0P_0 + G_0$
- $C_2 = C_0P_0P_1 + G_0P_1 + G_1$
- $C_3 = C_0P_0P_1P_2 + G_0P_1P_2 + G_1P_2 + G_2$
- $C_4 = C_0P_0P_1P_2P3 + G_0P_1P_2P_3 + G_1P_2P_3 + G_2P_3 + G_3$

These equations are important to remember.

These equations show that the carry-in of any stage full adder depends only on,

- Bits being added in the previous stages
- Carry bit which was provided in the beginning

Implementation of Carry Generator Circuits

The above carry generator circuits are usually implemented as,

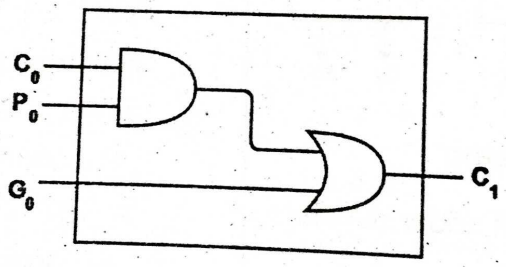
1. Two level combinational circuits.
2. Using AND and OR gates where gates are assumed to have any number of inputs.

Implementation of C_1

The carry generator circuit for C_1 is implemented as shown below.

It requires 1 AND gate and 1 OR gate.

$$C_1 = C_0P_0 + G_0$$



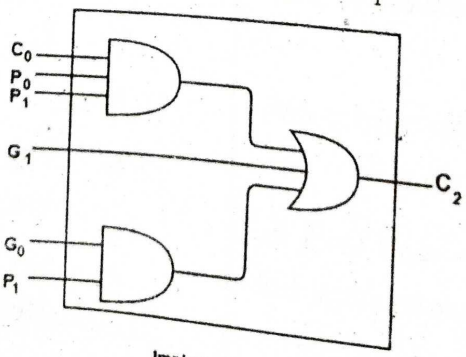
Implementation of C1

Implementation of C_2

The carry generator circuit for C2 is implemented as shown below.

1. It requires 2 AND gates and 1 OR gate.

$$C_2 = C_0P_0P_1 + G_0P_1 + G_1$$



Implementation of C2

Implementation of C_3 & C_4

Similarly, we implement C_3 and C_4.

- Implementation of C_3 uses 3 AND gates and 1 OR gate.
- Implementation of C_4 uses 4 AND gates and 1 OR gate.

Total number of gates required to implement carry generators (provided carry propagators P_i and carry generators G_i) are,

- Total number of AND gates required for addition of 4-bit numbers = 1 + 2 + 3 + 4 = 10.
- Total number of OR gates required for addition of 4-bit numbers = 1 + 1 + 1 + 1 = 4.

Advantages of Carry Look Ahead Adder

The advantages of carry look ahead adder are,

1. It generates the carry-in for each full adder simultaneously.
2. It reduces the propagation delay.

Disadvantages of Carry Look Ahead Adder

The disadvantages of carry look ahead adder are,

1. It involves complex hardware.
2. It is costlier since it involves complex hardware.
3. It gets more complicated as the number of bits increases.

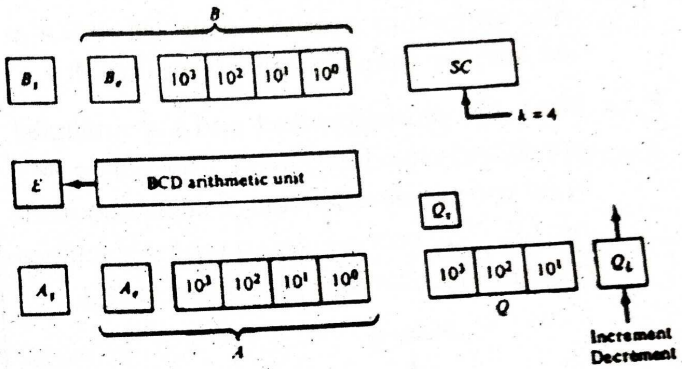
2.1.3 Multiplication of Positive Numbers

Q3. Explain with proper block diagram the Multiplication Operation on two decimal numbers. **(Imp.)**

Ans :

A decimal multiplier has digits ranging from 0 to 9.

- In the decimal case, the multiplicand must be multiplied by the digit multiplier and the result added to the partial product.
- This operation can be accomplished by adding the multiplicand to the partial product a number of times equal to the value of the multiplier digit. The registers organization for the decimal multiplication is shown in Fig. below.
- We are assuming here four-digit numbers, with each digit occupying four bits, for a total of 16 bits for each number. There are three registers, A, B, and Q, each having a corresponding sign flip-flop As, Bs, and Qs.



Registers for decimal arithmetic multiplication and division.

Registers A and B have four more bits designated by Ae, and Be, that provide an extension of one more digit to the registers

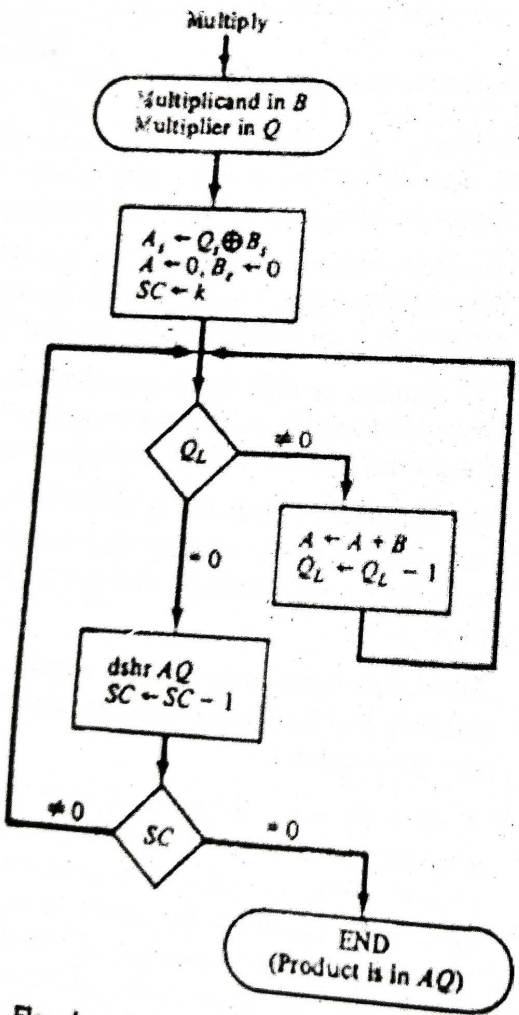
- The BCD arithmetic unit adds the five digits in parallel and places the sum in the fivedigitA register. The end-carry goes to flip-flop E.
- The purpose of digit Ae, is to accommodate an overflow while adding the multiplicand to the partial product during multiplication.
- The purpose of digit Be, is to form the 9's complement of the divisor when subtracted from the partial remainder during the division operation.
- The least significant digit in register Q is denoted by QL. This digit can be incremented or decremented.
- A decimal operand coming from memory consists of 17 bits. One bit (the sign) is transferred to Bs, and the magnitude of the operand is placed in the lower 16 bits of B. Both Be, and Ae, are cleared initially. The result of the operation is also 17 bits long and does not use the Ae, part of the A register

- Initially, the entire A register and Be are cleared and the sequence counter SC is set to a number k equal to the number of digits in the multiplier.

  The low-order digit of the multiplier in QL is checked. If it is not equal to 0, the multiplicand in B is added to the partial product in A once and QL is decremented. QL is checked again and the process is repeated until it is equal to 0.

- Any temporary overflow digit will reside in Ae and can range in value from 0 to 9.

- Next, the partial product and the multiplier are shifted once to the right. This places zero in Ae and transfers the next multiplier quotient into QL. The process is then repeated k times to form a double-length product in AQ.



Flowchart for decimal multiplication.

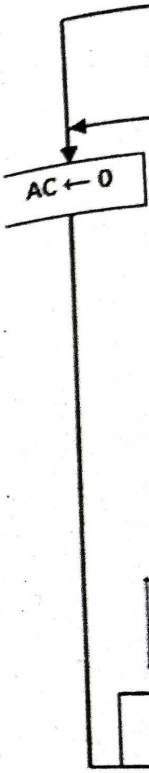
2.1.4 Signed Operand Multiplication and Fast Multiplication

Q4. Explain with proper block diagram the Multiplication Operation on two floating point numbers.

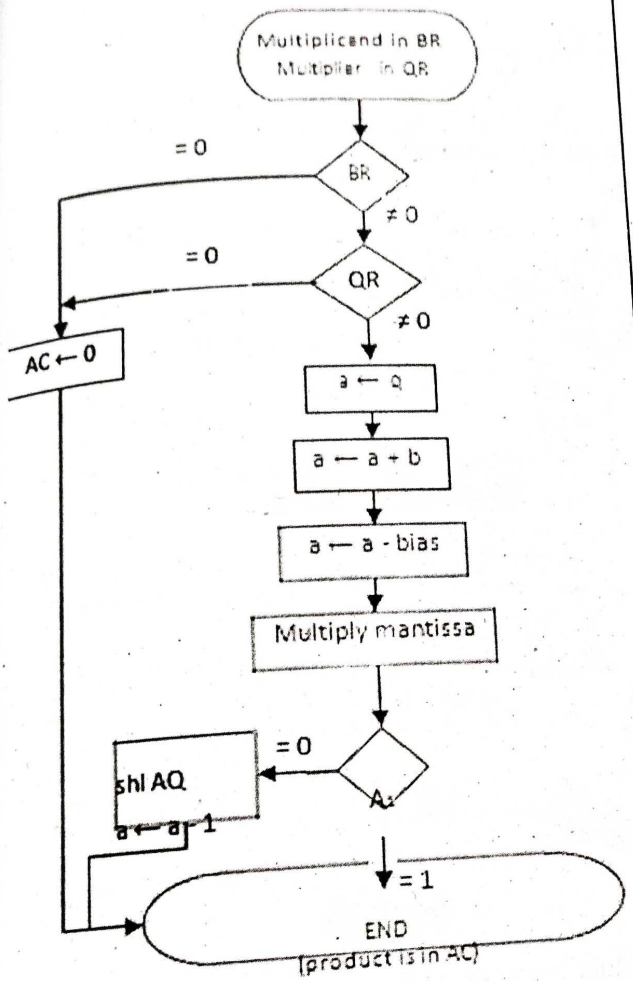
Ans :

(Imp.)

- The multiplication of two floating-point numbers requires that we multiply the mantissas and add the exponents.
- No comparison of exponents or alignment of mantissas is necessary.
- The multiplication of the mantissas is performed in the same way as in fixed-point to provide a double-precision product.
- The double-precision answer is used in fixed-point numbers to increase the accuracy of the product.
- In floating-point, the range of a single-precision mantissa combined with the exponent is usually accurate enough so that only single-precision numbers are maintained.
- Thus the half most significant bits of the mantissa product and the exponent will be taken together to form a single-precision floating-point product.
- The multiplication algorithm can be subdivided into four parts:
  - Check for zeros.
  - Add the exponents.
  - Multiply the mantissas.
  - Normalize the product.
- The flowchart for floating-point multiplication is shown in Figure 7.4. The two operands are checked to determine if they contain a zero.
- If either operand is equal to zero, the product in the AC is set to zero and the operation is terminated.
- If neither of the operands is equal to zero, the process continues with the exponent addition.



- The exponent of the multiplier is in q and the adder is between exponents a and b.



Multiplication of floating-point numbers

- The correct biased exponent for the product is obtained by subtracting the bias number from the sum.
- The multiplication of the mantissas is done as in the fixed-point case with the product residing in A and Q.
- Overflow cannot occur during multiplication, so there is no need to check for it.
- The product may have an underflow, so the most significant bit in A is checked. If it is a 1, the product is already normalized.
- If it is a 0, the mantissa in AQ is shifted left and the exponent decremented.
- Note that only one normalization shift is necessary. The multiplier and multiplicand were originally normalized and contained fractions. The smallest normalized operand is 0.1, so the smallest possible product is 0.01.
- Therefore, only one leading zero may occur.
- Although the low-order half of the mantissa is in Q, we do not use it for the floating-point product. Only the value in the AC is taken as the product.
- It is necessary to transfer the exponents from q to a, add the two exponents, and transfer the sum into a.
- Since both exponents are biased by the addition of a constant, the exponent sum will have double this bias.

Q5. Explain the Booth's algorithm with the help of flowchart.

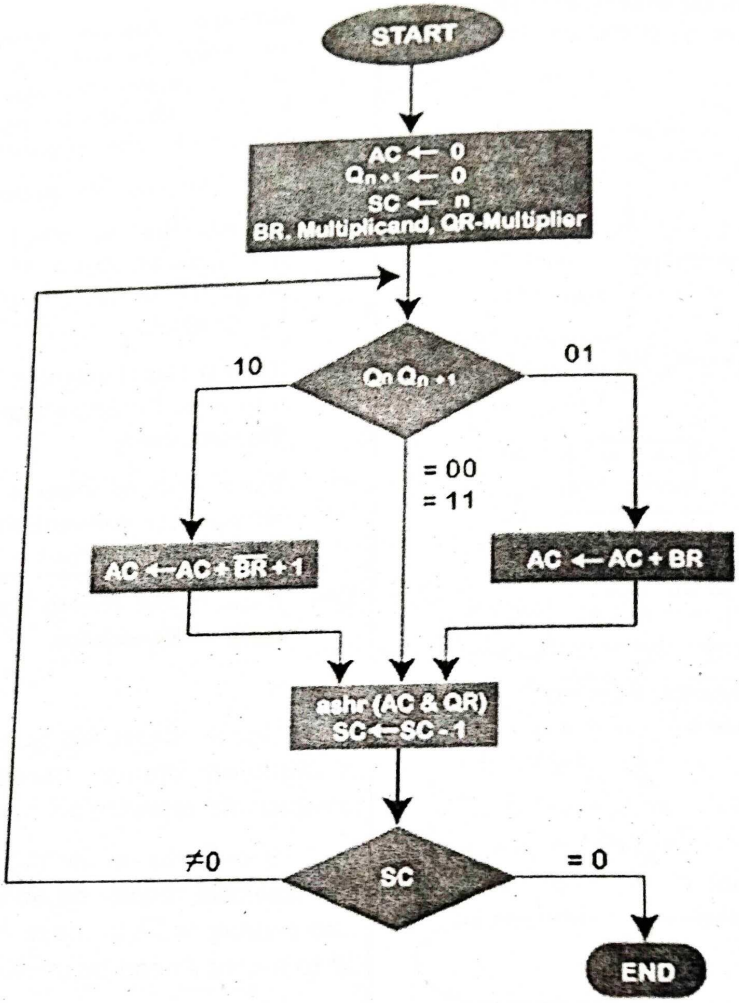
Ans :

Booth algorithm gives a procedure for multiplying binary integers in signed- 2's complement representation.

It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} - 2^m$.

For example, the binary number 001110 (+14) has a string 1's from 2^3 to 2^1 (k=3, m=1). The number can be represented as $2^{k+1} - 2^m. = 2^4 - 2^1 = 16 - 2 = 14$. Therefore, the multiplication M X 14, where M is the multiplicand and 14 the multiplier, can be done as M X 2^4 – M X 2^1.

- Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.
- As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of partial product.
- Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial, or left unchanged according to the following rules:



- The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.
- The multiplicand is added to the partial product upon encountering the first 0 in a string of 0's in the multiplier.
- The partial product does not change when multiplier bit is identical to the previous multiplier bit.
- The algorithm works for positive or negative multipliers in 2's complement representation.
- This is because a negative multiplier ends with a string of 1's and the last operation will be a subtraction of the appropriate weight.
- The two bits of the multiplier in Q_n and Q_{n+1} are inspected.
- If the two bits are equal to 10, it means that the first 1 in a string of 1 's has been encountered. This requires a subtraction of the multiplicand from the partial product in AC.
- If the two bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC.
- When the two bits are equal, the partial product does not change.

Q6. Multiply the tw

Ans :

Q_n	
1	0
1	
1	
0	
1	
1	

Q_{n+1} =

Hence

2.1.5 Inte

Q7. Expla

Ans :

Division

Posit

1. If th and

2. If th bac

Q6. Multiply the two numbers 23 and -9 by using the Booth's multiplication algorithm. **(Imp.)**

Ans :

Q_n	Q_{n+1}	M = 0 1 0 1 1 1 M' + 1 = 1 0 1 0 0 1	AC	Q	Q_{n+1}	SC
		Initially	000000	110111	0	6
1	0	Subtract M	101001			
			101001			
		Perform Arithmetic right shift operation	110100	111011	1	5
1	1	Perform Arithmetic right shift operation	111010	011101	1	4
1	1	Perform Arithmetic right shift operation	111101	001110	1	3
0	1	Addition (A + M)	010111			
			010100			
		Perform Arithmetic right shift operation	001010	000111	0	2
1	0	Subtract M	101001			
			110011			
		Perform Arithmetic right shift operation	111001	100011	1	1
1	1	Perform Arithmetic right shift operation	**111100**	**110001**	**1**	0

Q_{n+1} = 1, it means the output is negative.

Hence, 23 * -9 = 2's complement of 111100110001 => (00001100111)

2.1.5 Integer Division

Q7. Explain about division algorithm.

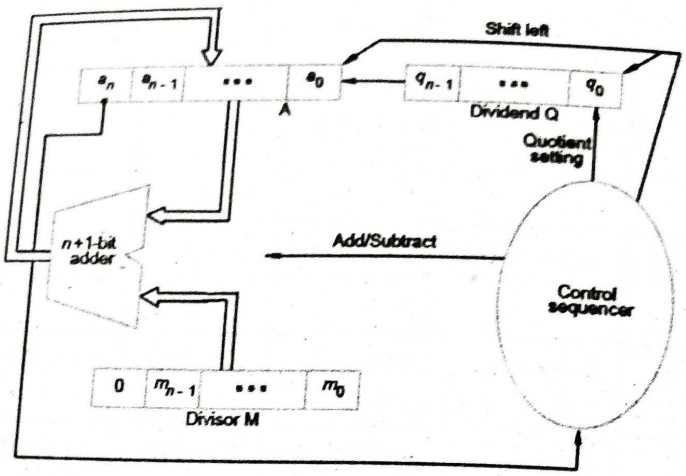
Ans :

Division Algorithm

Position the divisor appropriately with respect to the dividend and performs a subtraction.

1. If the remainder is zero or positive, a quotient bit of 1 is determined, the remainder is extended by another bit of the dividend, the divisor is repositioned, and another subtraction is performed.
2. If the remainder is negative, a quotient bit of 0 is determined, the dividend is restored by adding back the divisor, and the divisor is repositioned for another subtraction.

Circuit Arrangement



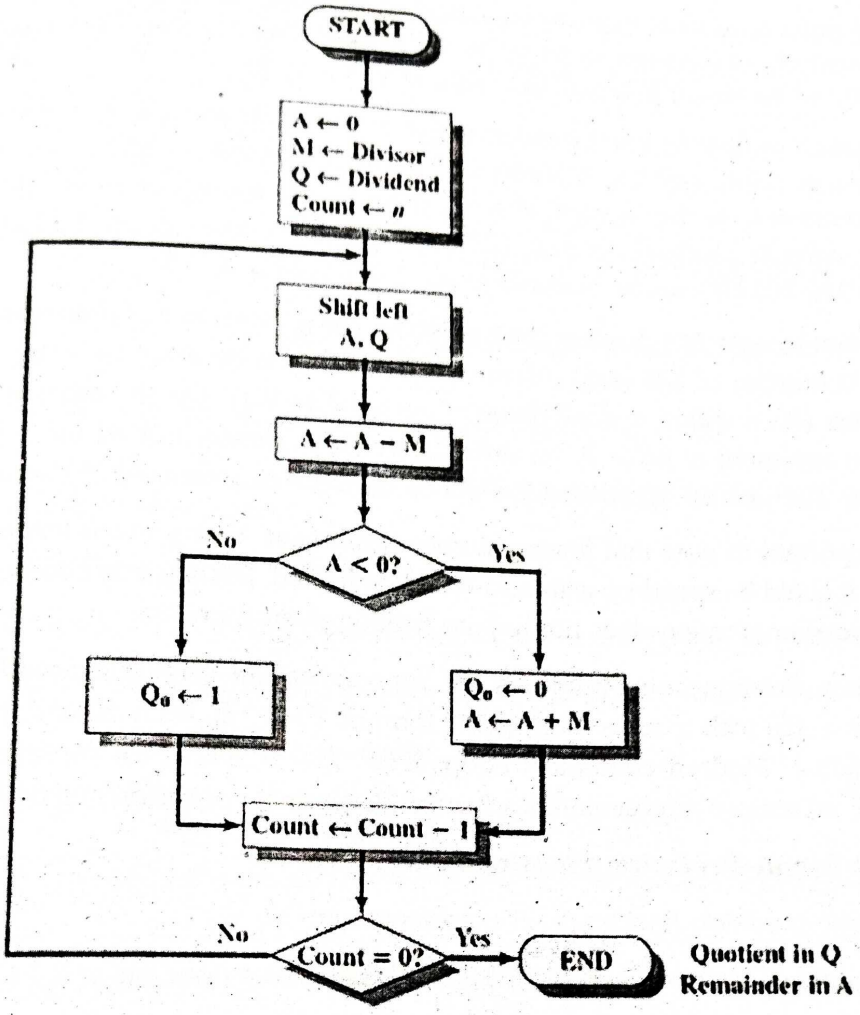
Restoring Division

- Shift A and Q left one binary position.
- Subtract M from A, and place the answer back in A.
- If the sign of A is 1, set q0 to 0 and add M back to A (restore A); otherwise, set q_0 to 1.
- Repeat these steps n times.

Non re-storing Division

1. Load the divisor into the M register. Load the dividend into the A, Q registers. (The dividend expressed as a 2n-bit positive number).
2. Shift A, Q left 1 bit position.
3. Subtract M from A.
   - If the result is positive, then add 1 to Q.
   - If the result is negative, restore the previous value of A.
4. This process repeated for each bit of the original dividend.
5. The remainder is in A and the quotient is in Q.

A	Q	
0000	0111	Initial value
0000	1110	Shift
1101		Use twos complement of 0011 for subtraction
1101		Subtract
0000	1110	Restore, set $Q_0 = 0$
0001	1100	Shift
1101		
1110		Subtract
0001	1100	Restore, set $Q_0 = 0$
0011	1000	Shift
1101		
0000	1001	Subtract, set $Q_0 = 1$
0001	0010	Shift
1101		
1110		Subtract
0001	0010	Restore, set $Q_0 = 0$



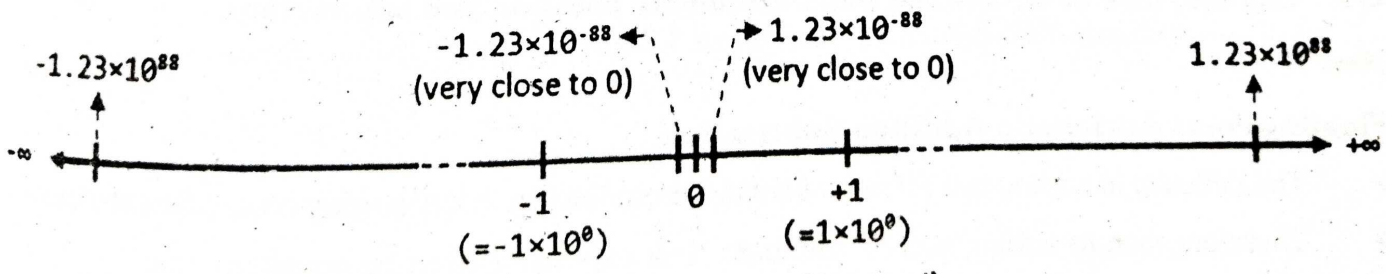
2.1.6 Floating Point Numbers And Operations

Q8. Write about the representation of floating point numbers.

Ans : **(Imp.)**

Floating-Point Representation (Scientific Notation)

A floating-point number (or real number) can represent a very large (1.23×10^88) or a very small (1.23×10^-88) value. It could also represent very large negative number (-1.23×10^88) and very small negative number (-1.23×10^88), as well as zero, as illustrated:



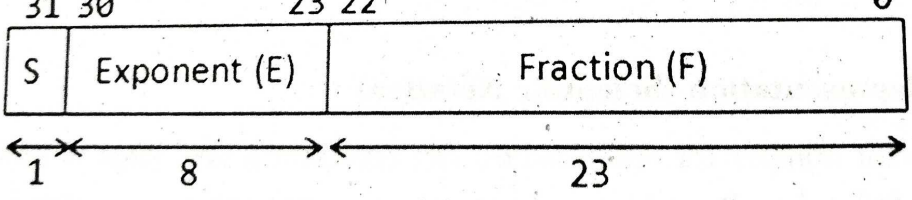
Floating-point Numbers (Decimal)

- A floating-point number is typically expressed in the scientific notation, with a fraction (F), a an exponent (E) of a certain radix (r), in the form of F×r^E. Decimal numbers use radix of (F×10^E); while binary numbers use radix of 2 (F×2^E).
- Representation of floating point number is not unique. For example, the number 55.66 can represented as 5.566×10^1, 0.5566×10^2, 0.05566×10^3, and so on. The fractional p can be normalized. In the normalized form, there is only a single non-zero digit before the ra point. For example, decimal number 123.4567 can be normalized as 1.234567×10^2; bina number 1010.1011B can be normalized as 1.0101011B×2^3.
- It is important to note that floating-point numbers suffer from *loss of precision* when represent with a fixed number of bits (e.g., 32-bit or 64-bit). This is because there are *infinite* number real numbers (even within a small range of says 0.0 to 0.1). On the other hand, a *n*-bit bina pattern can represent a *finite* 2^*n* distinct numbers. Hence, not all the real numbers can represented. The nearest approximation will be used instead, resulted in loss of accuracy.
- It is also important to note that floating number arithmetic is very much less efficient than integ arithmetic. It could be speed up with a so-called dedicated *floating-point co-processor*. Hence, u integers if your application does not require floating-point numbers.
- In computers, floating-point numbers are represented in scientific notation of *fraction* (F and *exponent* (E) with a *radix* of 2, in the form of F×2^E. Both E and F can be positive a well as negative. Modern computers adopt IEEE 754 standard for representing floating-poi numbers. There are two representation schemes: 32-bit single-precision and 64-bit double-precisio

IEEE-754 32-bit Single-Precision Floating-Point Numbers

In 32-bit single-precision floating-point representation:

- The most significant bit is the *sign bit* (S), with 0 for positive numbers and 1 for negative numbers
- The following 8 bits represent *exponent* (E).
- The remaining 23 bits represents *fraction* (F).



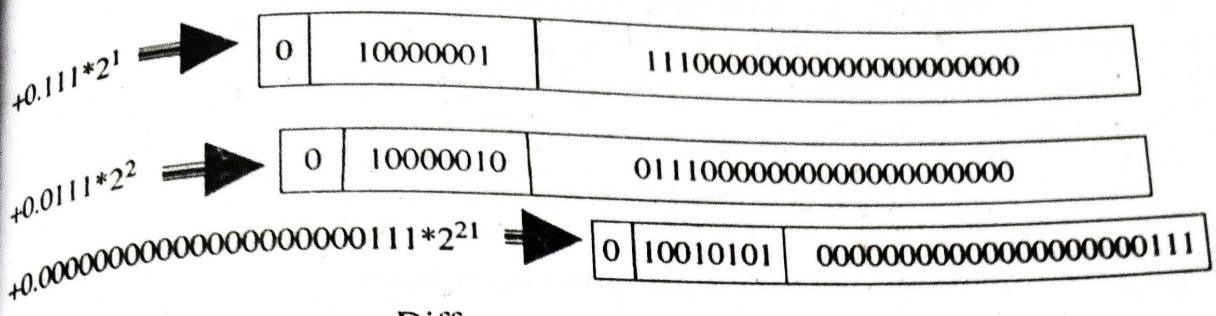
32-bit Single-Precision Floating-point Number

Q9. Explain, how to do floating point arithmetic addition and subtraction.

Ans : (Imp.

Floating-Point Arithmetic Addition/Subtraction

- The difficulty in adding two FP numbers stems from the fact that they may have different exponents
- Therefore, before adding two FP numbers, their exponents must be equalized, that Is, the mantissa of the number that has smaller magnitude of exponent must be aligned.



Different representation of an FP number

Steps Required to Add/Subtract Two Floating-Point Numbers

1. Compare the magnitude of the two exponents and make suitable alignment to the number with the smaller magnitude of exponent.
2. Perform the addition/subtraction.
3. Perform normalization by shifting the resulting mantissa and adjusting the resulting exponent.

Example Consider adding the two FP numbers $1.1100 * 2^4$ and $1.1000 * 2^2$.

- Alignment: $1.1000 * 2^2$ has to be aligned to $0.0110 * 2^4$
- Addition: Add the two numbers to get $10.0010 * 2^4$.
- Normalization: The final normalized result is $0.1000 * 2^6$ (assuming 4 bits are allowed after the radix point).

Q10. Explain, floating point multiplication and division.

Ans :

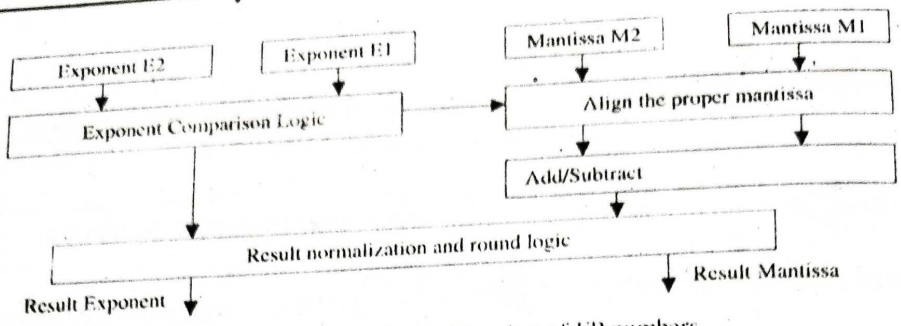
Multiplication Multiplication of a pair of FP numbers $X = mx * 2^a$ and $Y = m_y * 2^b$ is represented as $X * Y = (mx * my) * 2^{a+b}$

A general algorithm for multiplication of FP numbers consists of three basic steps. These are:

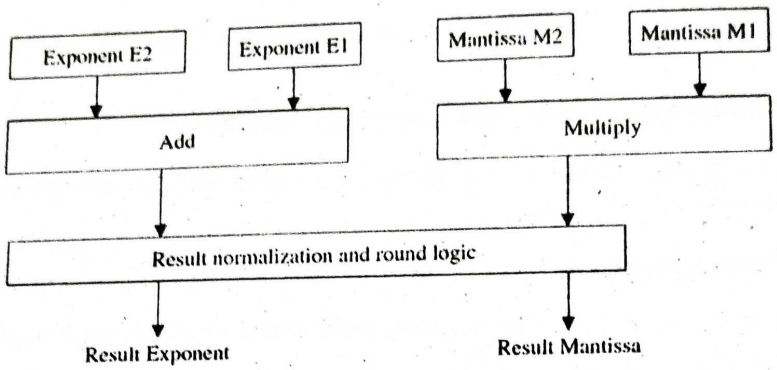
1. Compute the exponent of the product by adding the exponents together.
2. Multiply the two mantissas.
3. Normalize and round the final product.

Example Consider multiplying the two FP numbers X ¼ $1.000 * 2^{-2}$ and Y = – $1.010 * 2^{-1}$.

- Add exponents: $-2 + (-1) = -3$.
- Multiply mantissas: $1.000 * - 1.010 = -1.010000$.
- The product is $-1.0100 * 2^{-3}$.



Addition/subtraction of FP numbers



FP multiplication

Multiplication of two FP numbers can be illustrated using the schematic shown in Figure

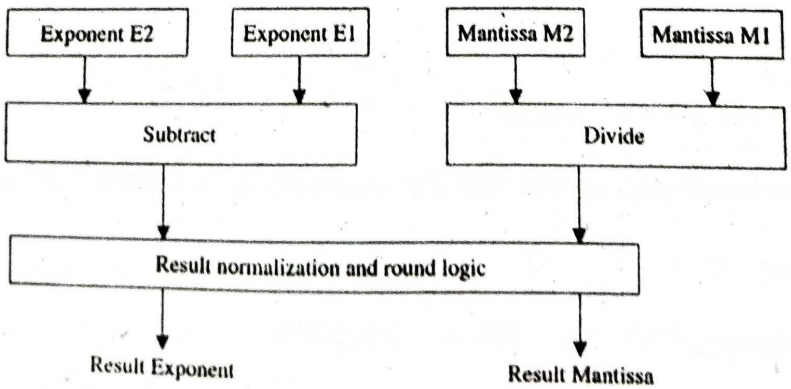
Division of a pair of FP numbers $X = mx * 2^a$ and $Y = my * 2^b$ is represented as $X/Y = (mx/my) * 2^{a\_b}$.

A general algorithm for division of FP numbers consists of three basic steps:

1. Compute the exponent of the result by subtracting the exponents.
2. Divide the mantissa and determine the sign of the result.
3. Normalize and round the resulting value, if necessary.

Example Consider the division of the two FP numbers $X = 1.0000 * 2^{-2}$ and $Y = -1.0100 * 2^{-1}$.

- Subtract exponents: $-2 - (-1) = -1$.
- Divide the mantissas: $1.0000 / -1.0100 = -0.1101$.
- The result is $-0.1101 * 2^{-1}$.
- Division of two FP numbers can be illustrated using the schematic shown in Figure.



FP division

Short Question and Answers

1. Procedure for Addition and Subtraction with signed-magnitude data.

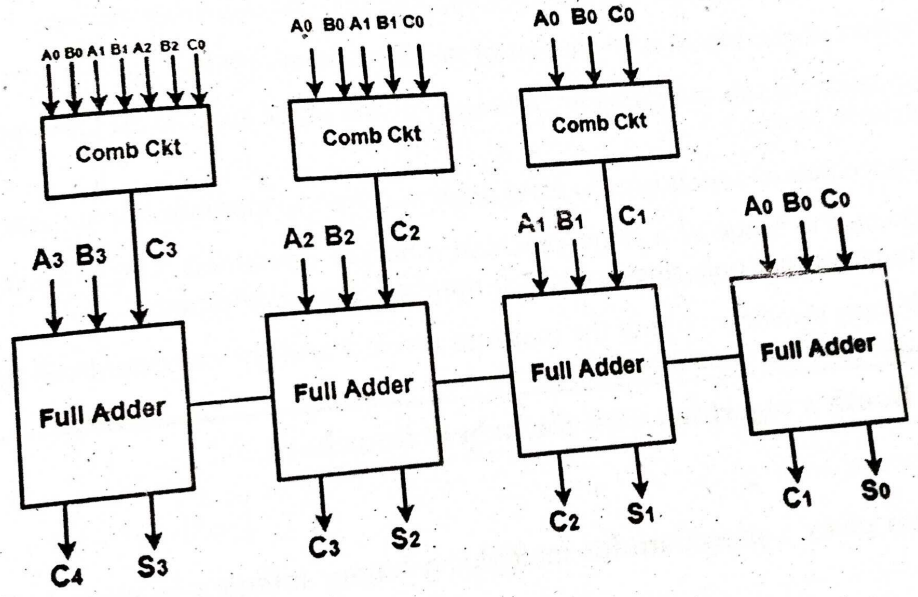
Ans :

- The two signs A, and B, are compared by an exclusive-OR gate.
- If the output of the gate is 0 the signs are identical; If it is 1, the signs are different.
- For an add operation, identical signs dictate that the magnitudes be added. For a subtract operation, different signs dictate that the magnitudes be added.
- The magnitudes are added with a microoperation EAU A + B, where EA is a register that combines E and A. The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred into the add-overflow flip-flop AVF.

2. Logic Diagram

Ans :

The logic diagram for carry look ahead adder is as shown below.



Carry Look Ahead Adder Working

The carry-in of any stage full adder depends only on the following two parameters-

- Bits being added in the previous stages
- Carry-in provided in the beginning

Now,

- The above two parameters are always known from the beginning.
- So, the carry-in of any stage full adder can be evaluated at any instant of time.
- Thus, any full adder need not wait until its carry-in is generated by its previous stage full adder.

3. Merits and Demerits carry lock ahead.

Ans :

Advantages of Carry Look Ahead Adder

The advantages of carry look ahead adder are,

1. It generates the carry-in for each full adder simultaneously.
2. It reduces the propagation delay.

Disadvantages of Carry Look Ahead Adder

The disadvantages of carry look ahead adder are,

1. It involves complex hardware.
2. It is costlier since it involves complex hardware.
3. It gets more complicated as the number of bits increases.

4. Multiplication Operation

Ans :

- The multiplication of two floating-point numbers requires that we multiply the mantissas and add the exponents.
- No comparison of exponents or alignment of mantissas is necessary.
- The multiplication of the mantissas is performed in the same way as in fixed-point to provide a double-precision product.
- The double-precision answer is used in fixed-point numbers to increase the accuracy of the product.
- In floating-point, the range of a single-precision mantissa combined with the exponent is usually accurate enough so that only single-precision numbers are maintained.
- Thus the half most significant bits of the mantissa product and the exponent will be taken together to form a single-precision floating-point product.

5. Explain the Booth's algorithm with the help of flowchart.

Ans :

Booth algorithm gives a procedure for multiplying binary integers in signed- 2's complement representation.

It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} - 2^m$.

For example, the binary number 001110 (+14) has a string 1's from 2^3 to 2^1 (k=3, m=1). The number can be represented as $2^{k+1} - 2^m$. $= 2^4 - 2^1 = 16 - 2 = 14$. Therefore, the multiplication M X 14, where M is the multiplicand and 14 the multiplier, can be done as M X 2^4 – M X 2^1.

- Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.
- As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of partial product.

6. Division Algorithm

Ans :

Position the divisor appropriately with respect to the dividend and performs a subtraction.

1. If the remainder is zero or positive, a quotient bit of 1 is determined, the remainder is extended by another bit of the dividend, the divisor is repositioned, and another subtraction is performed.
2. If the remainder is negative, a quotient bit of 0 is determined, the dividend is restored by adding back the divisor, and the divisor is repositioned for another subtraction.

7. Floating point multiplication and division.

Ans :

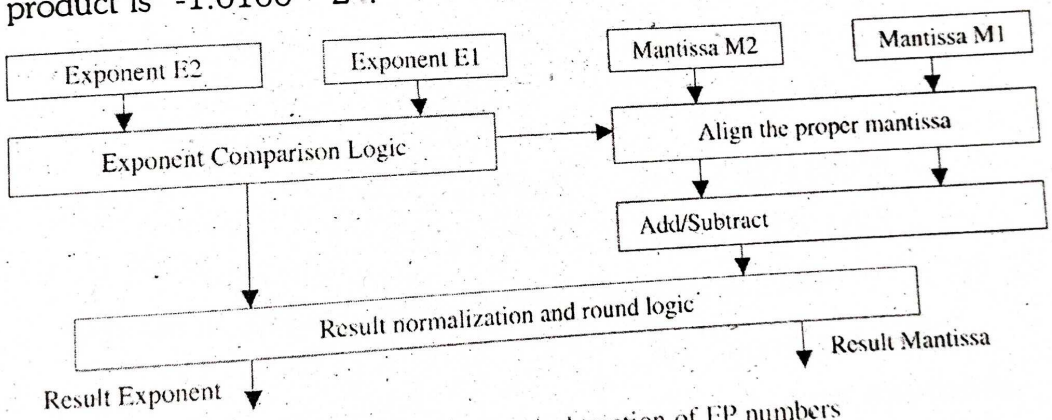
Multiplication Multiplication of a pair of FP numbers $X = mx * 2^a$ and $Y = m_y * 2^b$ is represented as $X * Y = (mx * my) * 2^{a+b}$

A general algorithm for multiplication of FP numbers consists of three basic steps. These are:

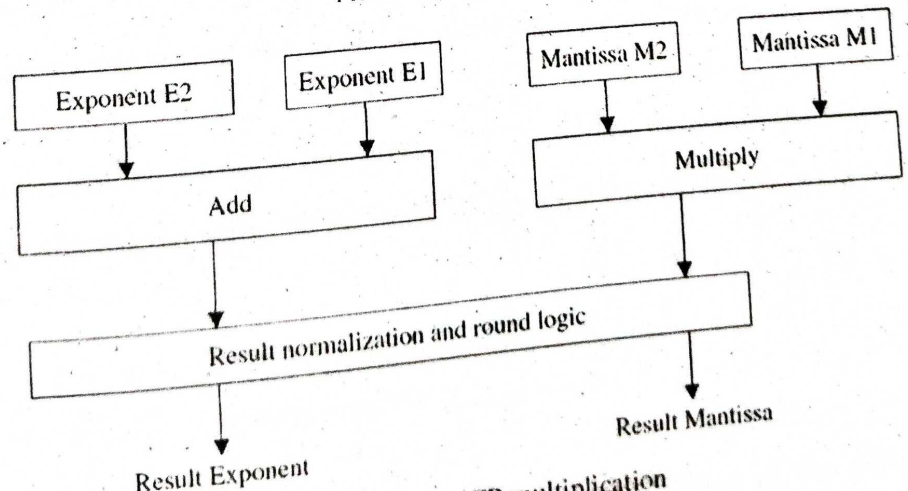
1. Compute the exponent of the product by adding the exponents together.
2. Multiply the two mantissas.
3. Normalize and round the final product.

Example Consider multiplying the two FP numbers X ¼ 1.000 * 2^{-2} and Y = – 1.010 * 2^{-1}.

- Add exponents: -2 + (-1) = -3.
- Multiply mantissas: 1.000 * - 1.010 = -1.010000.
- The product is -1.0100 * 2^{-3}.



Addition/subtraction of FP numbers



FP multiplication

Multiplication of two FP numbers can be illustrated using the schematic shown in Figure

Division of a pair of FP numbers $X = mx * 2^a$ and $Y = my * 2^b$ is represented as $X/Y = (mx/my) * 2a\_b$.

A general algorithm for division of FP numbers consists of three basic steps:

1. Compute the exponent of the result by subtracting the exponents.
2. Divide the mantissa and determine the sign of the result.
3. Normalize and round the resulting value, if necessary.

Choose the Correct Answers

1. When we add a two's complement, 4-bit, binary numbers 1101 and 0100, it would result in: [d]

(a) 1001 and no overflow
(b) 0001 and an overflow
(c) 1001 and an overflow
(d) 0001 and no overflow

2. What would be the 2's complement representation (in hexadecimal) of ("539)10? [b]

(a) 9E7
(b) DE5
(c) DBC
(d) ABE

3. Which of the following is performed by half adder? [b]

(a) Binary addition operation for 2 decimal inputs
(b) Binary addition operation for 2 binary inputs
(c) Decimal addition operation for 2 decimal inputs
(d) Binary addition operation for 2 binary inputs

4. The final addition sum of the numbers, 0110 & 0110 is _____ [a]

(a) 1101
(b) 1111
(c) 1001
(d) 1010

5. The delay reduced to in the carry look ahead adder is _____ [a]

(a) 5
(b) 8
(c) 10
(d) 2n

6. The product of -13 & 11 is _____ [b]

(a) 1100110011
(b) 1101110001
(c) 1010101010
(d) 1111111000

7. _____ constitute the representation of the floating number. [d]

(a) Sign
(b) Significant digits
(c) Scale factor
(d) All of the mentioned

8. The 32 bit representation of the decimal number is called as _____ [b]

(a) Double-precision
(b) Single-precision
(c) Extended format
(d) None of the mentioned

9. When 1101 is used to divide 100010010 the remainder is _____ [d]

(a) 101
(b) 11
(c) 0
(d) 1

10. The multiplier -6(11010) is recorded as _____ [a]

(a) 0-1-2
(b) 0-1+1-10
(c) -2-10
(d) None of the mentioned

Fill in the Blanks

1. The method used to reduce the maximum number of summands by half is _____
2. The sign followed by the string of digits is called as _____
3. In 32 bit representation the scale factor as a range of _____
4. The carry generation function: ci + 1 = yici + xici + xiyi, is implemented in _____
5. We make use of _____ circuits to implement multiplication.
6. The _____ is used to coordinate the operation of the multiplier.
7. For the addition of large integers, most of the systems make use of _____
8. The multiplier is stored in _____
9. The result that is smaller than the smallest number obtained is referred to as _____
10. In a normal n-bit adder, to find out if an overflow as occurred we make use of _____

Answers

1. Bit-pair recording
2. Mantessa
3. –128 to 127
4. Full adders
5. Fast adders
6. Control sequencer
7. Carry look-ahead adders
8. Shift register
9. Underflow
10. Xor gate