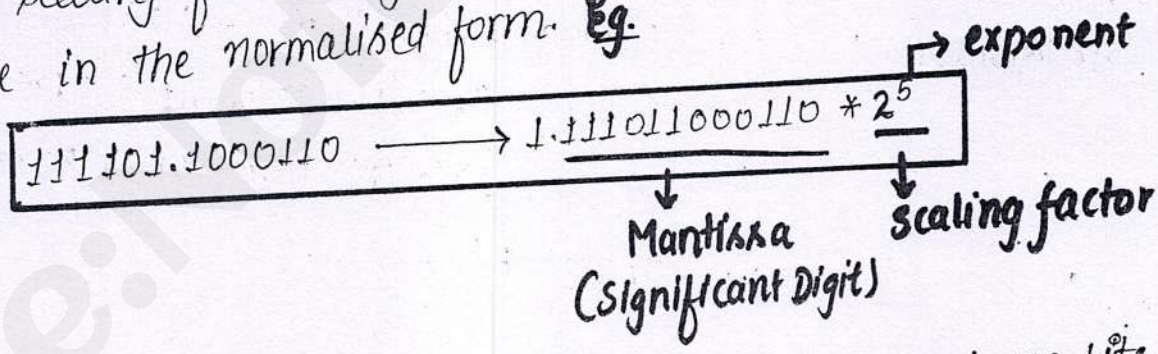
UNIT:-02

Arithmetic and Logic Unit

Floating Point Representation:-

To accomodate very large integers, and very small fractions, a computer must be able to represent numbers and operate on them in such a way that the position of the binary point is variable and is automatically adjusted as computation proceeds. In this case, the binary point is set to float, and the numbers are called floating point numbers. The floating point representation has three fields are:- sign, significant digits and exponent.

To represent the number in floating point format, first binary point is shifted to right of the first bit and the number is multiplied by the correct scaling factor to get the same value. The number is said to be in the normalised form. Eg.



The standards for representing floating point numbers in 32 bits and 64 bits have been developed by the institute of electrical and electronics engineers (IEEE).

It is also known as IEEE 754 standards.

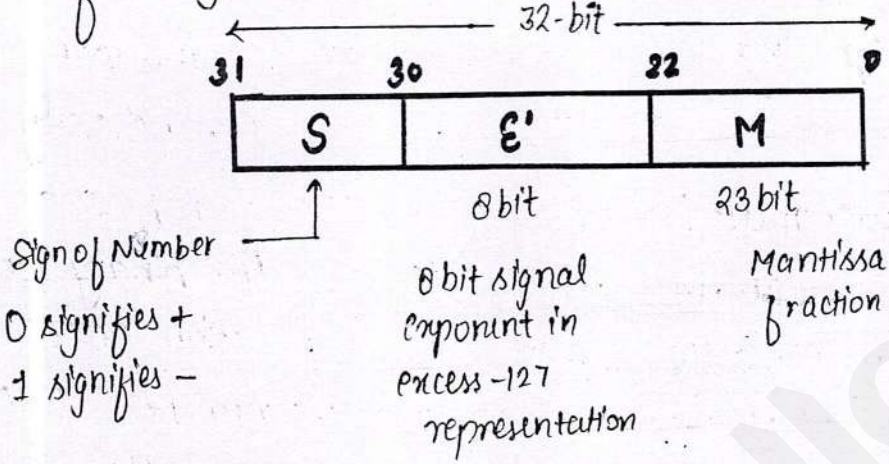
The 32-bit standard is called a single precision representation because it occupies a single 32-bit word. The 32-bits divided into three fields:- Field 1 ~~sign~~, sign-bit (1 bit).

Field 2 exponent (8-bit)

Field 3 Mantissa (23-bit).

Instead of the sign exponent (E), the value actually stored in the Exponent field is $E' = E +$ Base value.

In 32-bit floating point system, base is 127. Hence. $E' = E$ (scaling factor) $+127$

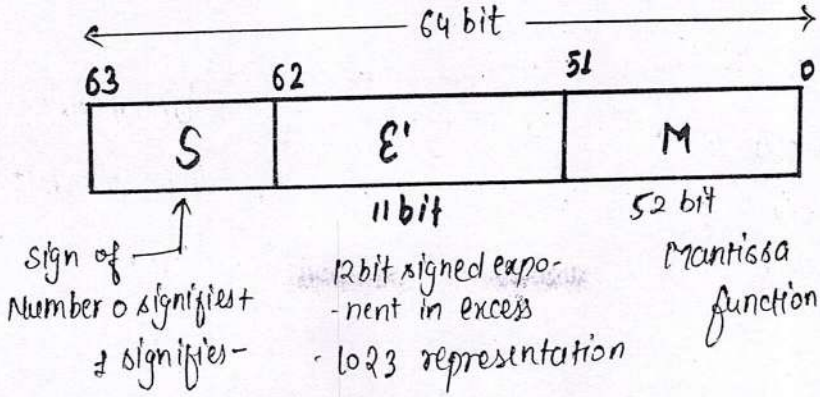


$E' = E + 127$

32 bit Representation

* The 64-bit representation is called a double precision Representation because it occupies two 32 bit words. The 64-bits are divided into three fields: field (1) sign - 1 bit
field (2) exponent → 11 bit
field (3) Mantissa → 52 bit.

* In double precision format, $E' = E + 1023$.



$E' = E + 1023$

★ Represent $(1259.125)_{10}$ in single and double precision format.

Soln:

2	1259	1
2	629	1
2	314	0
2	157	1
2	78	0
2	39	1
2	19	1
2	9	1
2	4	0
2	2	0
	1	

$.125 \times 2 = 0.250$ 0

$0.250 \times 2 = 0.500$ 0

$0.500 \times 2 = 1.000$ 1

$= (10011101011.001)_2$

$= 1.0011101011001 \times 2^{10}$

2	137	1
2	68	0
2	34	0
2	17	1
2	8	0
2	4	0
2	2	0
2	1	1
	0	

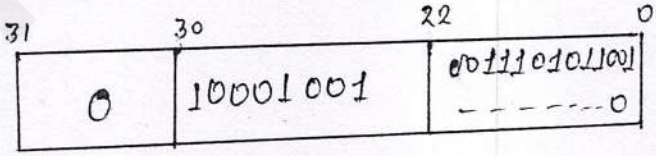
$\Rightarrow 10001001$

Single Precision Format

$E' = E + 127$

$E' = 10 + 127$

$\boxed{E' = 137}$



Double Precision Format

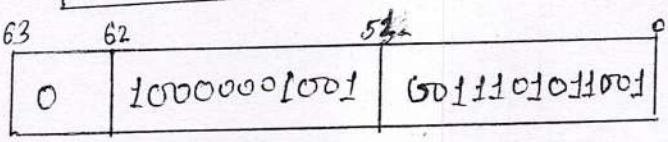
$E' = E + 1023$

$E' = 10 + 1023$

$\boxed{E' = 1033}$

$E' = 1033$

$E' = 10000001001$



② $(-309.1875)_{10}$

$309 = 100110101$

$0.1875 \times 2 = 0.3750 \quad 0$

$0.3750 \times 2 = 0.7500 \quad 0$

$0.7500 \times 2 = 1.5000 \quad 1$

$= -(100110101.0011)_2$

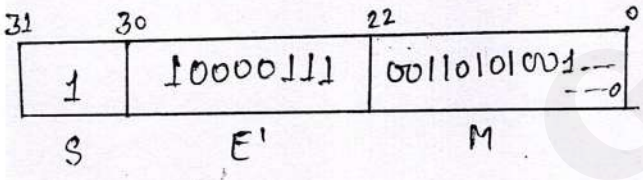
$= -1.00110101001 \times 2^8$

Single Precision

$E' = E + 127$

$E' = 8 + 127$

$\boxed{E' = 135} \quad = (10000111)_2$

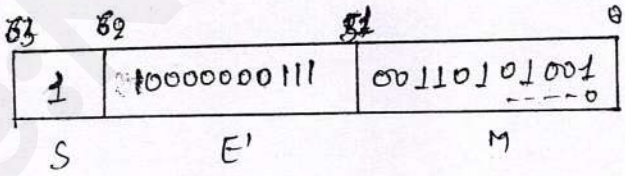


Double Precision

$E' = E + 1023$

$E' = 8 + 1023$

$\boxed{E' = 1031} \quad = (01000000111)_2$



③ $(-79.1258)_{10}$

$79 = (1001111)_2$

$0.1258 \times 2 = 0.2516 \quad 0$

$0.2516 \times 2 = 0.5032 \quad 0$

$0.5032 \times 2 = 1.0064 \quad 1$

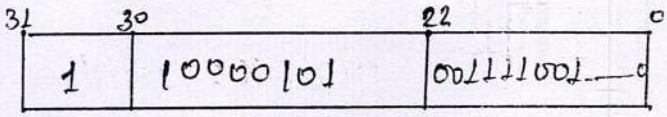
$(1001111.0011)_2 = 1.00111110011 \times 2^6$

Single Precision

$E' = E + 127$

$E' = 6 + 127$

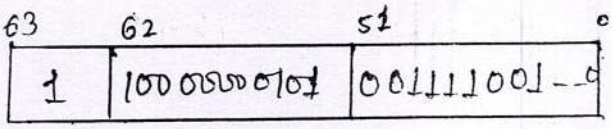
$\boxed{E' = 133} \quad = (10000101)_2$



Double Precision

$E' = 6 + 1023$

$E' = 1029 = (10000000101)_2$



(4.) $(1460.125)_{10}$

$1460 = (10110110100)$

$= (10110110100.001)_2$

$= (1.0110110100001 \times 2^{10}$

$.125 \times 2 = .250 \quad 0$

$0.250 \times 2 = .500 \quad 0$

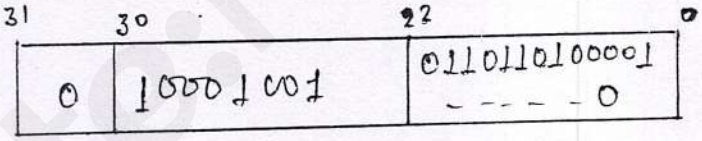
$0.500 \times 2 = 1.00 \quad 1$

Single Precision

$E' = E + 127$

$E' = 10 + 127$

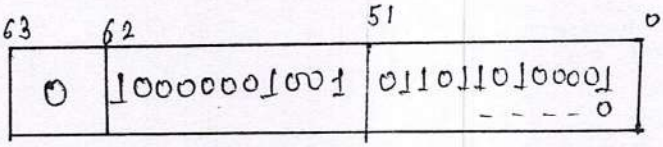
$\boxed{E' = 137} = (10001001)_2$



Double Precision

$E' = E + 1023 = 10 + 1023$

$\boxed{E' = 1033} = (10000001001)_2$



⑤ $(-1523.012)_{10}$

$(10111111001.0000)_2$

$1.0111111001 \times 2^{10}$

$E = 10$

$0.012 \times 2 = 0.024$

$0.024 \times 2 = 0.048$

$0.048 \times 2 = 0.096$

$0.096 \times 2 = 0.192$

$0.192 \times 2 = 0.384$

Single Precision

$E' = E + 127$

$E' = 10 + 127$

$E' = 137 = (10001001)_2$

1	10001001	01111110011000

Double Precision

$E' = E + 1023$

$E' = 1033 = (10000001001)_2$

1	10000001001	01111100111000

⑥ $(1486.125)_{10}$

$= (10111001110.001)_2$

$= 1.0111001110001 \times 2^{10}$

$E = 10$

Single Precision

$E' = E + 127$

$E' = 10 + 127$

$E' = 137 = (10001001)_2$

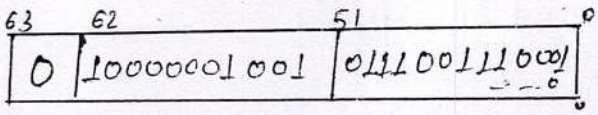
31	30 ... 23	22 ... 0
0	10001001	01110011100010

Double Precision

$E' = E + 1023$

$E' = 10 + 1023$

$E' = 1033 = (10000001001)_2$



(f) $(1243.725)_{10}$

$(10011011011.1011)_2$

$= 1.0011011011011 \times 2^{10}$

$E = 10$

Single Precision

$E' = E + 127$

$E' = 10 + 127$

$E' = 137 = (10001001)_2$

0	10001001	00110110111011

Double Precision

$E' = E + 1023$

$E' = 10 + 1023$

$E' = 1033 = (10000001001)_2$

0	10000001001	0011011011011

(g) $(-176.375)_{10}$

$= (10110000.011)_2$

$= 1.0110000011 \times 2^7$

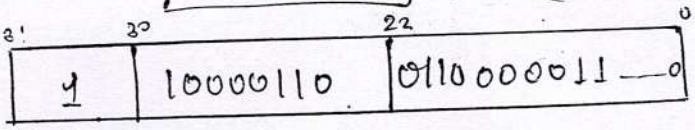
$E = 7$

Single Precision

$E' = E + 127$

$E' = 7 + 127$

$\boxed{E' = 134} = (10000110)_2$

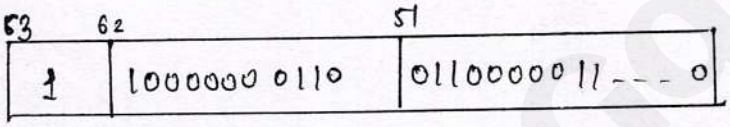


Double Precision

$E' = E + 1023$

$E' = 7 + 1023$

$\boxed{E' = 1030} = (10000000110)_2$



(2) $(-1976.453)_{10}$

$= (11110111000.0111)_2$

$= 1.1110111000.0111 \times 2^{10}$

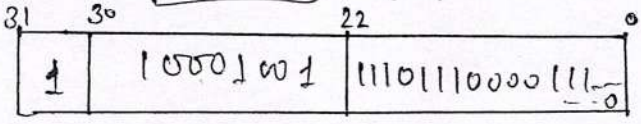
$\boxed{E = 10}$

Single Precision

$E' = E + 127$

$E' = 10 + 127$

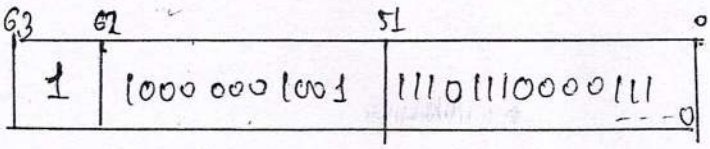
$\boxed{E' = 137} = (10001001)_2$



Double Precision

$E' = E + 1023 = 10 + 1023$

$\boxed{E' = 1033} = (10000001001)_2$



Parallel Adder

A single full-adder is capable of adding two one-bit numbers and an input carry. In order to add binary numbers with more than one bit, additional full-adders must be employed.

A n-bit, parallel adder can be constructed using number of full adder circuits connected in parallel.

Figure shows the block diagram of n-bit parallel adder using n-number of full adder circuits connected in cascade i.e., the carry output of each adder is connected to the carry input of the next higher-order adder.

It should be noted that either a half adder can be used for least significant position or the carry input of a full adder is made 0 because there is no carry into the least significant bit position.

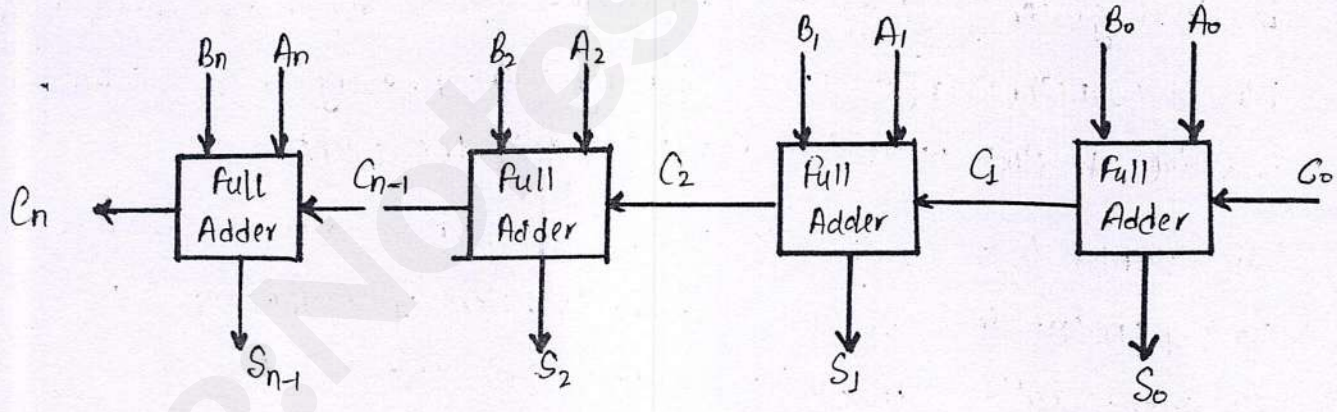


Fig. 2.6.11 :- Block diagram of n-bit parallel adder

Parallel Subtractor: The subtraction of binary numbers can be done most conveniently by means of complements.

The subtraction A-B can be done by taking 2's complemunt of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits.

The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry to get 2's complement as shown in the fig.

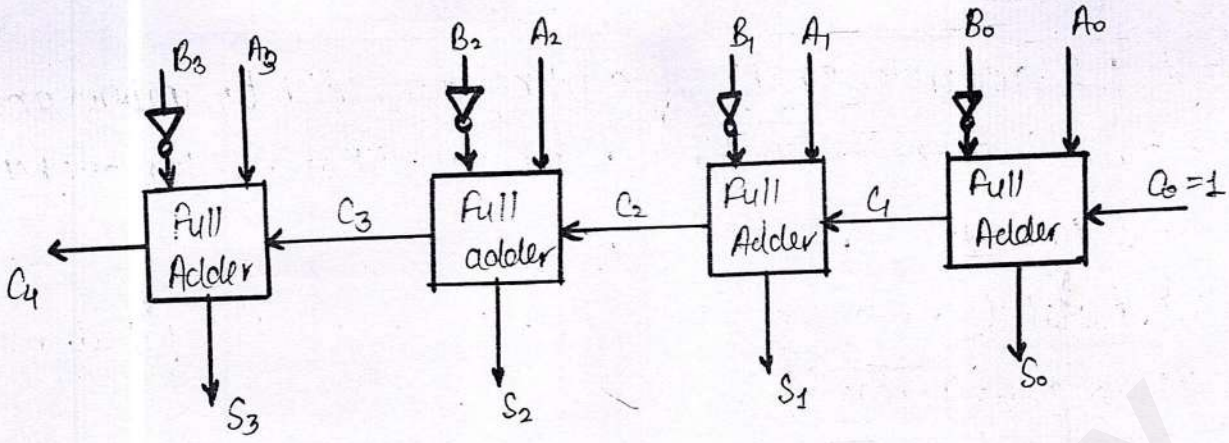


Fig. 4-bit parallel subtractor

Binary Incrementor :-

We have seen that the increment micro-operation adds one to a number in register. This micro-operation can be easily implemented with a binary counter. ~~every time~~ In a binary counter, every time the count enable is active, the clock pulse transition increments the count.

The counter is a sequential circuit. The Fig. shows a combinational circuit to implement increment micro-operation. Here, one of the inputs to the least significant half-adder (HA) is connected to logic 1 and the other input is connected to the least significant bit of the number to be increment. The output carry from lower order half adder is connected to the one of the inputs of the next higher order half adder. Such circuits gets the four bit number from bits A_0 through A_3, adds one to it, and generates the incremented output in S_0 through S_3. If the number to be incremented is 1111, the output carry C_4 is generated and we get 0000 at S_0 through S_3.

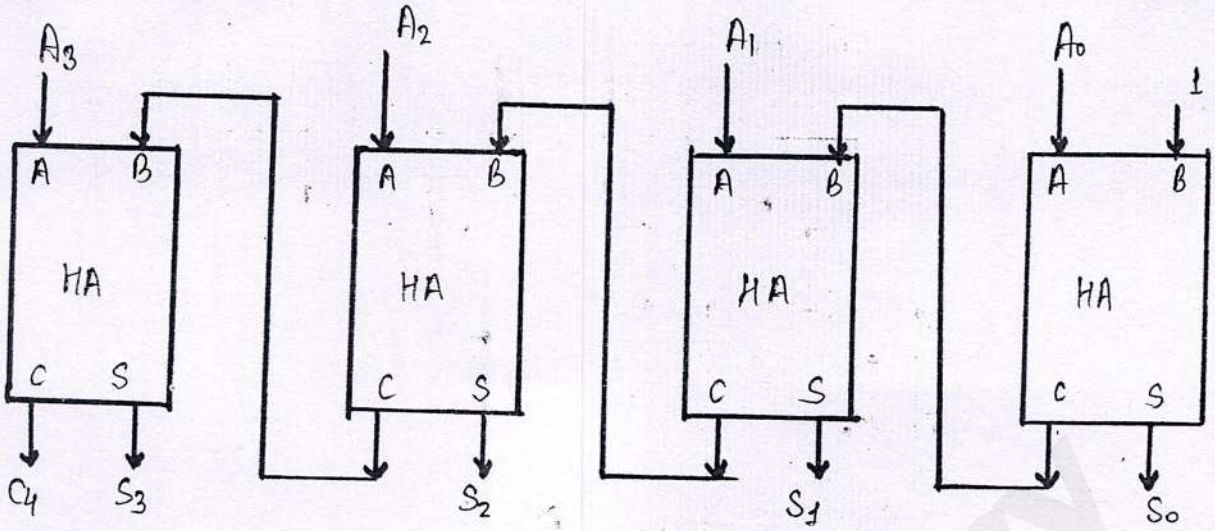


Fig: 2.6.17. 4-bit Incrementor using combinational circuit

Array Multiplier: The multiplication process of binary number is similar to decimal no.s. Actually binary multiplication is simple than decimal multiplication because it involves 0 and 1 only.

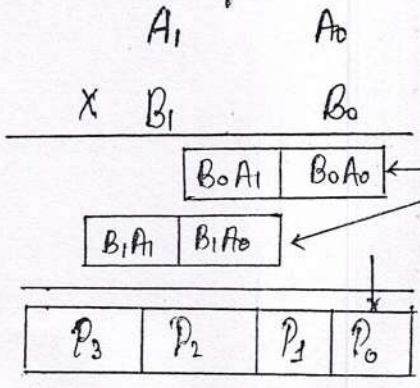
For multiplication in binary no.s, it wes n-shifts and ADDS to multiply n-bit binary number.

The combinational logic circuit implemented to perform such multiplication is called combinational multiplier or array multiplier.

Let us generalize the multiplication process for 2X2 multiplier for two unsigned 2-bit no.s - multiplicant $A = A_1 A_0$ and multiplier $B = B_1 B_0$.

The multiplication process involves multiplication of 2-bit no.s and addition of 2-bit no.s.

The multiplication of two bits can be implemented using 2 input AND Gate whereas addition of 2-bits can be implemented using half Adder.



Multiplication Process

Partial products

$P_0 = B_0A_0$

$P_1 = B_0A_1 + B_1A_0$

$P_2 = B_1A_1$ + Carryout of P_1

P_3 = carryout of P_2

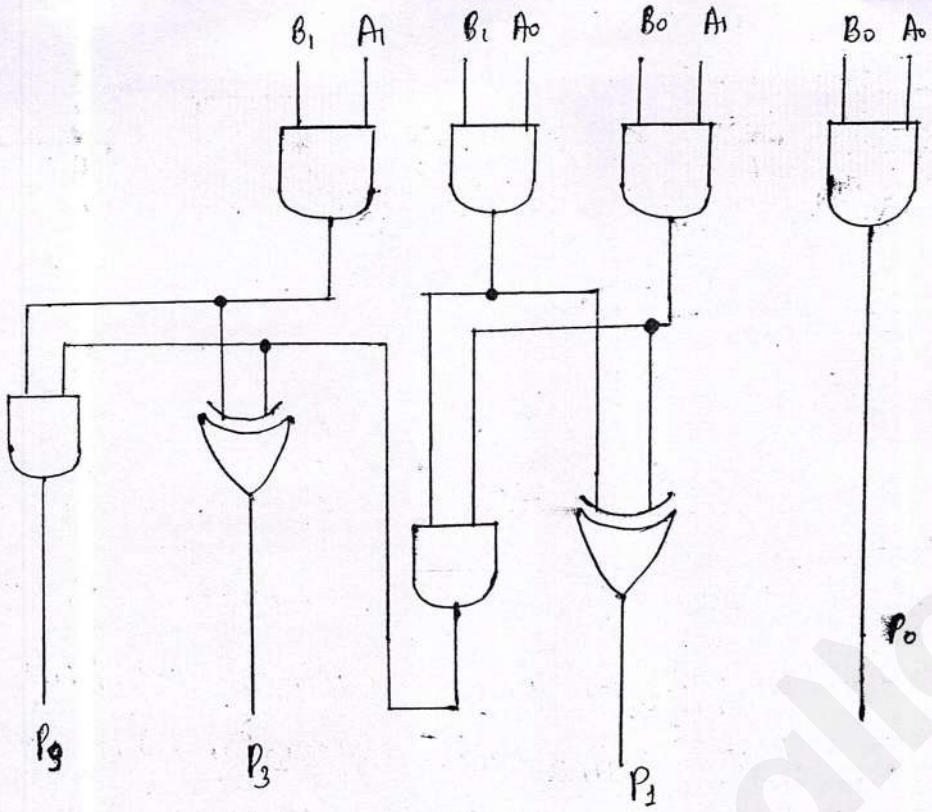
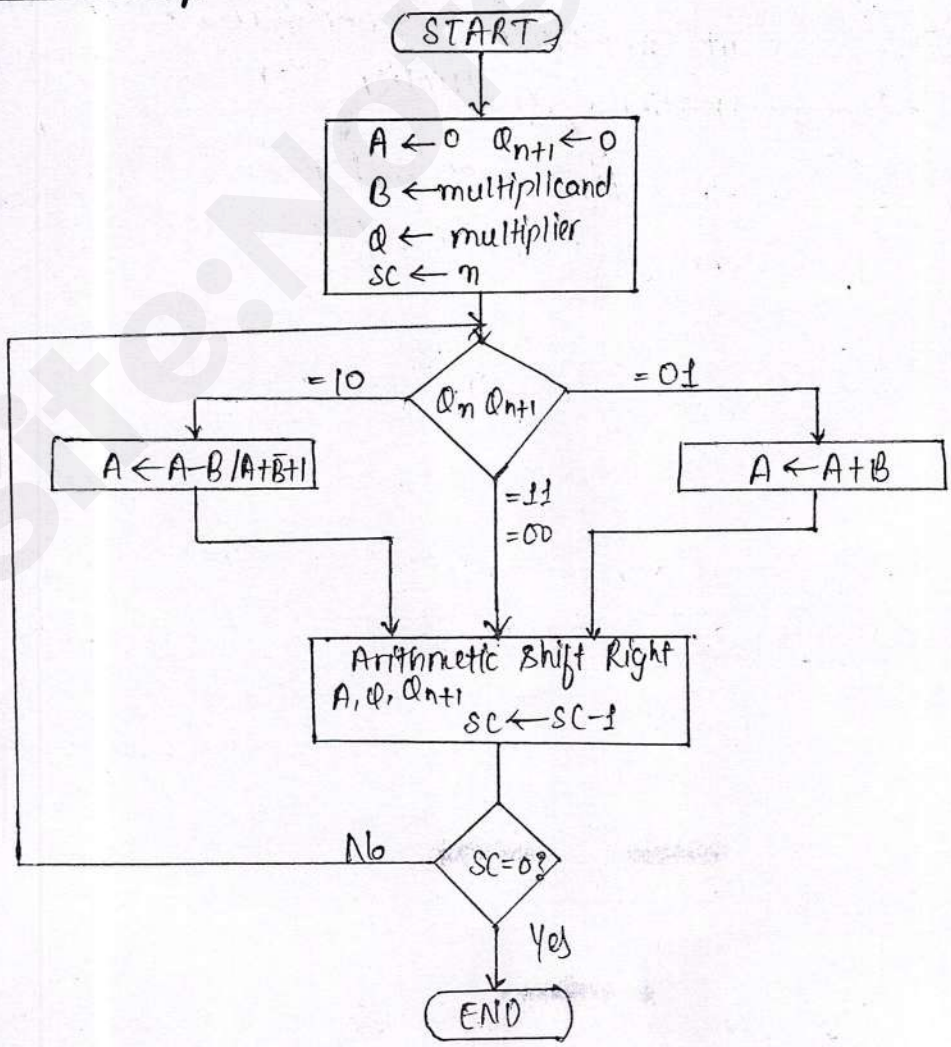


Fig 2.10.2 2x2 bit combinational array multiplier

Booth Multiplication



Algorithm:-

Step 1:- Load $A = 0$; $Q_{n+1} = 0$

B = Multiplicand

Q = multiplier

$SC = n$

Step 2:- Check the status of $Q_n Q_{n+1}$.

If $Q_n Q_{n+1} = 10$ perform $A \leftarrow A - B$

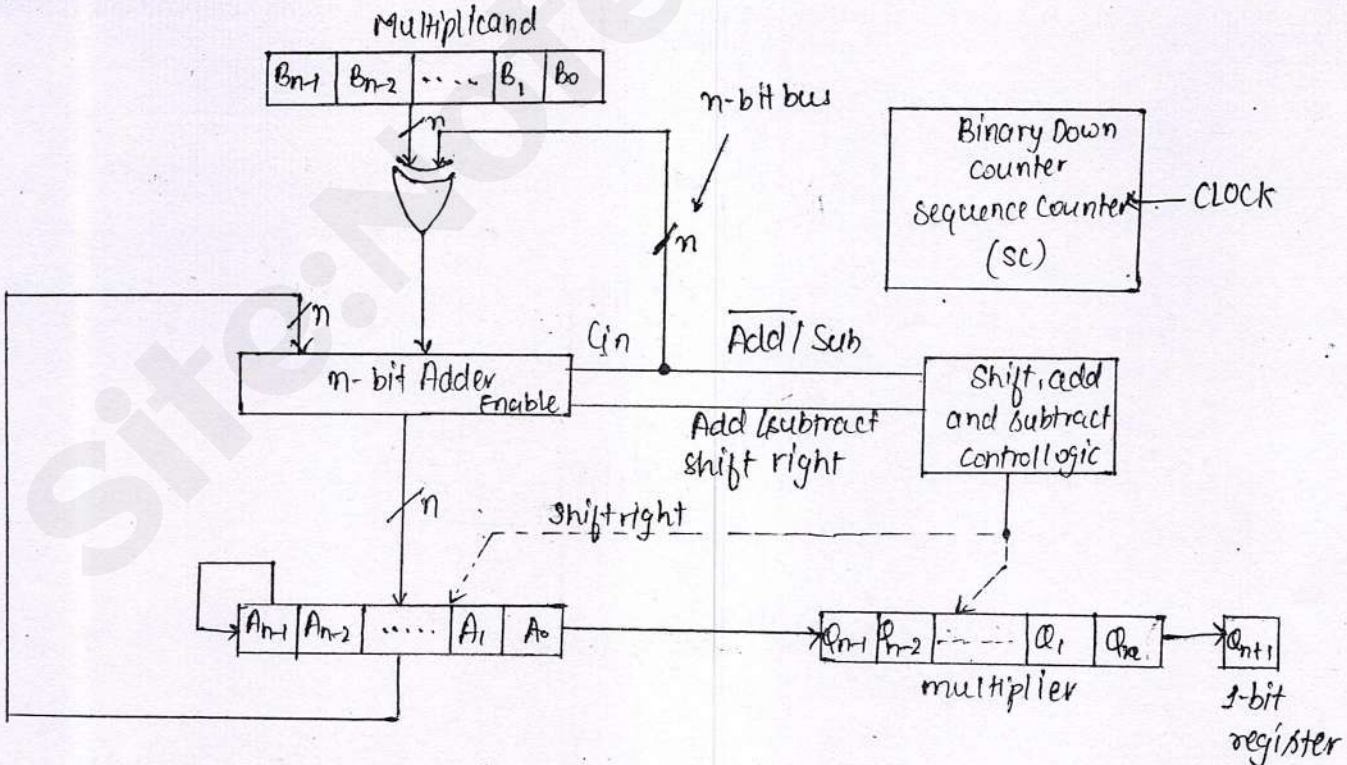
If $Q_n Q_{n+1} = 01$ perform $A \leftarrow A + B$

Step 3:- Arithmetic shift right: A, Q_n, Q_{n+1}

Step 4:- Decrement sequence counter if not zero, repeat step 2 through 4.

Step 5:- Stop.

Algorithm



- Initial settings: $A \leftarrow 0$ and $Q_{n+1} = 0$

-:- Hardware implementation of signed binary multiplication

(1) Evaluate

(i) 9*17

B = 9 = 001001 $\quad \overline{B}+1 = 110111$

Q = 17 = 010001

Q_n	Q_{n+1}	B = 001001 $\overline{B}+1$ = 110111	A	Q	Q_{n+1}	SC
		Initial	000000	010001	0	110
1	0	$A \leftarrow A+\overline{B}+1$	110111			
			110111			
		Ashr(A,Q,Q_{n+1})	111011	101000	1	101
0	1	$A \leftarrow A+B$	001001			
			000100			
		Ashr(A,Q,Q_{n+1})	000010	010100	0	100
		Ashr(A,Q,Q_{n+1})	000001	001010	0	011
0	0	Ashr(A,Q,Q_{n+1})	000000	100101	0	010
0	0	Ashr(A,Q,Q_{n+1})	000000	100101	0	010
1	0	$A \leftarrow A+\overline{B}+1$	110111			
			110111			
		Ashr(A,Q,Q_{n+1})	111011	110010	1	001
0	1	$A \leftarrow A+B$	001001			
			000100			
		Ashr(A,Q,Q_{n+1})	000010	011001	0	000

B * Q = 0000100011001

② $-13 * 16$

$B = -13 = 110011 \Rightarrow \bar{B}+1 = 001101$

$Q = 16 = 010000$

$+13 = 001101$

↓ 2's complement

$= 110010 + 1$

$-13 = 110011$

Q_n	Q_{n+1}	$B = 110011$ $\bar{B}+1 = 001101$	A	Q	Q_{n+1}	SC
		Initial	000000	010000	0	110
0	0	Ashr (A, Q, Q_{n+1})	000000	001000	0	101
0	0	Ashr (A, Q, Q_{n+1})	000000	000100	0	100
0	0	Ashr (A, Q, Q_{n+1})	000000	000010	0	011
0	0	Ashr (A, Q, Q_{n+1})	000000	000001	0	010
1	0	$A \leftarrow A+\bar{B}+1$	001101 001101			
		Ashr (A, Q, Q_{n+1})	000110	100000	1	001
0	1	$A \leftarrow A+B$	110011 111001			
		Ashr (A, Q, Q_{n+1})	111100	110000	0	000

$B * Q = 111100110000$

③ 18 * -14

B = 18 = 010010 $\quad \bar{B}+1 = 101101+1 = 101110$

Q = -14 = 110010

14 = 001110

↓ 2's complement

= 110001+1

= 110010

Q_n	Q_{n+1}	B = 010010, $\bar{B}+1$ = 101110	A	Q	Q_{n+1}	SC
		Initial	000000	110010	0	110
0	0	Ashr(A,Q,Q_{n+1})	000000	011001	0	101
1	0	A ← A+$\bar{B}$+1	101110			
		Ashr(A,Q,Q_{n+1})	110111	001100	1	100
0	1	A ← A+B	010010			
		Ashr(A,Q,Q_{n+1})	001001	100110	0	011
0	0	Ashr(A,Q,Q_{n+1})	000100	010011	0	010
1	0	A ← A+$\bar{B}$+1	101110			
		Ashr(AQQ_{n+1})	111000	001001	1	001
1	1	Ashr(A,Q,Q_{n+1})	111100	000100	1	000

B * Q = 111100000100

④ -12 * -11

B = -12 = 10100 $\quad \overline{B}+1 = 01011+1 = 01100$

Q = -11 = 10101

+12 = 01100 $\quad$ +11 = 01011

↓ 2's comp. $\quad$ ↓ 2's comp.

10011+1 $\quad$ 10100+1

-12 = 10100 $\quad$ -11 = 10101

Q_n	Q_{n+1}	$B = 10100$ $\overline{B}+1 = 01100$	A	Q	Q_{n+1}	SC
		Initial	00000	10101	0	101
1	0	$A \leftarrow A+\overline{B}+1$	01100			
			01100			
		Ashr (A,Q,Q_{n+1})	00110	01010	1	100
0	1	$A \leftarrow A+B$	10100			
			11010			
		Ashr(A,Q, Q_{n+1})	11101	00101	0	011
1	0	$A \leftarrow A+\overline{B}+1$	01100			
			01001			
		Ashr(A,Q,Q_{n+1})	00100	10010	1	010
0	1	$A \leftarrow A+B$	10100			
			11000			
		Ashr(A,Q,Q_{n+1})	11100	01001	0	001
1	0	$A \leftarrow A+\overline{B}+1$	01100			
			01000			
		Ashr (A,Q,Q_{n+1})	00100	00100	1	000

B * Q = 00100 00100

Look Ahead Carry Adder (CLA)
↳ Carry Look Ahead.

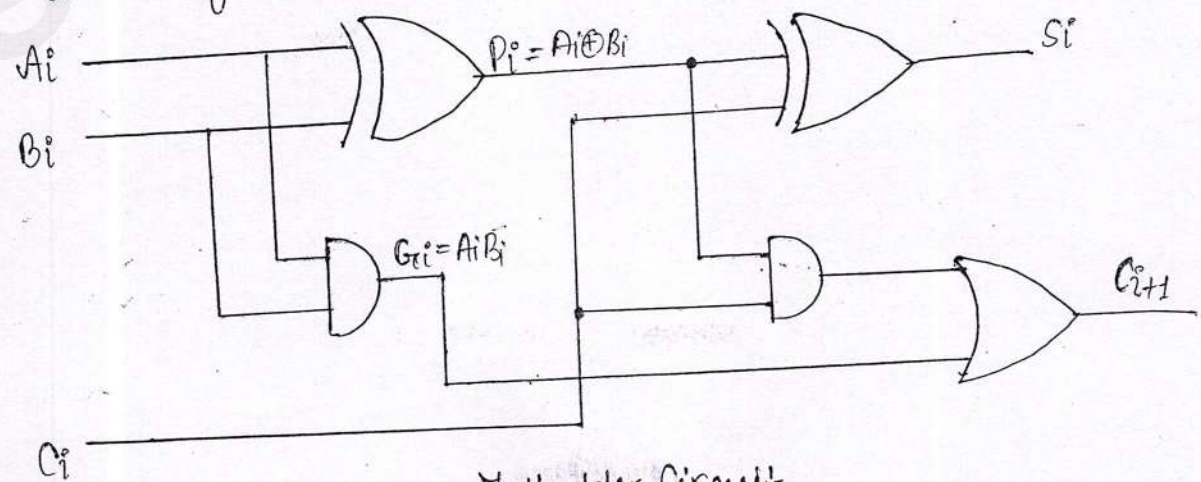
* It is also known as fast adder. / high speed adder.
* In ripple carry adder, the sum and carry outputs of any stage cannot be produced until the input carry occurs, this leads to a time delay in the addition process.
* This delay is known as carry propagation delay.

Considering the example:

$$\begin{array}{r} 0101 \\ 0011 \\ +\quad 1000 \\ \hline \end{array}$$

* Addition of the LSB position produces a carry into the second position.
* This carry, when added to the bits of the second position produces a carry for the third position. The latter carry when added to the bits of third position, produces a carry to the last position.
* One method of speeding up of this process by eliminating interstage carry delay is called look ahead carry addition.
* It uses two functions - carry generate (G_i) and carry propogate (C_i).

Consider the full adder which is designed with the help of two half adder



Fulladder Circuit

* We define two functions carry generate and carry propagate as follows:

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i \quad \} \text{ —— (1)}$$

* The output sum and carry can be expressed as:

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i \quad \} \text{ —— (11)}$$

* G_i is called carry generate and it produce a carry when both A_i and B_i are 1, regardless of the input carry.
* P_i is called carry propagate because it is term associated with propogation of carry G_i to C_{i+1}.
* for example, 4 stage carry look aheat adder can be defined as follows.
* For initial condition, $C_i = C_0$. which is equal to zero.

Put $i=0$ in eqn (2)

$$C_{i+1} = G_i + P_i C_i \quad \} \text{ —— (3.)}$$
$$C_1 = G_0 + P_0 C_0$$

$i=0$;

where $G_0 = A_0 B_0$ and $P_0 = A_0 \oplus B_0$ and $C_0 = 0$

Hence → $C_1 = G_0 + P_0 C_{in}$

now put $i=1$ in eqn (2)

$i=1$

$$C_2 = G_1 + P_1 C_1$$
$$C_2 = G_1 + P_1 (G_0 + P_0 C_{in})$$
$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_{in} \text{ —— (4)}$$

now put $i=2$ in eqn (2)

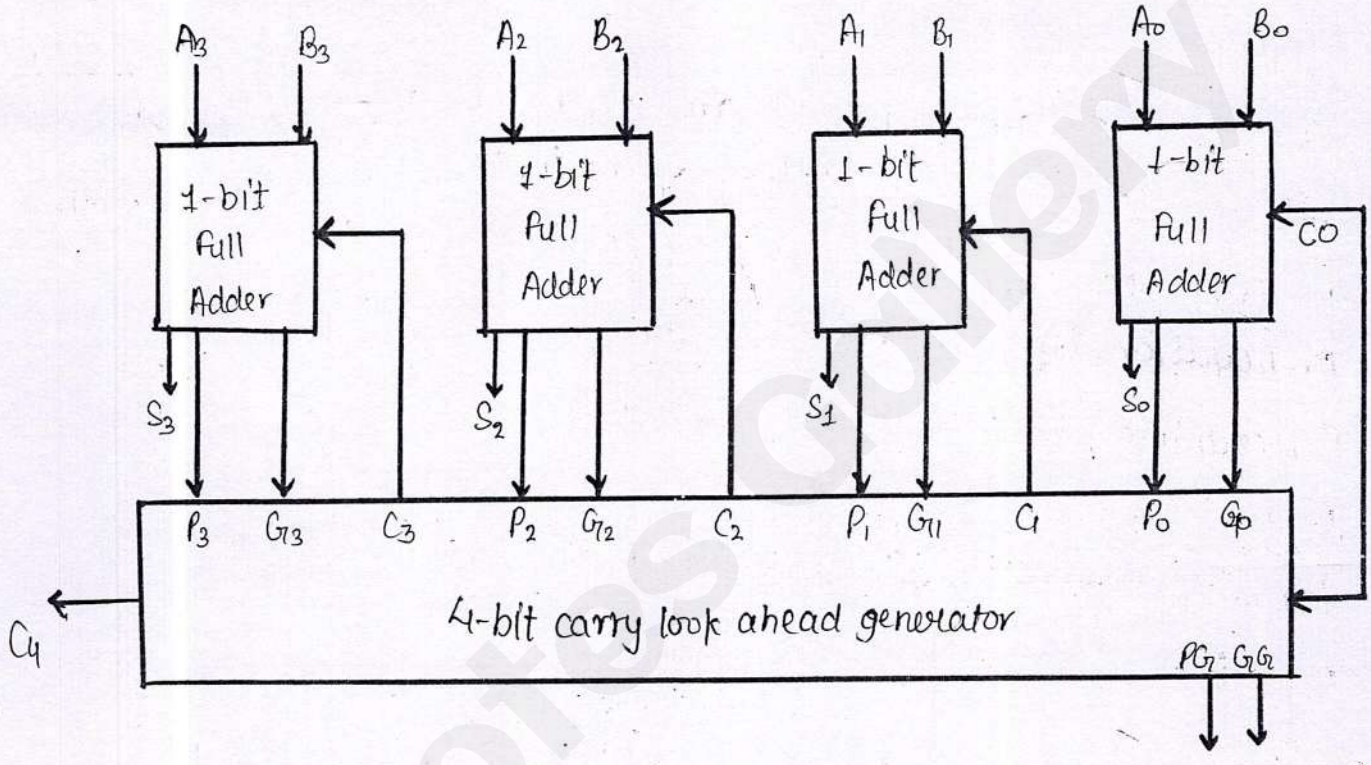
$$C_3 = G_2 + P_2 C_2$$
$$C_3 = G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_{in})$$
$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in} \text{ —— (5.)}$$

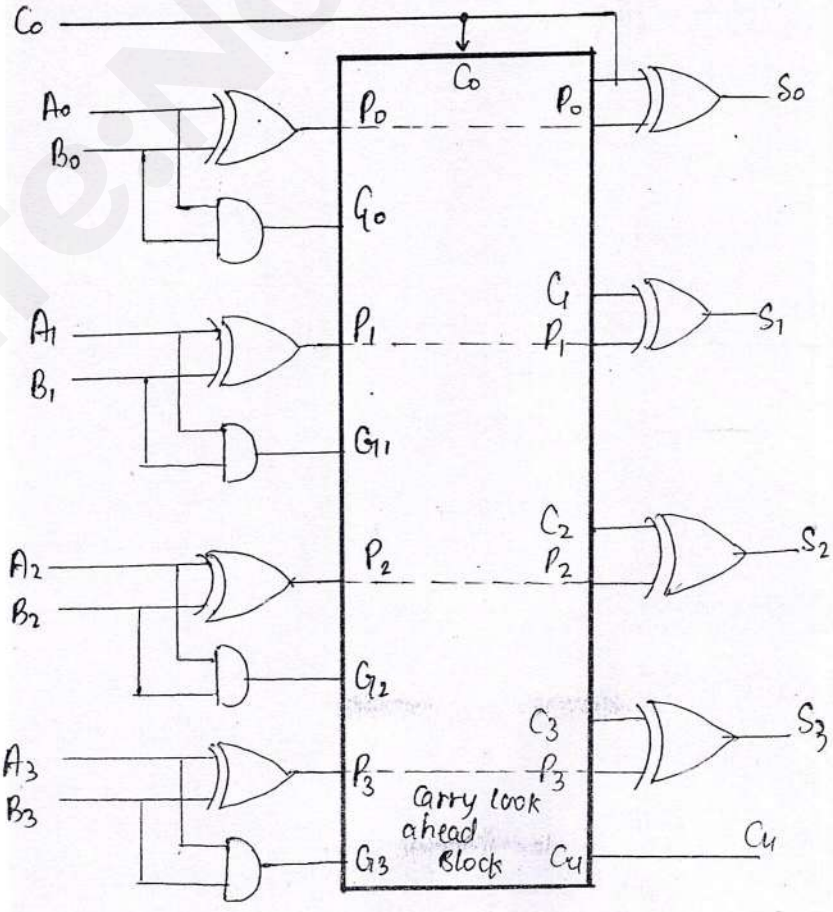
now put $i=3$ eqn (2)

$$C_4 = G_{13} + P_3 C_3$$

$$C_4 = G_{13} + P_3 (G_{12} + P_2 G_{11} + P_2 P_1 G_{10} + P_2 P_1 P_0 C_{in})$$

$$C_4 = G_{13} + P_3 G_{12} + P_3 P_2 G_{11} + P_3 P_2 P_1 G_{10} + P_3 P_2 P_1 P_0 C_{in} \quad \text{—— (6)}$$

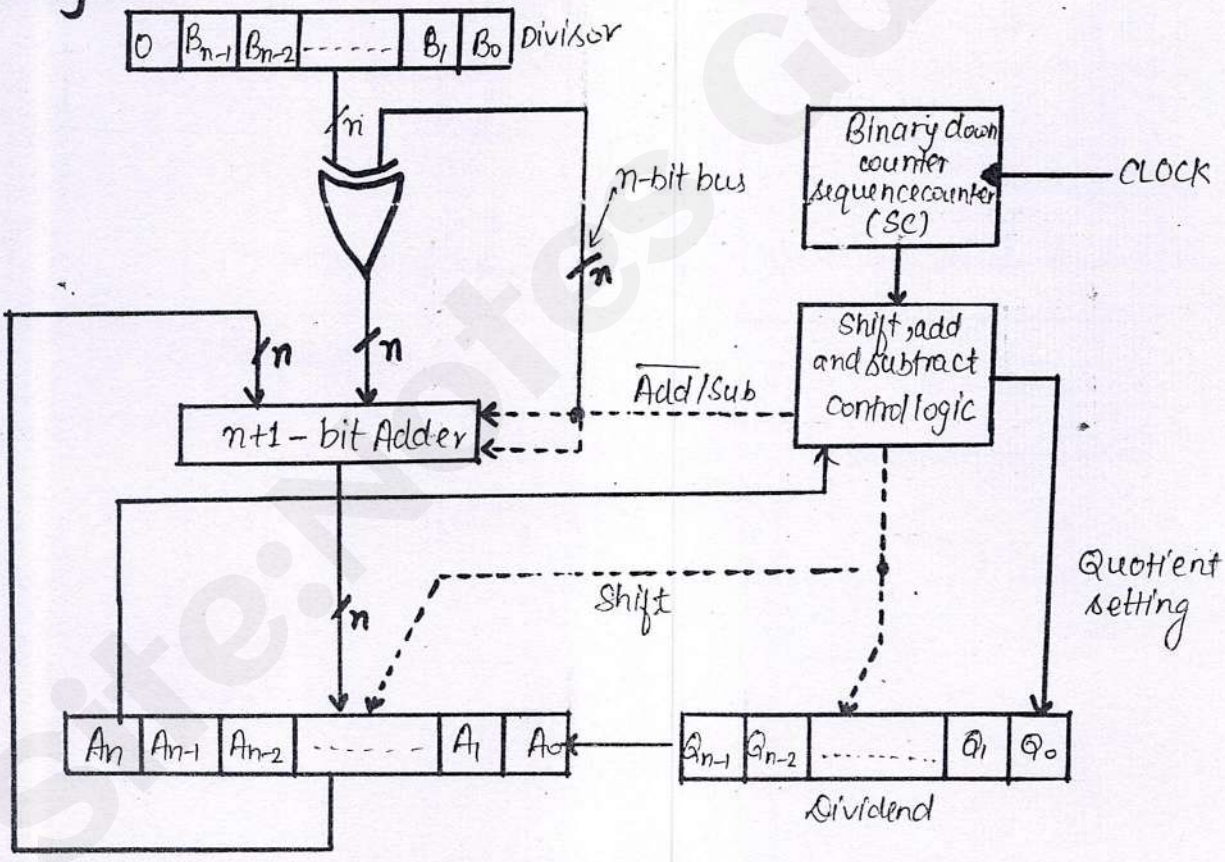




Booth's Division Algorithm

The division process for binary number is similar to decimal number. In division process, first the bits of the dividend are examined from left to right until the set of bits examined represents a no. of number greater than or equal to the divisor until this condition occurs. 0's are placed in the qc quotient. from left to right when the condition is satisfied one is placed in the quotient and the divisor is subtracted from the partial dividend. This result is referred as partial remainder.

Restoring Booth's Division Algorithm



Hardware to implement binary division

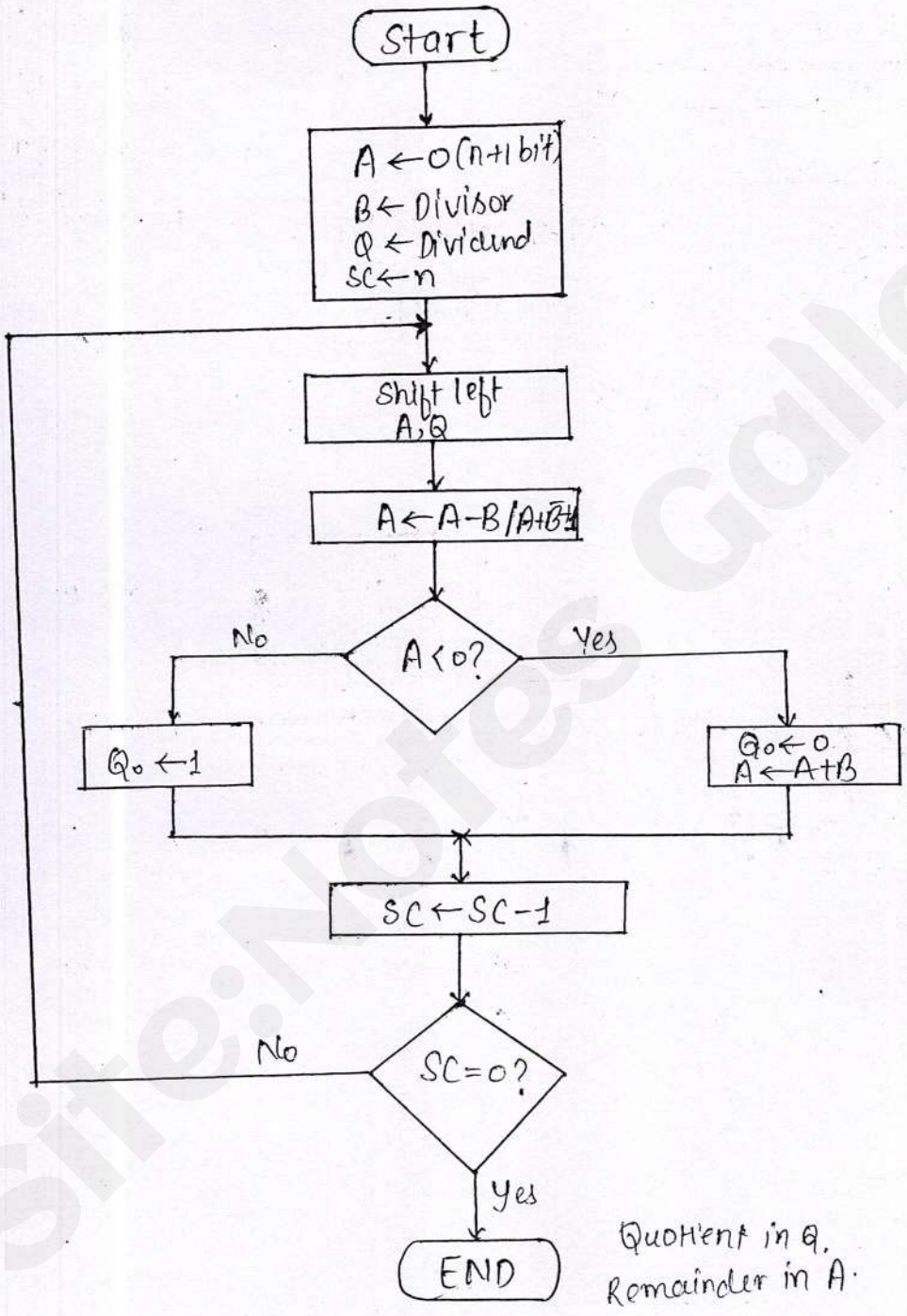
Division operation steps:

1. Shift A and Q left one binary operation. position.
2. Subtract divisor (i.e. add 2's complement of divisor (B)) from A and place answer back in A (A ← A-B)
3. If the sign bit of A is 1, set Q_0 to 0 and add divisor back to A

(that is, restore A); otherwise, set Q_0 to 1.

4. Repeat steps 1, 2 and 3 n times.

A flowchart for division operation is as shown in fig. 2.11.3



∴ Flowchart for restoring division algorithm

Perform the division of the following nos using restoring division algorith,

1. Dividund = 1010 and Divisor = 0011

Divisor(B) = 0011

Dividend (Q) = 1010

B = 00011

$\bar{B}+1$ = 11101

Operation	A	Q	SC
initial	00000	1010	100
Shift left(A,Q)	00001	0100	
$A \leftarrow A+\bar{B}+1$	11101		
	11110		
Set $Q_0 \leftarrow 0$	00011		
$A \leftarrow A+B$	00010		011
Shift left(A,Q)	00101	1000	
$A \leftarrow A+\bar{B}+1$	11101		
	00010		
Set $Q_0 \leftarrow 0$	00011		
$A \leftarrow A+B$	00010		010
Shift left(A,Q)	00101	0001	
$A \leftarrow A+\bar{B}+1$	11101		
	00010		
Set $Q_0 \leftarrow 1$			001
Shift Left(A,Q)	00100	0011	
$A \leftarrow A+\bar{B}+1$	11101		
	00001		
			000

Ans Quotient = 0011

Remainder = 00001

(ii) 17÷3

B = 3 = 00011

Q = 17 = 10001

B = 00011

$\bar{B}+1$ = 11101

Operation	A	Q	SC
initial	000000	10001	101
Shift Left (A,Q)	00001	10010	
$A \leftarrow A+\bar{B}+1$	11101		
	①11110		
Set $Q_0 \leftarrow 0$	00011		
$A \leftarrow A+B$	00001		100
Shift left (A,Q)	00010	00100	
$A \leftarrow A+\bar{B}+1$	11101		
	①11111		
Set $Q_0 \leftarrow 0$	00011		
$A \leftarrow A+B$	00010		011
Shift Left (A,Q)	00100	01001	
$A \leftarrow A+\bar{B}+1$	11101		
	⓪00001		010
Set $Q_0 \leftarrow 1$			
Shift Left (A,Q)	00010	10010	
$A \leftarrow A+\bar{B}+1$	11101		
	①11111		
Set $Q_0 \leftarrow 0$	00011		
$A \leftarrow A+B$	00010		001

Shift Left(A,Q)	000101	0101	
$A+\bar{B}+1$	111101		
	⓪00010		000

Set $Q_0 \leftarrow 1$

17/3 = 000010 ⇒ Quotient

00101 ⇒ Remainder

(ii) **1101/101**

Q = 1101

B = 00101

$\bar{B}+1$ = 11011

Operation.	A	Q	SC
Initial	00000	1101	100
Shift left(A,Q)	00001	1010	
$A \leftarrow A+\bar{B}+1$	11011		
	①11100		
Set $Q_0 \leftarrow 0$			
$A \leftarrow A+B$	00101		011
	00001		
Shift Left(A,Q)	00011	0100	
$A \leftarrow A+\bar{B}+1$	11011		
	①11110		
Set $Q_0 \leftarrow 0$	00101		
$A \leftarrow A+B$	00011		010
Shift Left(A,Q)	00110	1001	
$A \leftarrow A+\bar{B}+1$	11011		
	⓪00001		
Set $Q_0 \leftarrow 1$			001

Shift Left (A,Q) 00011 0010

$A \leftarrow A + \bar{B} + 1$ 11011

Set $Q_0 \leftarrow 0$ 11110

$A \leftarrow A + B$ 00101

00011 000

Remainder ⇒ 00011

Quotient = 0010

(iv) 1000/111

Q = 1000

B = 00111

$\bar{B}+1$ = 11001

Operation	A	Q	SC
initial	00000	1000	100
Shift Left (A,Q)	00001	0000	
$A \leftarrow A + \bar{B} + 1$	11001		
	(1)10010		
Set $Q_0 \leftarrow 0$	00111		
$A \leftarrow A + B$	00001		011
Shift Left (A,Q)	00010	0000	
$A \leftarrow A + \bar{B} + 1$	11001		
	(1)10011		
Set $Q_0 \leftarrow 0$	00111		010
$A \leftarrow A + B$	00010		
Shift Left (A,Q)	00100	0000	
$A \leftarrow A + \bar{B} + 1$	11001		
	(1)11101		010
Set $Q_0 \leftarrow 0$			

$A \leftarrow A+B$	11101 00111 01100		0010
Shift Left (A,Q)	01000	0001	
$A \leftarrow A+\bar{B}+1$	11001 (1)00001		
Set $Q_0 \leftarrow$ 1			
~~$A \leftarrow A+B$~~			0001

Shift Left (A,Q)	00010	0010
$A \leftarrow A+\bar{B}+1$	11001 (1)11011	
Set $Q_0 \leftarrow 0$	00111	
$A \leftarrow A+B$	00010	0000
~~Set~~		

Quotient = 00010

Remainder = 00001

Non-Restoring Division

Consider the sequence of operations that takes place after the subtraction operation in the restoring algorithm.

If A is positive

Shift Left and subtract
divisor → 2A−B

If A is negative

Restore → A+B
shift left and ~~subject~~ subtract
divisor → 2(A+B) − B
= 2A+B

Looking at the above operations that takes place after the subtraction operation is

Looking at the above operations, we can write following steps for non-restoring algorithm

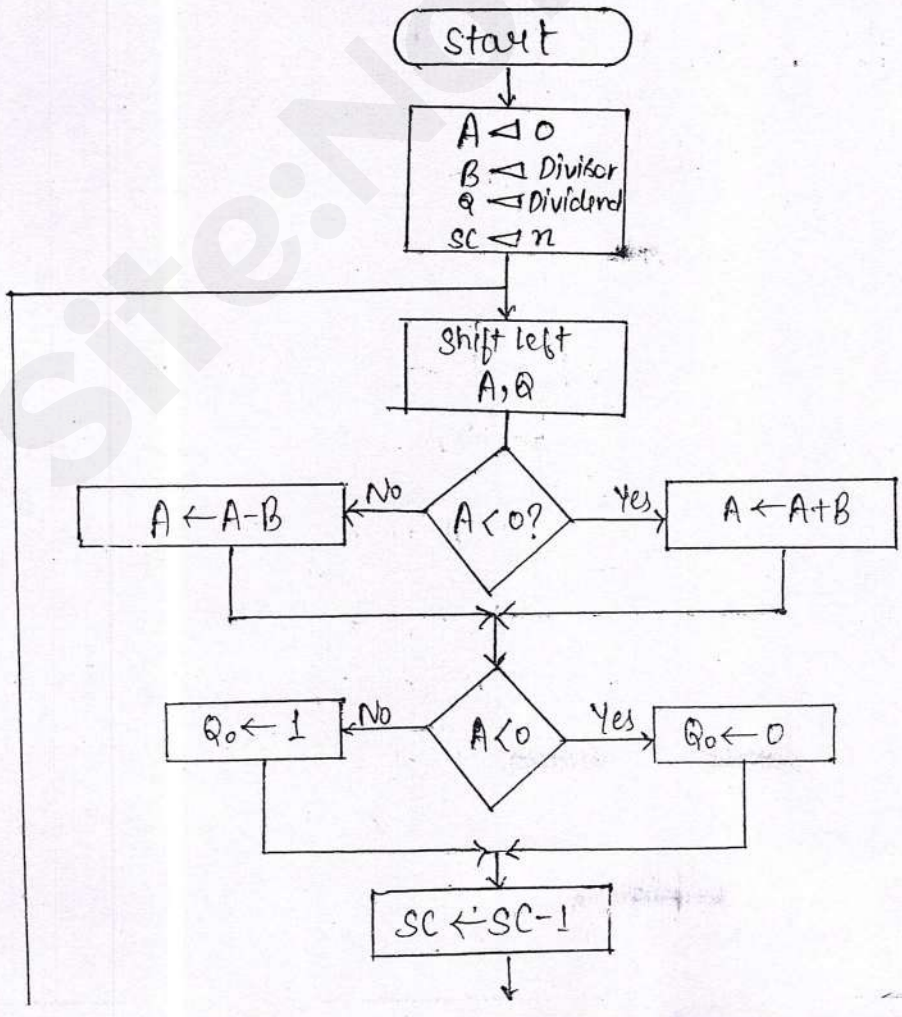
Step 1: If the sign of A is 0, shift A and Q left one bit position and subtract divisor from A, otherwise, shift A and Q left and add divisor to A. If the sign of A is 0, set Q_0 to 1, otherwise, set Q_0 to 0.

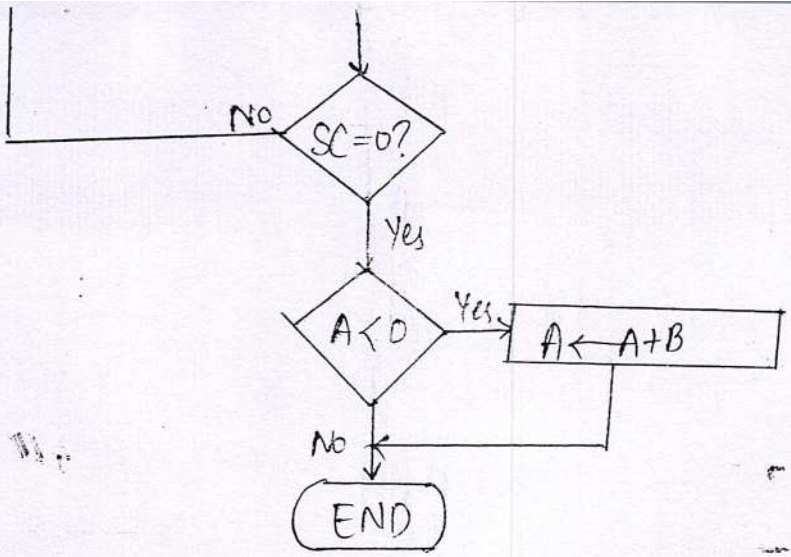
Step 2: Repeat steps 1 and 2 for n times.

Step 3: If the sign of A is 1, add divisor to A.

Note: Step 3 is required to leave the proper positive remainder in A at the end of n cycles

A flowchart for non-restoring division operation is as shown in Fig 2.11.10





Quotient in Q
Remainder in A.

(i) 1101 / 11

Q = 1101

B = 0011

B = 00011

$\bar{B}+1$ = 11101

Operation	A	Q	SC
initial	00000	1101	100
Shift left A, Q	00001	1010	
$A \leftarrow A+\bar{B}+1$	11101		011
Set $Q_0 \leftarrow 0$	11110		
Shift Left (A, Q)	11101	0101	
A + B	00011		
Set $Q_0 \leftarrow 1$	00000		
Shift left (A, Q)	00000	1010	
$A \leftarrow A+\bar{B}+1$	11101		001
Set $Q_0 \leftarrow 0$	11101		
Shift Left (A, Q)	11011	0100	
A ← A + B	00011		000
A+B	00011		Q = 0100
	00001		R = 00001

(ii) 17/3

$Q = 17 = 10001$

$B = 3 = 00011$

$B = 000011$

$\bar{B}+1 = 11101$

Operation	A	Q	SC
initial	000000	10001	101
Shift Left (A,Q)	000001	00010	
$A \leftarrow A+\bar{B}+1$	11101		
	111110		
Set $Q_0 \leftarrow 0$			100
Shift Left (A,Q)	111100	00100	
$A \leftarrow A+B$	000011		
	111111		011
Set $Q_0 \leftarrow 0$			
Shift Left (A,Q)	111110	01001	
$A \leftarrow A+B$	000011		
	000001		010
Set $Q_0 \leftarrow 1$			
Shift Left (A,Q)	000010	10010	
$A \leftarrow A+\bar{B}+1$	11101		
	111111		001
Set $Q_0 \leftarrow 0$			
Shift Left (A,Q)	111111	00101	
$A \leftarrow A+B$	000011		
	000010		000
Set $Q_0 \leftarrow 1$			

(iii) 19/4

$Q = 19 = 10011$

$B = 00100$

$B = 000100$

$\bar{B}+1 = 111011+1$

$\bar{B}+1 = 111100$

Operation	A	Q	SC
initial	000000	10011	101
Shift Left (A, Q)	000001	00110	
$A \leftarrow A+\bar{B}+1$	111100		
	111101		
Set $Q_0 \leftarrow 0$			100
Shift Left (A, Q)	111010	01100	
$A \leftarrow A+B$	000100		
	111110		
Set $Q_0 \leftarrow 0$			011
Shift Left (A, Q)	111100	11001	
$A \leftarrow A+B$	000100		
	000000		
set $Q_0 \leftarrow 1$			010
Shift Left (A, Q)	000001	10010	
$A \leftarrow A+\bar{B}+1$	111100		
	111101		
set $Q_0 \leftarrow 0$			001
Shift Left (A, Q)	111011	100100	
$A \leftarrow A+B$	000100		
Set $Q_0 \leftarrow 0$	111111		000

$A+B$

$$\begin{array}{r} 000100 \\ \hline 000011 \end{array}$$

Comparison b/w restoring and non-restoring algorithm.

Restoring	Non-restoring
Needs restoring of register A if the result of subtraction is negative	Does not need restoring
In each cycle content of register A is first shifted left and then divisor is subtracted from it.	In each cycle content of register A is first shifted left and then division is added or subtracted with the contents of register A depending on the sign of A.
Does not need restoring of remainder.	Needs restoring of remainder if remainder is negative
Slower algorithm	Faster algorithm