Data Representation [start from next page diagram] (13)

Data is either positive or negative. Here we study basically three types of data representation for negative numbers.

(i) Signed Magnitude
(ii) 1's complement
(iii) 2's complement.

(i) Signed Magnitude :-

MSB → ~~sign~~ signbit: 0 (+), 1 (-)

(It is already mentioned to reader that I am using signed magnitude representation)

⊛ MSB

$\boxed{1}001 = -1$ (001 = Magnitude)

MSB

$\boxed{0}001 = +1$

⊛ 1111 = -7

0111 = +7

(ii) 1's complement :-

⊛ Write -5 in 1's complement

→ First of all write binary form of 5, then take compliment of each and every bit.

5 = 0101

↓ complement of every bit

1010 = -5

Ex. $6 = 0110$ (+)
$\downarrow$ complement of every bit
Now $-6 = 1001$

(iii) **2's complement:-**
First take 1's complement of the number then add 1 in that then it will be the 2's complement of the no.

Ex. $5 \rightarrow 0101$
$\downarrow$ 1's compleemt
$-5 \rightarrow 1010$
$+1$
1011 (This is the 2's complement of the no. 5).

Range of Numbers:-

(i) Sign Mag:- $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$
(ii) 1's Comp: $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$
(iii) 2's Comp: -2^{n-1} to $+(2^{n-1}-1)$

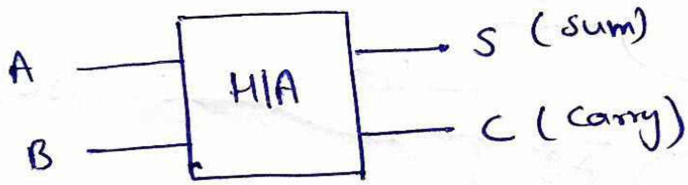
Data Representation
- Magnitude
  - Unsigned: (+) binary no.
  - Signed: both ⊕ & ⊖
- Complement
  - 1's complement: ⊕ & ⊖
  - 2's complement: ⊕ & ⊖

Computer Organization and Architecture Lab

Lab-1

Implementing HALF ADDER, FULL ADDER using basic logic gates.

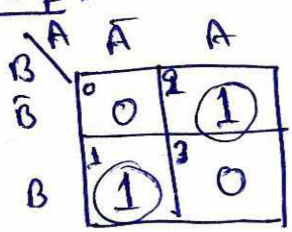
HALF ADDER:- The half adder adds two single binary digits A and B. It has two outputs sum (S) and carry (C).

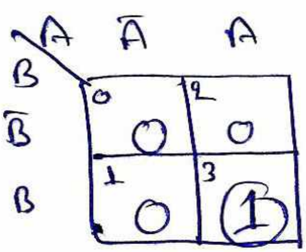


Truth Table:-

	Input		Output	
	A	B	S	C
0	0	0	0	0
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1

K-Map -



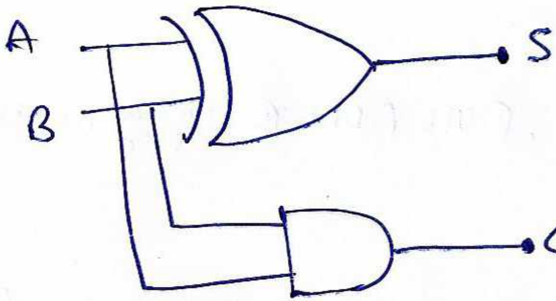


$S = A\bar{B} + \bar{A}B$

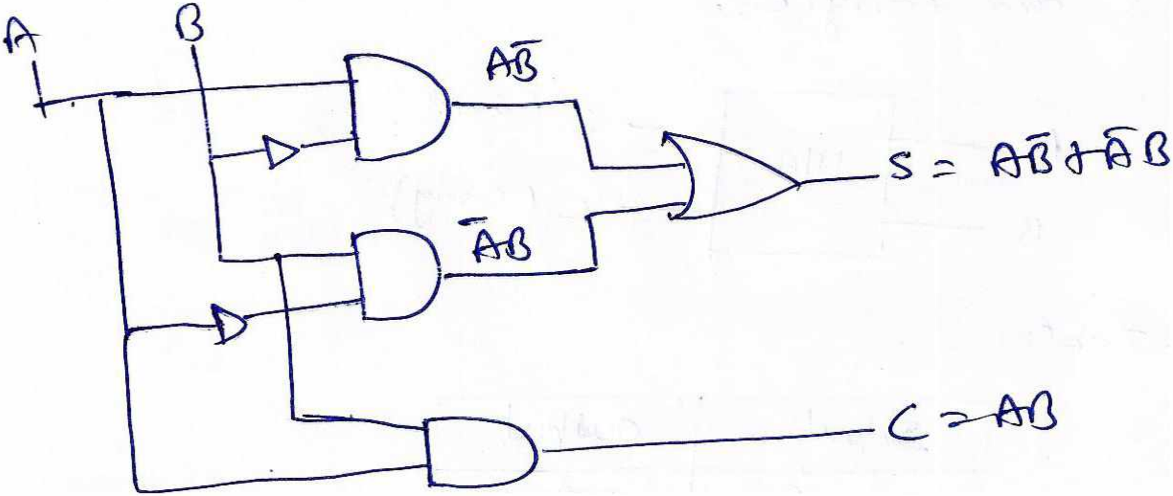
$= A \oplus B$

$C = AB$

Circuit:-

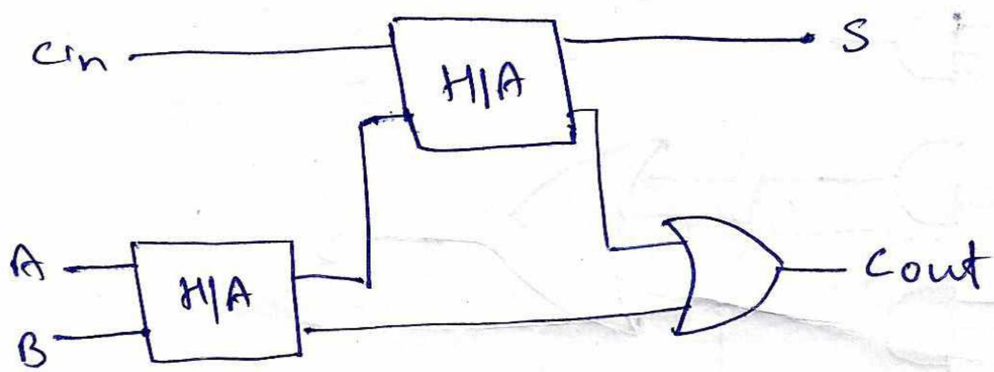


or



FULL ADDER :- (TWO HALF ADDER)

A full adder adds binary numbers and accouts for values carried in as well as out. A one bit full adder adds three one-bit numbers, often written as A, B and Cin. A and B are operand and Cin is a bit carried in from previous stage.

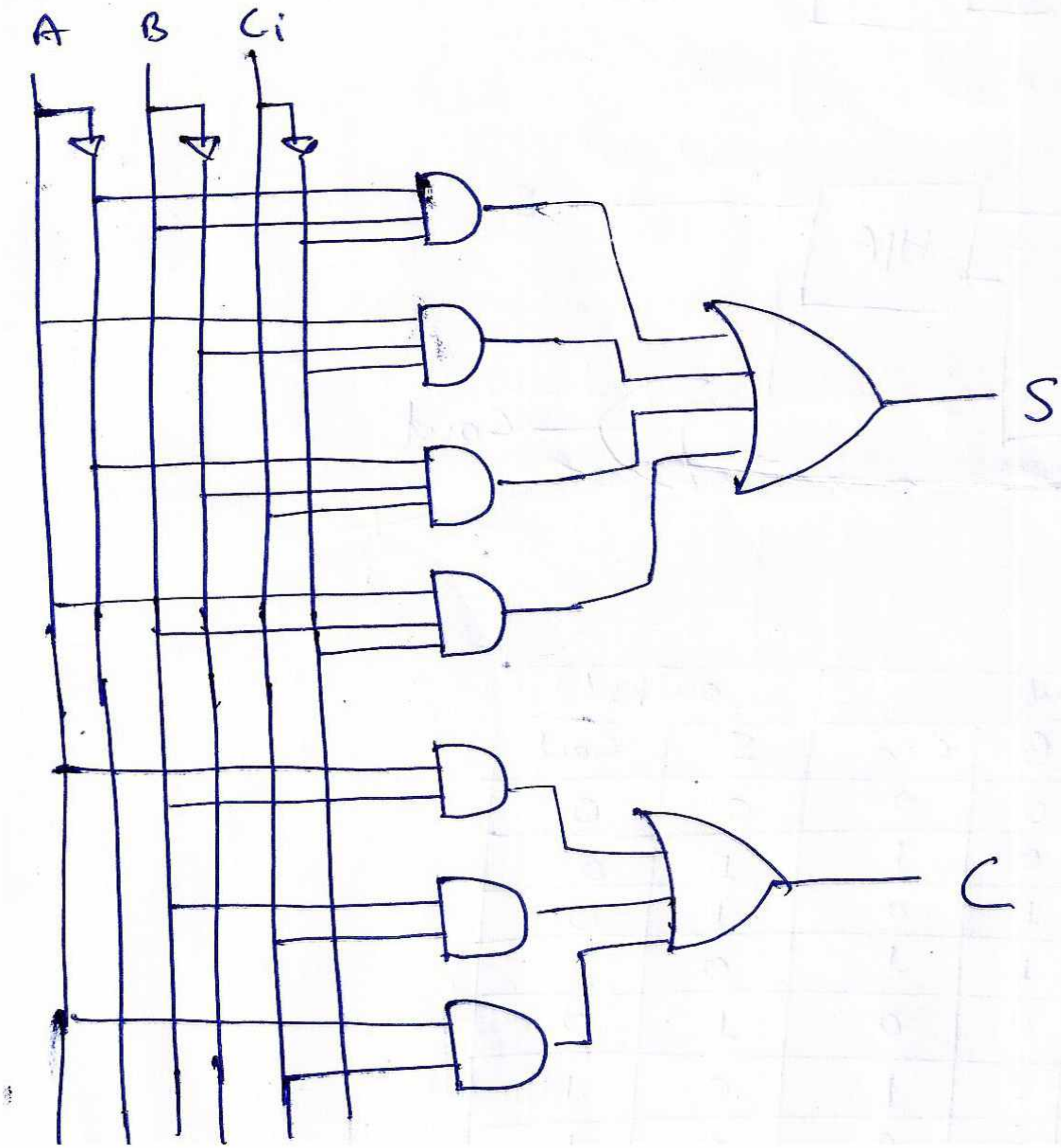


Truth Table :-

	Input			output	
	A	B	Cin	S	Cout
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

$S = \bar{A}B\bar{C_i} + A\bar{B}\bar{C_i} + \bar{A}\bar{B}C_i + ABC_i$ $\qquad C = AB + BC_i + AC_i$

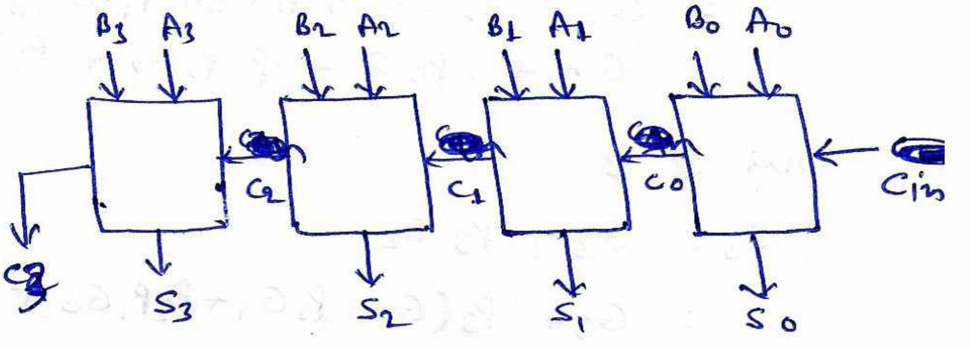
<u>Circuit:-</u>



Carry Look Ahead Adder (CLA)

CLA is used to reducing the carry propagation delay by calculating the carry signals in advance, based on the input signals.

A	B	Cin	Co
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



CLA adder circuit is based on the fact that a carry signal will be generated in two cases:

(i) when both bits A_i and B_i are 1, or

(ii) when one of the two bit is 1 and the carry-in (carry of the previous stage) is 1.

$$C_o = \underbrace{A \cdot B}_{\substack{\text{carry} \\ \text{generator} \\ (G)}} + \underbrace{(A \oplus B)}_{\substack{\text{carry} \\ \text{propagater} \\ (P)}} \cdot Cin$$

$$\boxed{C_o = G + P \cdot Cin}$$

Generalization of the above equation $(i-1)$

$$\boxed{C_i = G_i + P_i C_{i-1}}$$

Now put $i = 0$

$$C_0 = G_0 + P_0 C_{-1} \quad \text{— (I)}$$

put $i = 1$

$$C_1 = G_1 + P_1 C_0 \quad \text{— (II)}$$

Now put the value of C_0 in equ. (11)

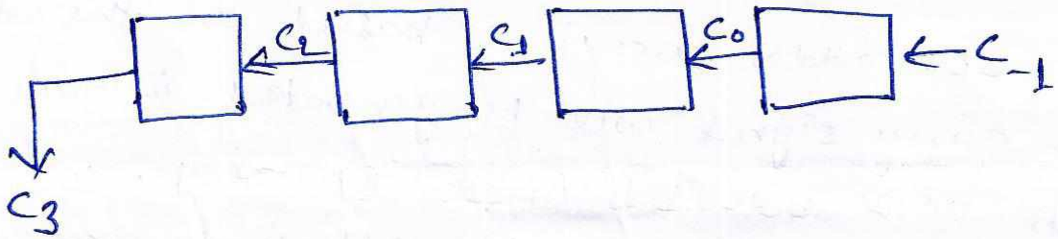
$$C_1 = G_1 + P_1(G_0 + P_0 C_{-1}) = G_1 + P_1 G_0 + P_1 P_0 C_{-1}$$

put $i = 2$

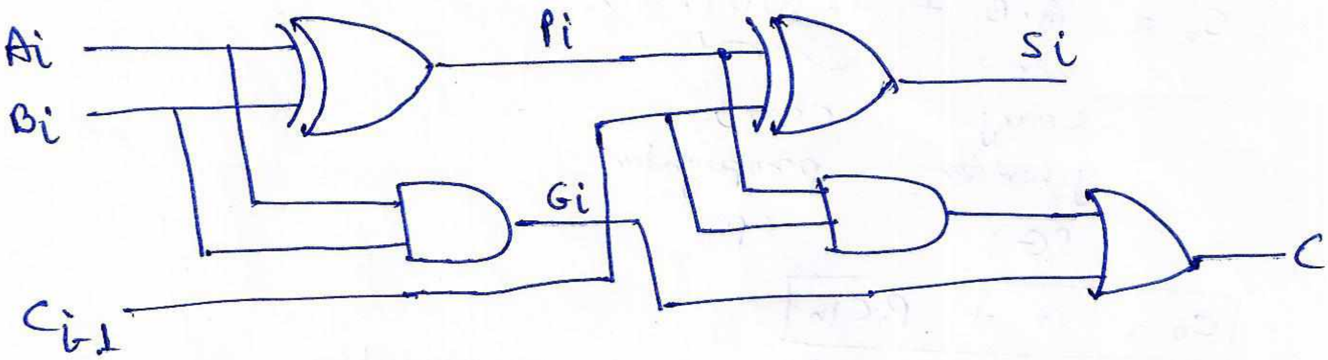
$$C_2 = G_2 + P_2 C_1$$
$$= G_2 + P_2(G_1 + P_1 G_0 + P_1 P_0 C_{-1})$$
$$= G_2 + P_2 G_1 + P_1 P_2 G_0 + P_2 P_1 P_0 C_{-1}$$

put $i = 3$

$$C_3 = G_3 + P_3 C_2$$
$$= G_3 + P_3(G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1})$$
$$\boxed{C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{-1}}$$



Full adder circuit used to add the operand bits in the i^{th} column, i.e. $A_i B_i$



$P_i = A_i \oplus B_i$

$G_i = A_i B_i$

$S_i = P_i \oplus C_{i-1}$

$C_i = G_i + P_i \cdot C_{i-1}$

(iii) 1011.001 – 110.10

Solution:

1's complement of 0110.100 is 1001.011 Hence

Minued -		1 0 1 1 . 0 0 1
1's complement of subtrahend -		1 0 0 1 . 0 1 1
Carry over -	1	0 1 0 0 . 1 0 0
		1
		0 1 0 0 . 1 0 1

Hence the required difference is 100.101

(iv) 10110.01 – 11010.10

Solution:

1's complement of 11010.10 is 00101.01

1 0 1 1 0 . 0 1

0 0 1 0 1 . 0 1

1 1 0 1 1 . 1 0

Hence the required difference is – 00100.01 i.e. – 100.01

Subtraction by 2's Complement

With the help of subtraction by 2's complement method we can easily subtract two binary numbers.

The operation is carried out by means of the following steps:

(i) At first, 2's complement of the subtrahend is found.

(ii) Then it is added to the minuend.

(iii) If the final carry over of the sum is 1, it is dropped and the result is positive.

(iv) If there is no carry over, the two's complement of the sum will be the result and it is negative.

```
                System.out.println("Sum ="+(a+b));
                break;
            case 2:
                System.out.println("Difference ="+(a-b));
                break;
            case 3:
                System.out.println("Product ="+(a*b));
                break;
            case 4:
                System.out.println("Quotient ="+(a/b));
                break;
        }

    }
}}
```

Output:-

Command Prompt

```
C:\Users\Prathmesh Singh\Desktop\Lab FIle>javac Calc.java

C:\Users\Prathmesh Singh\Desktop\Lab FIle>java Calculator
Enter First Number:5
Enter Second Number:6
Select Choice:
1)Addition
2)Substraction
3)Multiplication
4)Division
Enter Choice:4
Quotient =0.8333333333333334

C:\Users\Prathmesh Singh\Desktop\Lab FIle>
```

Subtraction by 1's Complement

In subtraction by 1's complement we subtract two binary numbers using carried by 1's complement.

The steps to be followed in subtraction by 1's complement are:

i) To write down 1's complement of the subtrahend.

ii) To add this with the minuend.

iii) If the result of addition has a carry over then it is dropped and an 1 is added in the last bit.

iv) If there is no carry over, then 1's complement of the result of addition is obtained to get the final result and it is negative.

(i) 110101 – 100101

Solution:

1's complement of 10011 is 011010. Hence

Minued -		1 1 0 1 0 1
1's complement of subtrahend -		0 1 1 0 1 0
Carry over -	1	0 0 1 1 1 1
		1
		0 1 0 0 0 0

The required difference is 10000

(ii) 101011 – 111001

Solution:

1's complement of 111001 is 000110. Hence

Minued -	1 0 1 0 1 1
1's complement -	0 0 0 1 1 0
	1 1 0 0 0 1

Hence the difference is – 1 1 1 0

Program No.:8

Objective:- Program to display a message on the screen.

Software Required:- TextEditor(Notepad++),Web Browser

Code:-

```
<html>
<head><title></title></head>
<body>
<h2>Program 8</h2>
<ul>
	<li>hello</li>
	<li>hello</li>
</ul><br>
<ol>
	<li>hello</li>
	<li>hello</li>
</ol><br>
<table border = "1">
	<tr>
		<td>hello</td>
		<td>hello</td>
	</tr>
```

(i) 110110 - 10110

Solution:

The numbers of bits in the subtrahend is 5 while that of minuend is 6. We make the number of bits in the subtrahend equal to that of minuend by taking a `0' in the sixth place of the subtrahend.

Now, 2's complement of 010110 is (101101 + 1) i.e.101010. Adding this with the minuend.

		1 1 0 1 1 0	Minuend
		1 0 1 0 1 0	2's complement of subtrahend
Carry over	1	1 0 0 0 0 0	Result of addition

After dropping the carry-over we get the result of subtraction to be 100000.

(ii) 10110 – 11010

Solution:

2's complement of 11010 is (00101 + 1) i.e. 00110. Hence

Minued -	1 0 1 1 0
2's complement of subtrahend -	0 0 1 1 0
Result of addition -	1 1 1 0 0

As there is no carry over, the result of subtraction is negative and is obtained by writing the 2's complement of 11100 i.e.(00011 + 1) or 00100.

Hence the difference is – 100.

(iii) 1010.11 – 1001.01

Solution:

2's complement of 1001.01 is 0110.11. Hence

Minued -		1 0 1 0 . 1 1
2's complement of subtrahend -		0 1 1 0 . 1 1
Carry over	1	0 0 0 1 . 1 0

After dropping the carry over we get the result of subtraction as 1.10.

Program No.:7

Objective:- To write a program of calculator .

Software Required:- TextEditor(Notepad++),JDK

Code:-

```
import java.util.*;
class Calculator
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.print("Enter First Number:");
		double a=sc.nextDouble();
		System.out.print("Enter Second Number:");
		double b=sc.nextDouble();
		System.out.print("Select
Choice:\n1)Addition\n2)Substraction\n3)Multiplication\n4)Division\n
Enter Choice:");
		int choice=sc.nextInt();
		switch (choice)
		{
			case 1:
```

(iv) 10100.01 – 11011.10

Solution:

2's complement of 11011.10 is 00100.10. Hence

Minued -	1 0 1 0 0 . 0 1
2's complement of subtrahend -	0 0 1 0 0 . 1 0
Result of addition -	1 1 0 0 0 . 1 1

As there is no carry over the result of subtraction is negative and is obtained by writing the 2's complement of 11000.11.

Hence the required result is – 00111.01.

Output:-

Command Prompt

```
C:\Users\Prathmesh Singh\Desktop\Lab FIle>javac Prog6.java

C:\Users\Prathmesh Singh\Desktop\Lab FIle>java Eod
Enter number:3
Number is odd

C:\Users\Prathmesh Singh\Desktop\Lab FIle>java Eod
Enter number:4
Number is even

C:\Users\Prathmesh Singh\Desktop\Lab FIle>_
```

Multiplication Algorithms :- (Signed Operand Multiplication)

Multiplication of two fixed point binary number in signed magnitude representation is done by with paper and pen by a process of successive shift and add operation.

```
   101        5   Multiplicant
 × 011        3   Multiplier
 -----
   101
  101  +
 000
 -----       --
 01111       15   Product.
```

Procedure :-

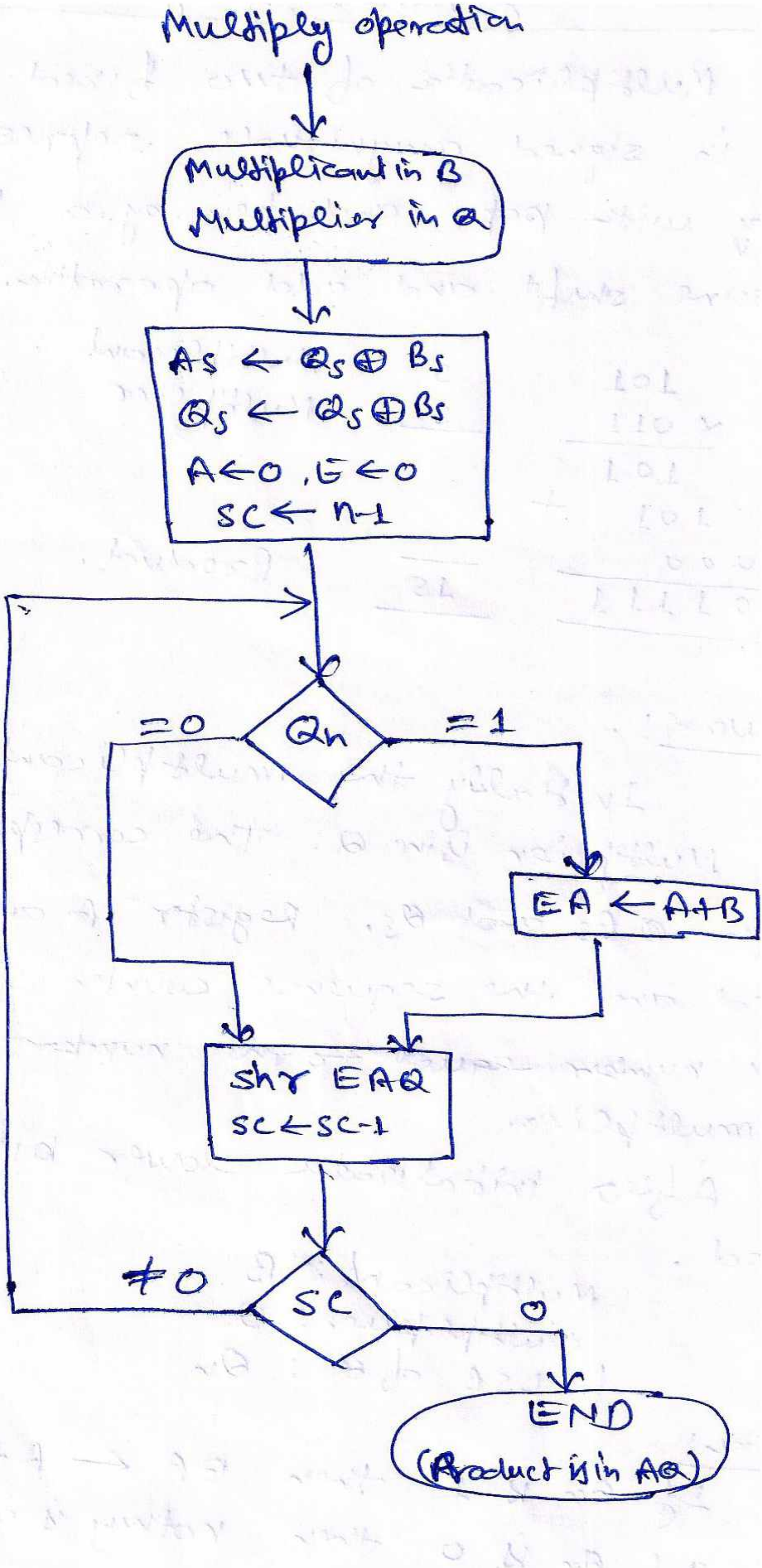
Initially the multiplicant is in B and Multiplier is in Q. The corresponding signs are in Bs and Qs. Register A and E are cleared and the sequence counter SC is set to a number equal to the number of bits of the multiplier.

After initialization lower bit of Qn is tested.

Multiplicant : B
Multiplier : Q
LSB of Q : Qn

Condition!
(i) If Qn is 1 then EA ← A+B
(ii) If Qn is 0 then nothing is done.

Register EAQ is then shifted right.



Flow chart for multiply operation

Booth's Multiplication Algorithm :-

Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation. It operats on the fact that strings of 0's in multiplier require no addition but just shifting.

consider a four bit number ie

~~Q : Q_1 Q_2 Q_3 Q_4 | Q_5~~

Q : Q_1 Q_2 -- Q_n | Q_{n+1}

Conditions:-

1. Q_n Q_{n+1}
   0 0
   1 1 } when both Q_n and Q_{n+1} bit are same perform only arithmetic right shift (ashr)

2. 0 1 } Then perform $A \leftarrow A+M$ then ashr

3. 1 0 } Then perform $A \leftarrow A-M$ then ashr.

A is Accumular initialize with 0.

ashr is Arithmetic shift right.

AQ is the final answer

(If any one multiplicant or multiplier is negative then final answer is 2's complement of AQ).

Exp:- $7 \times 3 = 21$

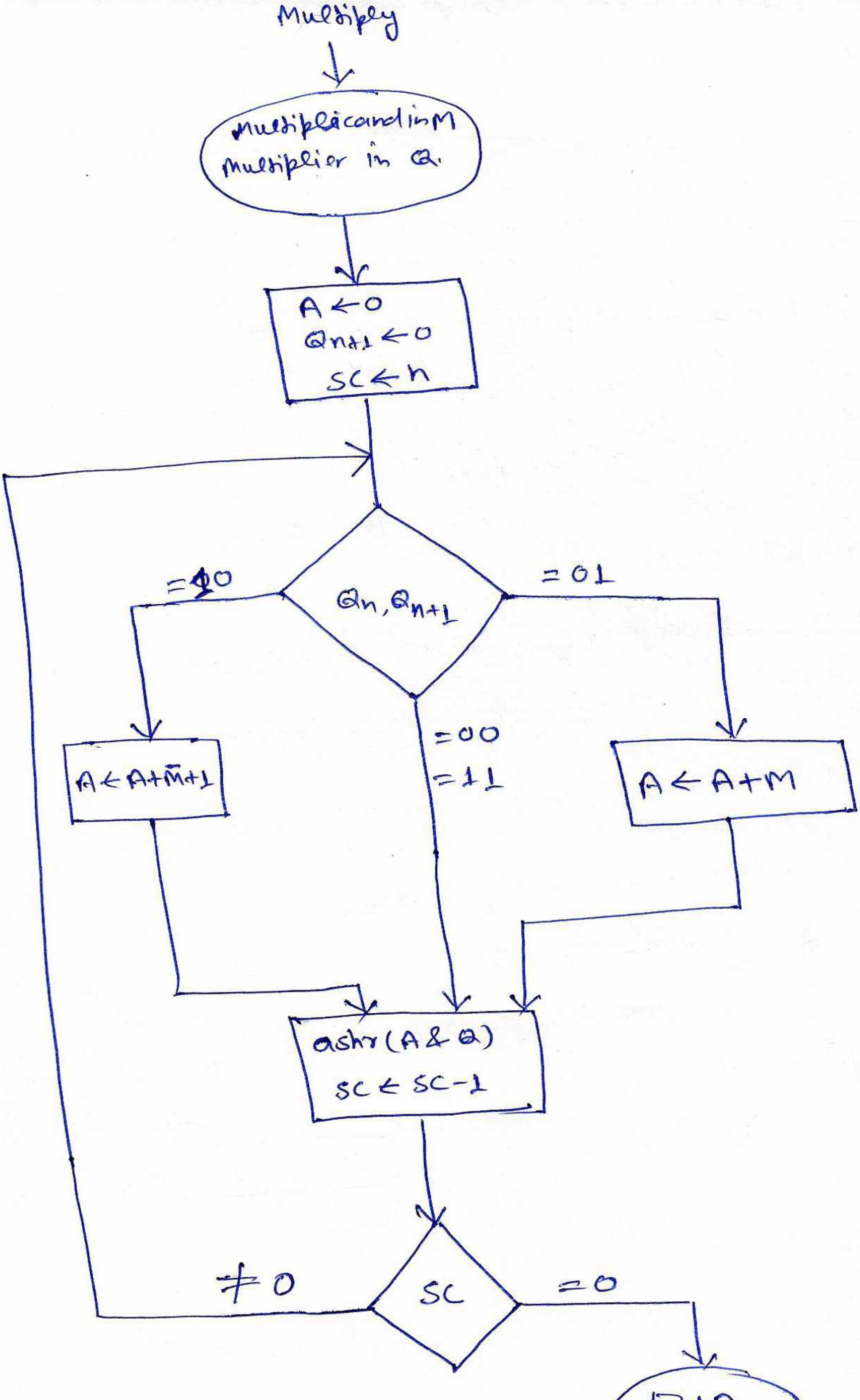
$M = 7 = 0111$ Multiplicant

$Q = 3 = 0011$ Multiplier

$A = 0000$, $Q_{n+1} = 0$ $SC = 4$

A	Q	Q_{n+1}	SC	Action
0000	0011	0	4	$A \leftarrow A - M \equiv A +$ 2's complement of M 0000 1001 1001
1001	0011	0		ashr
1100	1001	1	3	
1110	0100	1	2	ashr
				$A \leftarrow A + M$ 1110 0111 ~~1~~0101
0101	0100	1	1	ashr
0010	1010	0		
0001	0101	0	0	END

$10101 = 21$ Ans



Booth's Algorithm for multiplication of signed-2's complement numbers

Checking the bits of the multiplier one at a time and forming partial products is a sequential operation that requires a sequence of add and shift microoperations.

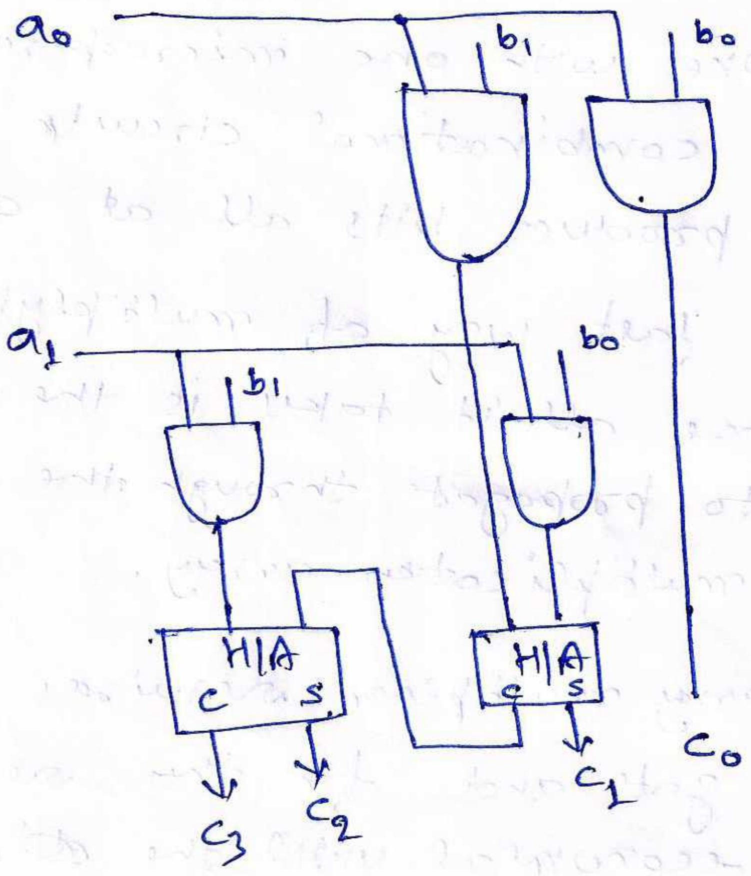
The multiplication of two binary numbers can be done with one microoperation by means of a combinational circuit that forms the product bits all at once. This is a fast way of multiplying two numbers since all it takes is the time for the signal to propogate through the gates that form the multiplication array.

An array multiplier requires a large number of gate and for this reason it was not economical until the ~~de~~ development of integrated circuits.

P.T.O.

eg.

		b_1	b_0
		a_1	a_0
		$a_0 b_1$	$a_0 b_0$
	$a_1 b_1$	$a_1 b_0$	
c_3	c_2	c_1	c_0



2-bit by 2-bit Array Multiplier

Division Algorithms :-

Division of two fixed point binary number in signed-magnitude representation is done with paper and pencil by a process of successive compare, shift and subtract operation.

Exp. Divisor

```
              11010
B = 10001 ) 0111000000
            01110
            011100
             10001
             010110
               10001
               0010100
                 10001
                  0110
```

Quotient = Q

Dividend = A

5 bit of $A < B$

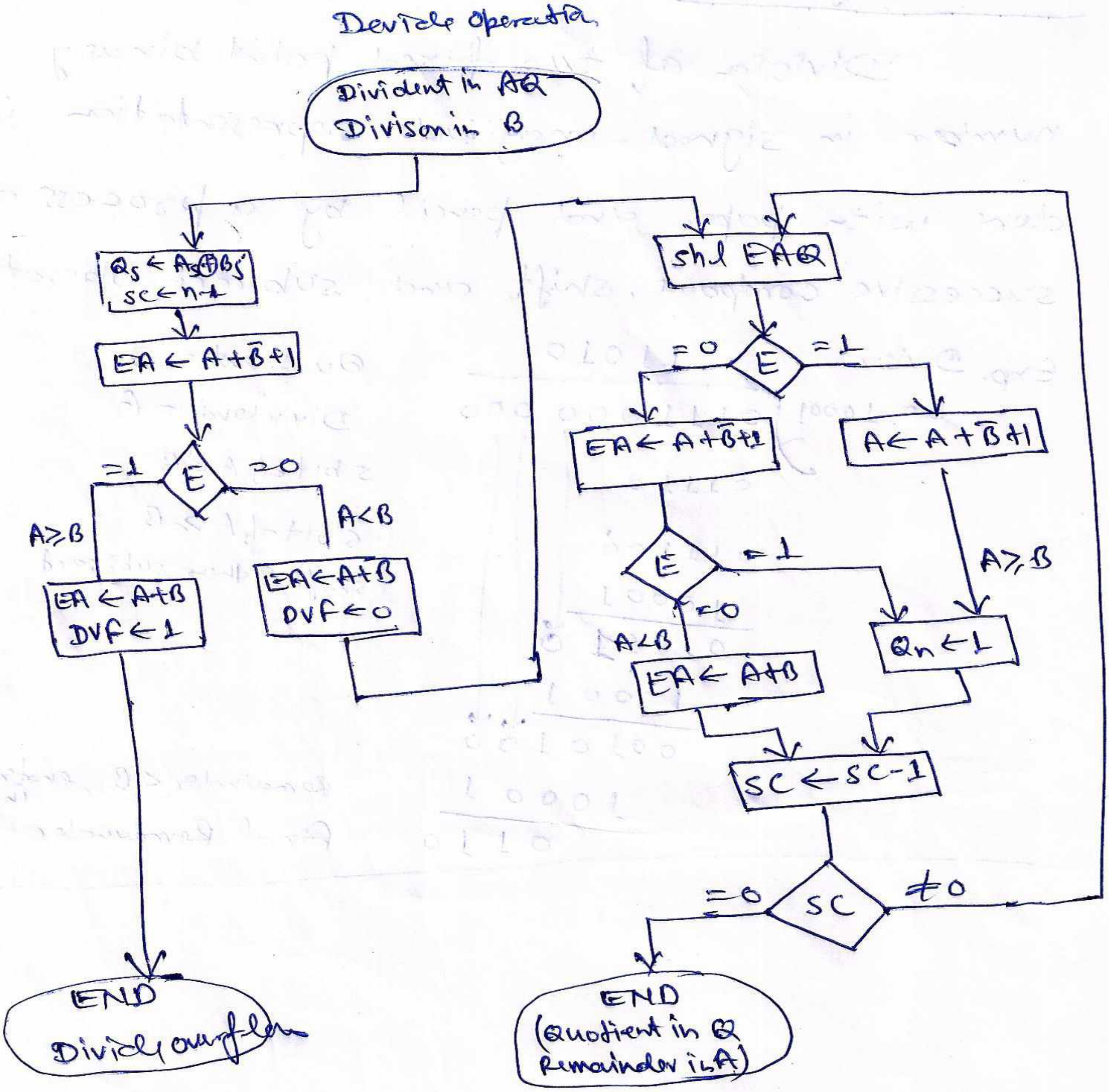
6 bit of $A \geq B$

shift B then subtract

Remainder $< B$, enter 0 in Q.

Final Remainder

Hardware Implementation :-



Flow chart for Divide Operation

Divide overflow (DVF) :- Arises because the length of register is finite and will not hold a number that exceeds the standard length. Another problem associated with division is that division by zero is also avoided. It is also handle by DVF.

Procedure :- Divident A

Divisor B

Double length dividend stored in AQ

E initial empty

Step 1- First off all perform shl EAQ

Step 2- (i) If $E=1$ then $A \leftarrow A-B$

and $Q_n \leftarrow 1$

(ii) If $E=0$ then $EA \leftarrow A-B$

(a) If $E=1$ then $Q_n \leftarrow 1$

(b) If $E=0$ then $EA \leftarrow A+B$

Step3. $SC \leftarrow SC-1$ (Repeat until $SC=0$)

Note (i) subtraction perform by 2's complement.

(ii) $\boxed{A-B = A+\bar{B}+1}$

(iii) SC = no-of bit in ~~Dividend~~ Q. (Quotients).

Exp.

$B = 10001 \quad \overline{B}+1 = 01111$

$A = 01110$

	E	A	Q	SC
		01110	00000	5
shl	0	11100	00000	
		01111		
	1	01011	00001	4
	0	10110	00010	
		01111		
	1	00101	00011	3
	0	01010	00110	
		01111		
	0	11001		
		10001		
	1	01010	00110	2
	0	10100	01100	
		01111		
	1	00011	01101	1
	0	00110	11010	
		01111		
	0	10101		
		10001		
	1	00110	11010	0

Ans. Remainder in A

Quotient in Q

Neglect E.

Floating Point Arithmatic Operation

Single precision (32 bit) | Double precision (64 bit)

IEEE 32-bit Floating Point Number :-

31	30 – 23	22 – 0
S	E'	M
1 bit	8 bit	23 bit

$E' = E + bias \Rightarrow (E + 127)$

$S \rightarrow$ sign ($0 \rightarrow +ve$, $1 \rightarrow -ve$)

$E' \rightarrow$ Exponent 8 bit

$M \rightarrow$ Mantissa 23 bit

Normalized Form:-

113.927×10^{-6}

$\Rightarrow 1.13927 \times 10^{-4}$

$E = -4$

$M = 13927$

Exp.

Exp 1. Write $(1460 \cdot 125)_{10}$ into IEEE 32 bit floating point representation.

Sol: $(1462 \cdot 125)_{10} = (10110110100 \cdot 001)_2$

Normalized form

$1 \cdot 01101101000001 \times 2^{10}$

$S = 0$

$M = 01101101000001$

$E = 10 \Rightarrow E' = E + 127$

$= 10 + 127 = 137$

$E' = 10001001$

0	10001001	01101101000001 0

Exp 2. Write $-(0.75)_{10}$ into IEEE 32 bit floating point number.

Sol: $-(0.75)_{10} = -(0.11)_2$

$.75 \times 2 = 1.50 \quad 1$

$.50 \times 2 = 1.0 \quad 1$

Normalized form

1.1×2^{-1}

$S = 1$

$M = 1$

$E = -1 \Rightarrow E' = E + 127 = 126 =$

128	64	32	16	8	4	2	1
0	1	1	1	1	1	1	0

1	01111110	1000 ......