UNIT - II

(1)

Adder :-

- Half adder (2 bits)
- full Adder (3 bits)

Half adder

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

0+0 = 0

0+1 = 0

1+0 = 1

1+1 = 0 with carry 1

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

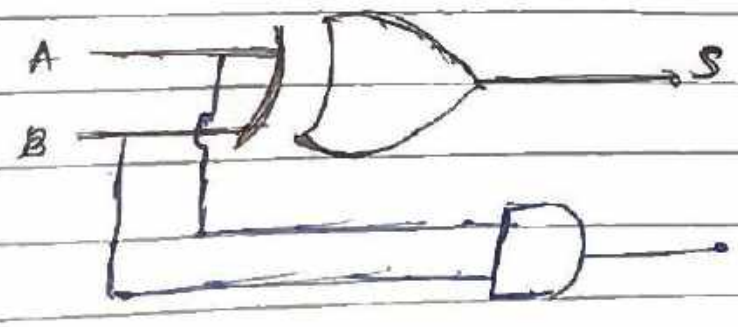
$S = A \oplus B$

$C = AB$.

Expression

- SOP — Sum of Product
  - 1 → real
  - 0 → comp
- POS — Product of Sum
  - 1 - compon
  - 0 - real.

comp.



Full adder

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

$$S = C_{in}(\bar{A}B + A\bar{B}) + \bar{C}_{in}(AB + A\bar{B})$$

$$S = C_{in}(A \odot B) + \bar{C}_{in}(A \oplus B)$$

$$S = C_{in}(\overline{A \oplus B}) + \bar{C}_{in}(A \oplus B)$$

let $A \oplus B = X$

$$S = C_{in}\bar{X} + \bar{C}_{in}X$$

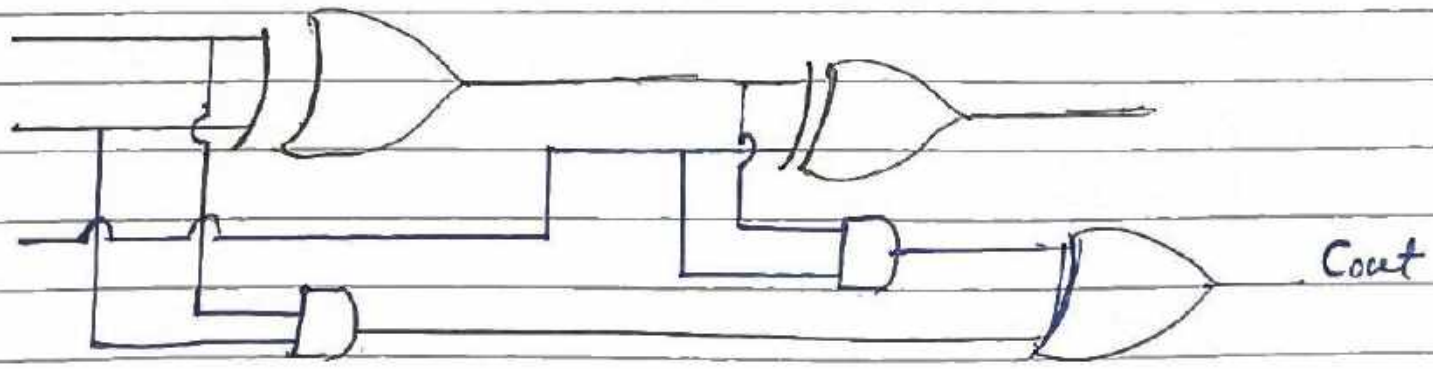
$$S = X \oplus C_{in}$$

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$$

$$C_{out} = C_{in}(\bar{A}B + A\bar{B}) + AB(\bar{C}_{in} + C_{in})$$

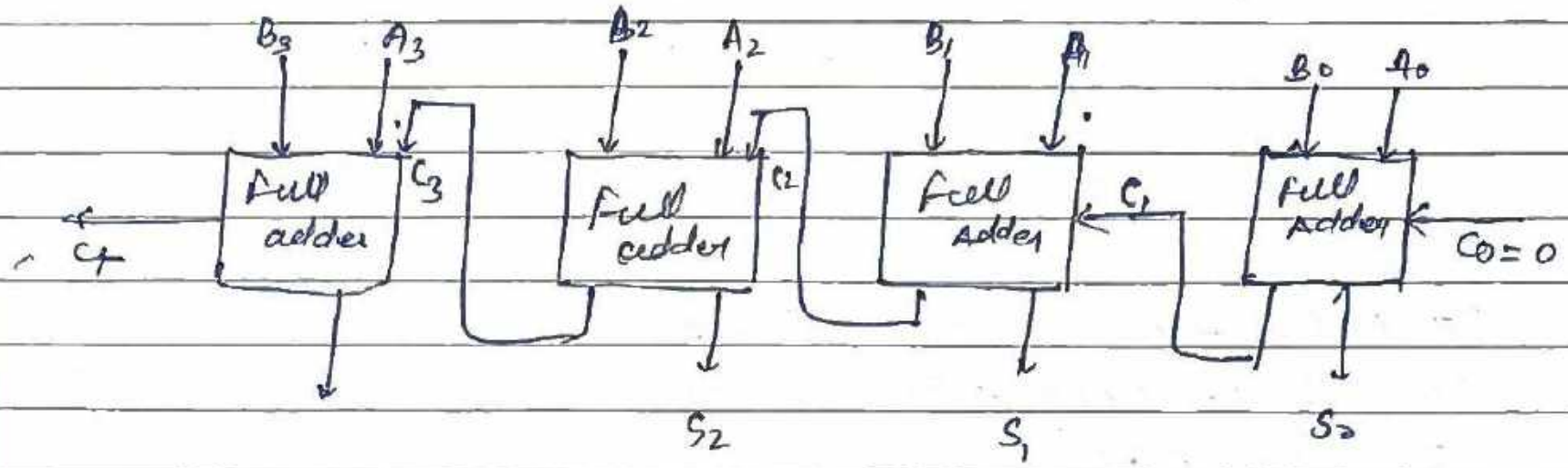
$$C_{out} = C_{in}(A \oplus B) + AB$$



Parellel Adder/ Ripple Adder/ Binary Adder.

C_3 C_2 C_1 C_0
A_3 A_2 A_1 A_0
\+ B_3 B_2 B_1 B_0

C_4 S_3 S_2 S_1 S_0



Look Ahead Carry Adder. :-

① To over come the problem of execessive delay in ripple carry ~~order~~ adder a fast adder known as look a-head carry adder can be design.

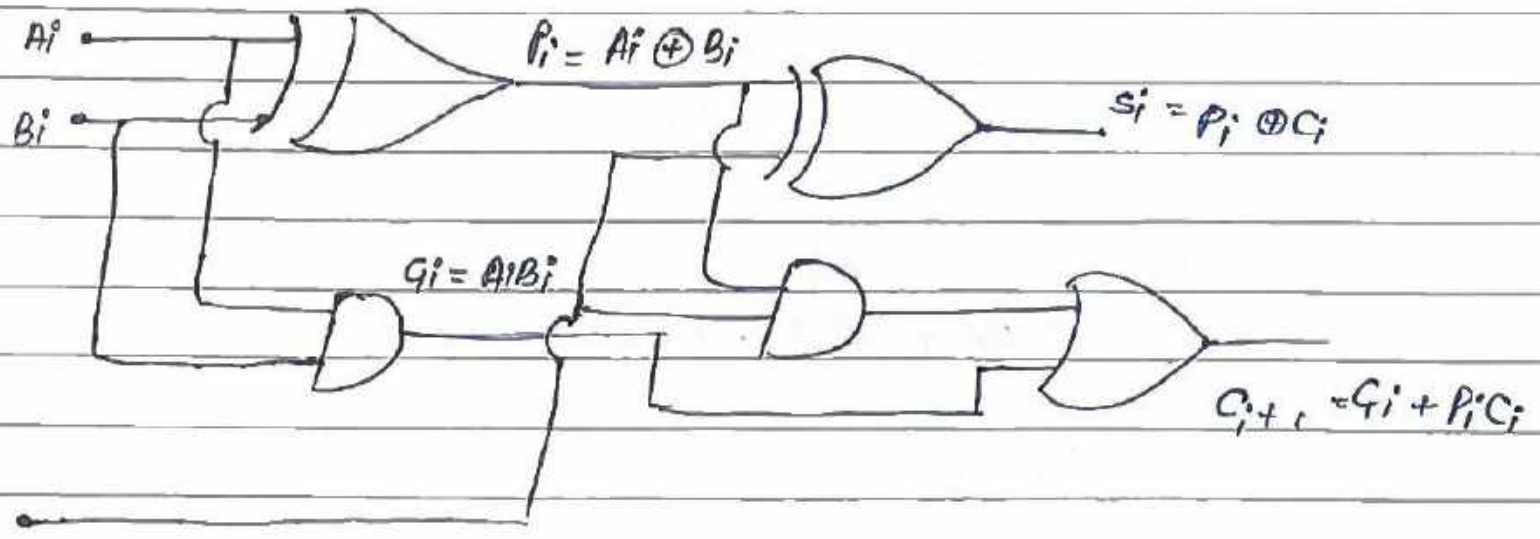
② In this adder the carry in for various steges can be generated directly by the logic expression.

- The generated method for designing ~~for~~ the fast adder is to reduce the time require to form carry signal
- It is uses the logic gate to look at lower order bits of the data to see if a higher order carry is to be generated. It uses ~~two~~ two functions

① Carry Generate

② Carry Propogate.

- Following figure shows the full adder circuit used to add the operand bits in the ~~highest~~ i^{th} column.



- form the given circuit Pi and Gi are given by $A_i \oplus B_i$ and A_iB_i respectively. The output of full adder that is sum and carray can be defined as

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_iC_i$$

- Gi is known as carry generate signal. A carry C_{i+1} is generated whenever $G_i = 1$, Regardless of input carry

- Pi is known as carry propogate signal, whenever $P_i = 1$ the input carry is propagated to the output carry.

- The Boolean expression for the carry output of various stages can be written as follow.

$$C_{i+1} = G_i + P_i C_i \quad \text{——①}$$

$i = 0 \quad C_1 = G_0 + P_0 C_0 \quad \text{——①}$

$i = 1 \quad C_2 = G_1 + P_1 C_1$

$C_2 = G_1 + P_1 (G_0 + P_0 C_0)$

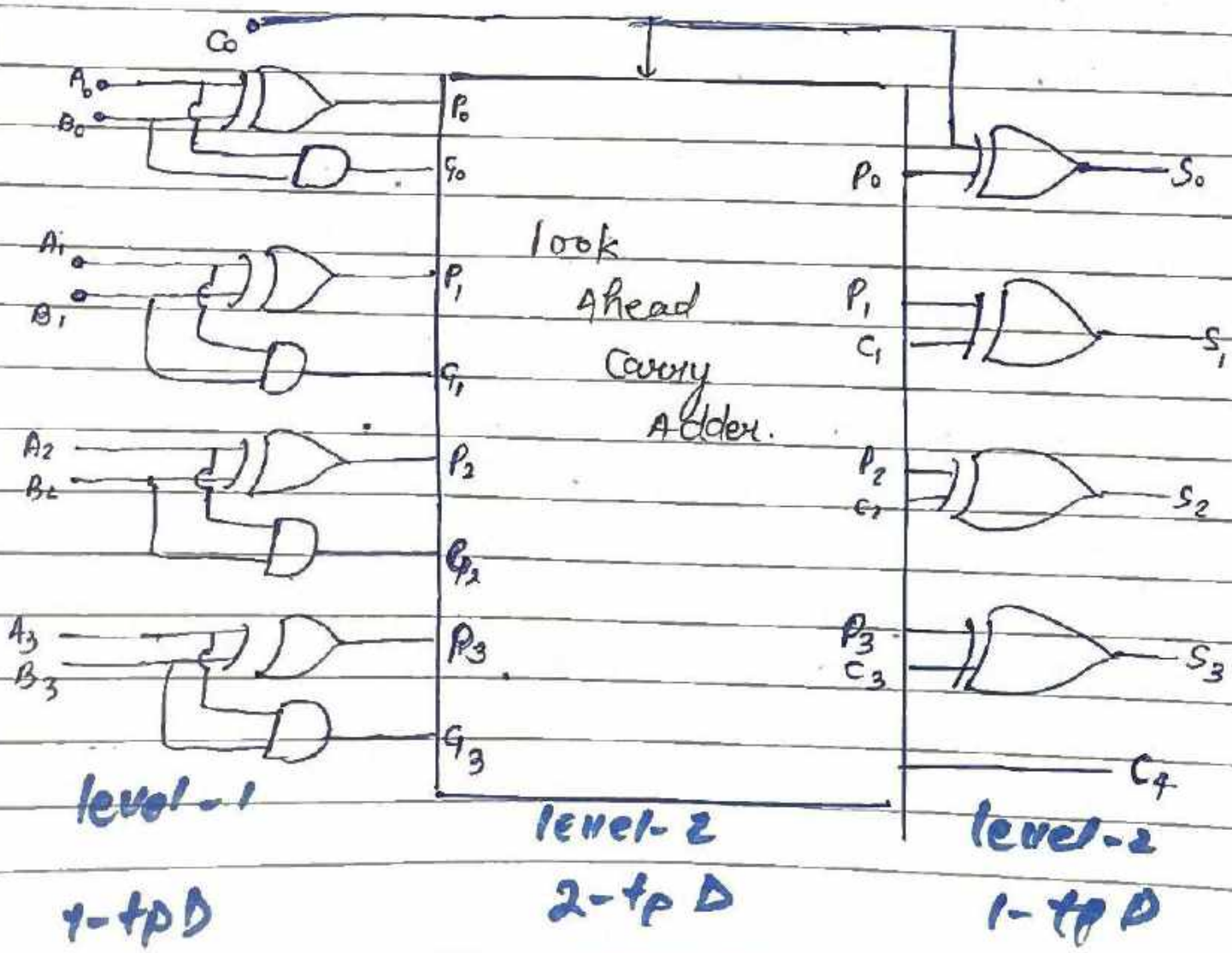
$C_2 = G_1 + G_0 P_1 + P_0 P_1 C_0 \quad \text{——②}$

$i = 2 \quad C_3 = G_2 + P_2 C_2$

$C_3 = G_2 + P_2 (G_1 + G_0 P_1 + P_0 P_1 C_0)$

$C_3 = G_2 + G_1 P_2 + G_0 P_1 P_2 + P_0 P_1 P_2 C_0. \quad \text{——③}$

- From the ~~over~~ above expression each carry signal is expressed as a direct SoP function rather than its preceding carry signal



- The four bit carry look ahead adder consist of three level of logic.
- In the first level it generates all the P & G signal and it provides 1-tpD.
- In the second level it carry look ahead ~~Boolt~~ adder consist of ~~four~~ (four) 2 level implementation circuit logic. It generates carry signal C_1, C_2, C_3 C_4 as defined by the upper expression and it provides 2-tpD time delay.
- In the third level four X-OR gate generates the sum ($S_i = P_i \oplus G_i$) and it provides 1-tpD time delay
- Therefore the total time delay of look ahead carry adder is 4-tpD.

Sign Addition or Subtraction :- 2's complement representation is the best choice for addition of 2 sign number. if x and y are the two positive numbers then

$$x - y = x + (2\text{'s complement of } y)$$

$$-x + y = (2\text{'s complement of } x) + y$$

$$-x - y = (2\text{'s comp of } x) + (2\text{'s comp of } y)$$

Que Find the value of following expression using 2's comp.

① +55+27

+55 = 00110111
+27 = 00011011
82 01010010

② −55+27

+55 ⇒ 00110111
2's comp ⇒ 1's + 1
11001000
+1
11001001

11001001
+ 00011011
11100100

③ +55−27

+55 −

00110111
+ 11100101
00011100

+27 ⇒ 00011011
11100100
+1
11100101

Binary Multiplication (Unsigned number):-

As compare to addition and subtraction multiplication is the complex operation whether perform in hardware or software. For multiplication of unsigned binary several important observation can be make.

They are. ① Multiplication involves the generation of partial product one for each digit in the multiplier. These partial product are then some to produce the final product.

② The partial product are easly defined either the multiplier width is zero the partial product is zero.

③ The total product is produced by summing the partial product. For this operation each succesive partial product is shifted one position to the left related to the preciding partial product.

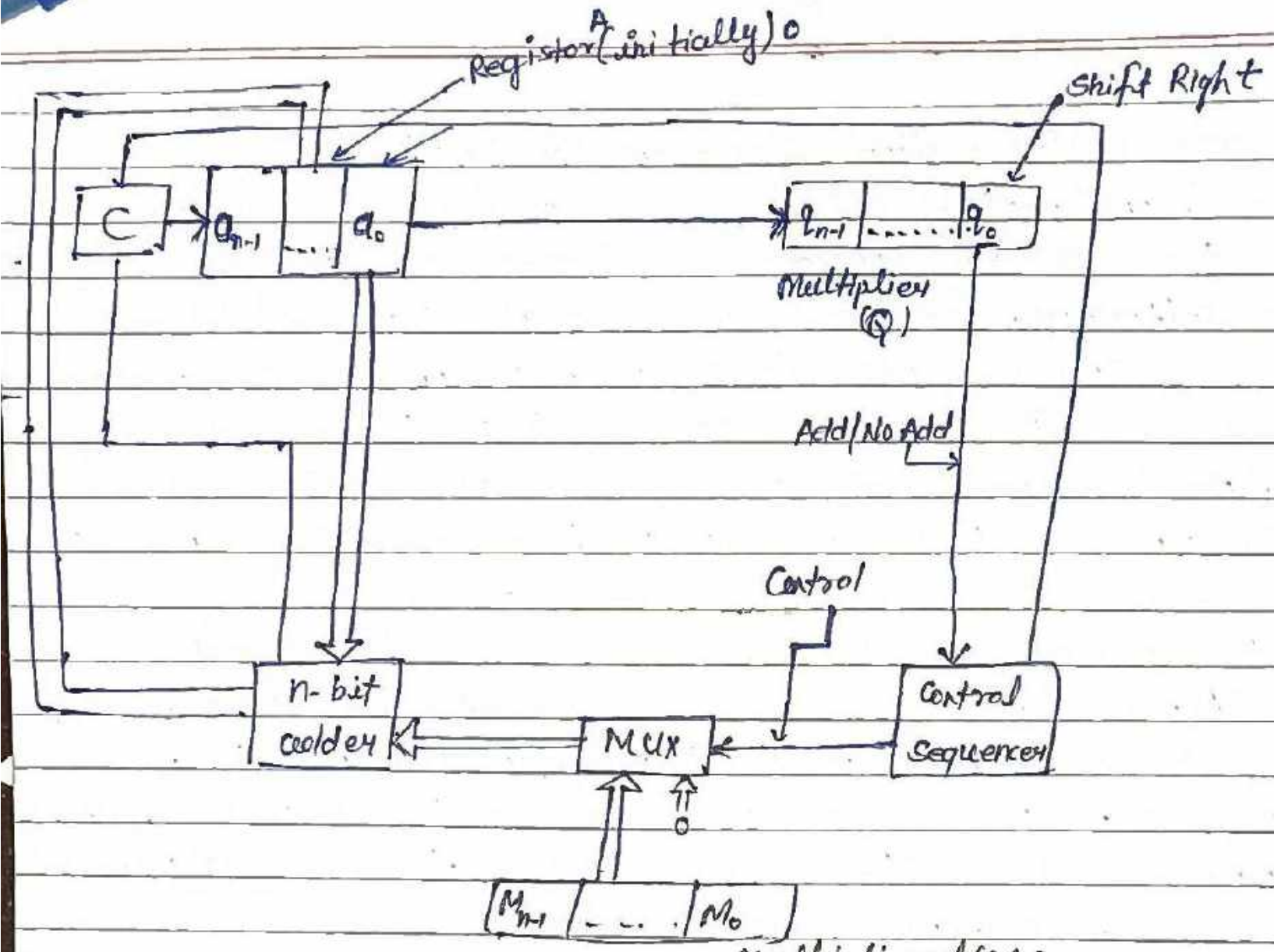
④ The multiplication 2 n bit. width binary result in a product of upto 2^n bits in length.
So the product of two (four bit) number fits into 8 bits

5) In the binary system of multiplication of the multiplicant by one bit of the multiplier is easy. If the multiplier bit is one, the multiplicant is enter in the appropriate position added to the partial product

Ex

1101 → Multiplicand (M)
1011 → Multiplier (Q)

1101
1101x
xx



The operation of multiplier is as follow.

(i) Control sequencer reads the bit of the multiplier one at a time. If Q_0 is one the the multiplicand is added to A register and the result is store in A register with the ~~cap~~ C bit use for overflow.

(ii) Then all of the bits of the C, A, Q register are shifted to the write one bit ~~shift~~ so that C bit goes in a_{n-1}, a_0 goes q_{n-1}, and q_0 is lost

(iii) If q_0 is zero than no addition is perform, just right shifting is done. This process is repeated for each ~~bit~~ bit of the original multiplier.

C	A	Q	M	
	0000	1011	1101	Initial value.
0	1101	1011	1101	Add } First
0	0110	1101	1101	Shift } cycle
0	0011	1101	1101	Add } Second
1	1001	1110	1101	Shift } Cycle
0	0100	1111	1101	Add & Third cycle
0	0001	1111	1101	Add } Fourth
1	1000	1111	1101	Shift } cycle.
0				

Que =

C	A	Q	M	
	0000	1100	1001	Initial Value
0	1001	1100	1001	Add } First cycle
0	0100	1110	1001	Shift }
0	0010	0111	1001	Add } Second cycle
0	1011	0111	1001	
0	0101	1011	1001	Add } Third cycle
0	0010	0111	1001	
0	1011	0111	1001	
0	0101	1011	1000	

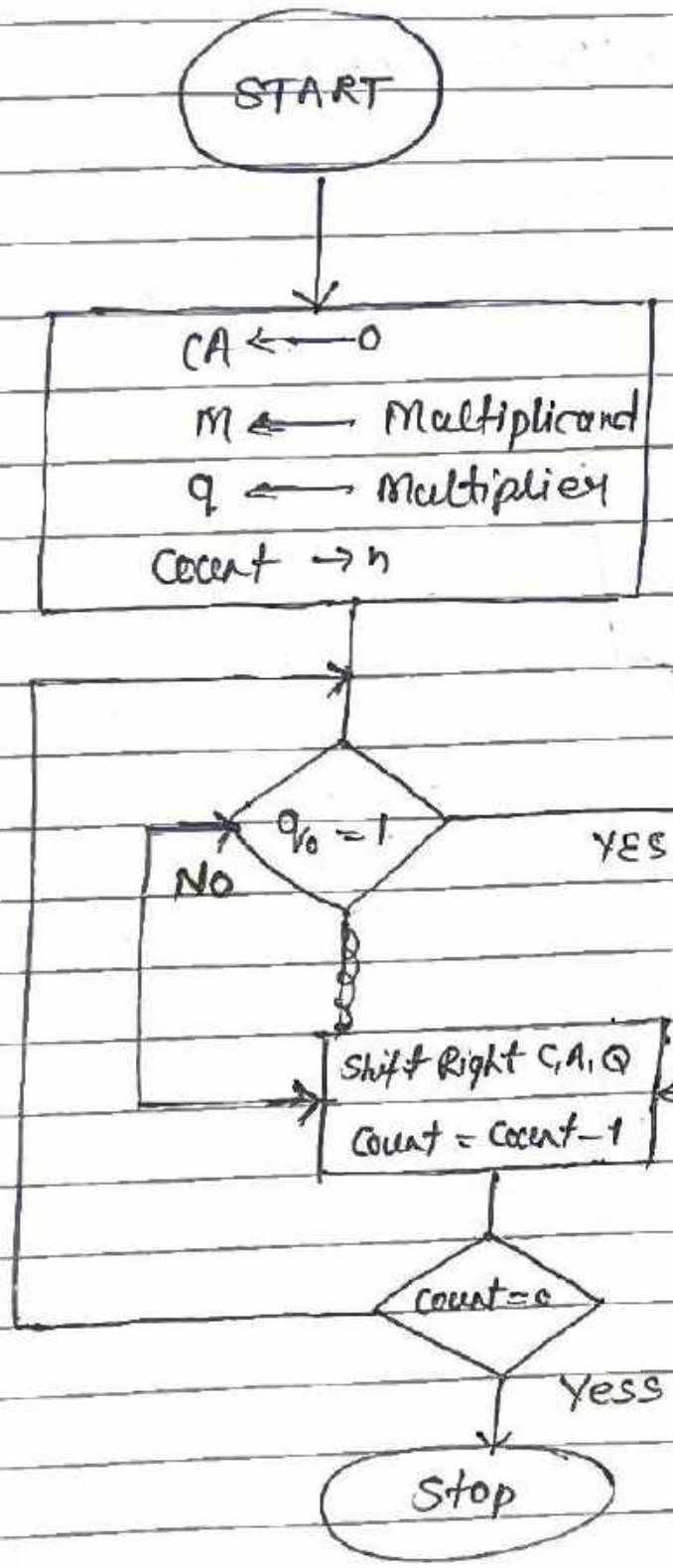
1101
1011
1101
1101
0000
1101
10001111

11101 → M
10011 → Q

C	A	Q	M	
0	0000	1001	1100	Initial value
0	1100	1001	1100	Add } First cycle
0	0110	0100	1100	Shift }
0	0011	0010	1100	Second - cycle
0	0001	1001	1100	Third - cycle
0	1101	1001	1100	Add } fourth
0	0110	1100	1100	Shift } cycle.

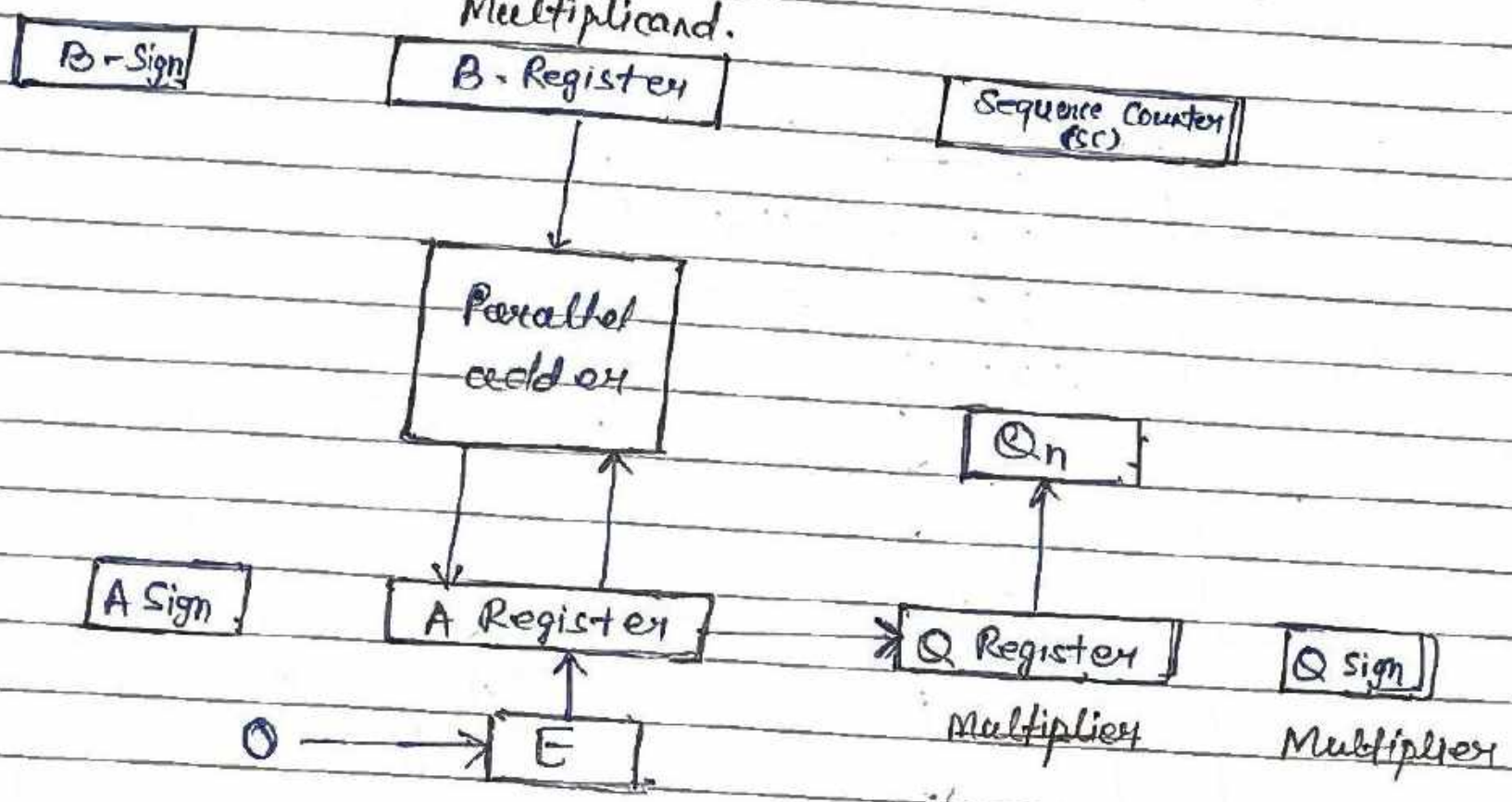
Ans = 11101
10011

C	A	Q	M	
	00000	10011	11101	
0	11101	100101	11101	Add } → First
0	01110	11001	11101	Shift } cycle
1	01011	11001	11101	Add } → second
0	10101	11100	11101	Shift } cycle
0	01010	11110	11101	Add } - Third cycle
0	00101	01111	11101	Add } - Fourth cycle
1	00010	01111	11101	Add } → Fifth cycle
0	10001	00111	11101	Shift }



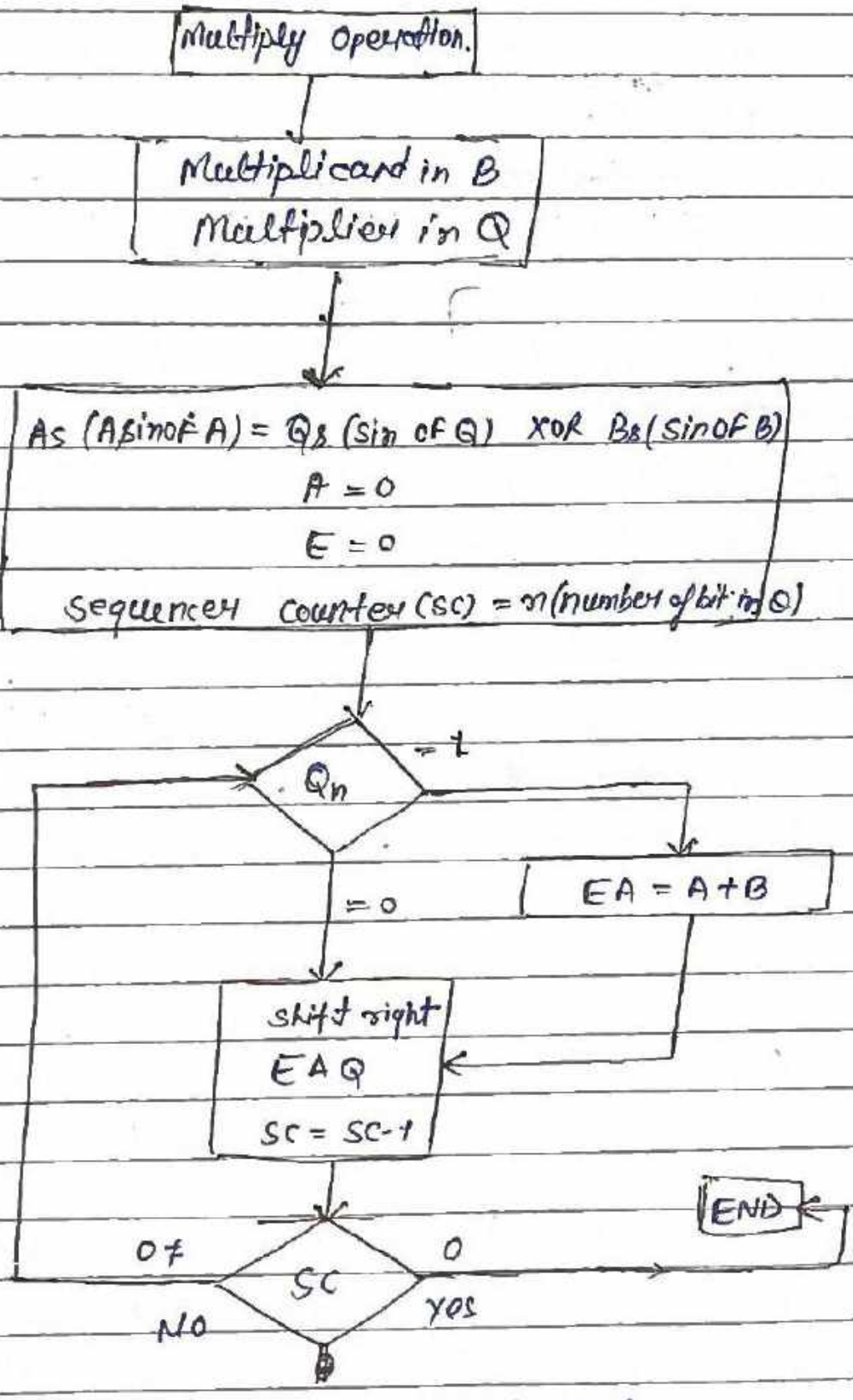
Flow chart for unsigned Binary No.

Signed operand Multiplication :-



1. The sign of the product is determine from the sign of the multiplicand and multiplier. If they are alike sign of the product is positive else negative.
2. Two register B and Q are used to store multiplicand and multiplier respectively.
3. Register A is used to store partial product during multiplication.
4. Sequence counter register (SC) is used to store number of bit in the multiplier.
5. To store sign bit of register 3 Flip-Flop are required. (A sign, B sign, Q sign).
6. Flip-Flop E is used to store carry bit generated during partial product addition.
7. Parallel adder is used in calculating partial order. That is perform addition required.

Flow chart of Multiplication :-



1.) Initially multiplicand is stored in B register and multiplier ~~regis~~ is stored in Q register.

2.) Sign of Register B and Q are compared using x-OR function. (that is if the both sign are alike output is zero else one) and output is store

in register A.

3.) Initially zero is assign to register A and E. sequence counter is initiallized with value n where n is the number of bits in the multiplied.

4.) Next LSB of Multiplier is checked, if a ~~it it is~~ case 1 add the content of register A with multiplicand and result is assign in A register with carry bit in flip-flop E. Content of E, A, Q is shifted to right by one position.

5.) If Qn equal to zero only right shift operation on content of E, A, & Q is perform.

6.) Content of sequence counter is decremented by one.

7.) Check the content of sequence counter. If it is zero end the process and the final product is present in register A and Q. If it is not zero repeat the process.

Multiplicand = 10111

Multiplier = 10011

B = 10111

10111
01011
100010

10111
00100
11011

E	A	Q	SC	
0	00000	10011	10	
0	10111	10011	100	Add
0	01011	11001		Shift
1	00010	11001	010	Add
0	10001	01100		shift
0	01000	10110	010	shift
0	00100	01011	001	Shift
0	11011	01011	000	Add
0	01101	10101		Shift

↳ Final Product

Multiplicand = 10011

Multiplier = 11001

```
 10011          10011
 00010          01010
 -----          -----
     0          11101
 10101
```

B = 10011	E	A	Q	SC	
	0	00000	11001	101	
	0	10011	11001	100	Add
	0	01001	11100		Shift
	0	00100	11110	011	Shift
	0	00010	01111	010	Shift
	0	10101	01111	001	Add
	0	01010	10111		Shift
	0	11101	10111	000	Add
	0	01110	11011		Shift.

Final Product

421

110

Multiplicand = 101110

Multiplier = 100110

011011

B = 101110	E	A	Q	SC	
	0	000000	100110	110	
	~~0~~	~~110110~~	~~100110~~	~~101~~	Add
	~~0~~	~~011011~~	~~010011~~	.	Shift
	0	000000	010011	101	shift
	0	101110	010011	0100	Add
	0	010111	001001		shift
	1	000101	001001	011	Add
	0	100010	100100		shift
	0	010001	010010	010	~~Add~~ shift
	0	001000	101001	001	Shift
	0	110110	101001	000	Add
	0	011011	010100		shift.

```
 101110          101110
 010111          001000
 ------          ------
 100010          110110
```

1's → 01101
10010
+ 1
00111

1024
116
896

Booth Algorithm:- The algorithm that is used to multiply binary integer in sign 2's complement form is called booth multiplication algorithm

- It works on the principle that string of zero (0) in the multiplier require no addition but just shifting and string of one (1) in the multiplier from bit weight 2^k to 2^m can be treated as $(2^{k+1} - 2^m)$ for

example :- binary number 001110 → +14

It has string of once from 2^3 to 2^1 (that is $k = 3$ and $m = 1$) then the number can be represented at

$2^{3+1} - 2^1$

$2^4 - 2^1$

14

Therefore the multiplication $M \times 14$ where M is the multiplicand and 14 is the multiplier can be done as $(M \times 2^4) - (M \times 2^1)$

Thus the product can be obtain by shifting the binary multiplicand M four time to the left and subtracting M shifted left once.

10 → M 0000 1010
×14 → Q 1st → 0001 0100
2nd → 0010 1000
3rd → 0101 0000
4th = 1010 0000

Multiplicand 0000 1010
1st shift 0001 0100
2's 1110 1011
+1
1110 1100 →

```
  10100000
+ 11101100
(1)10001100
```

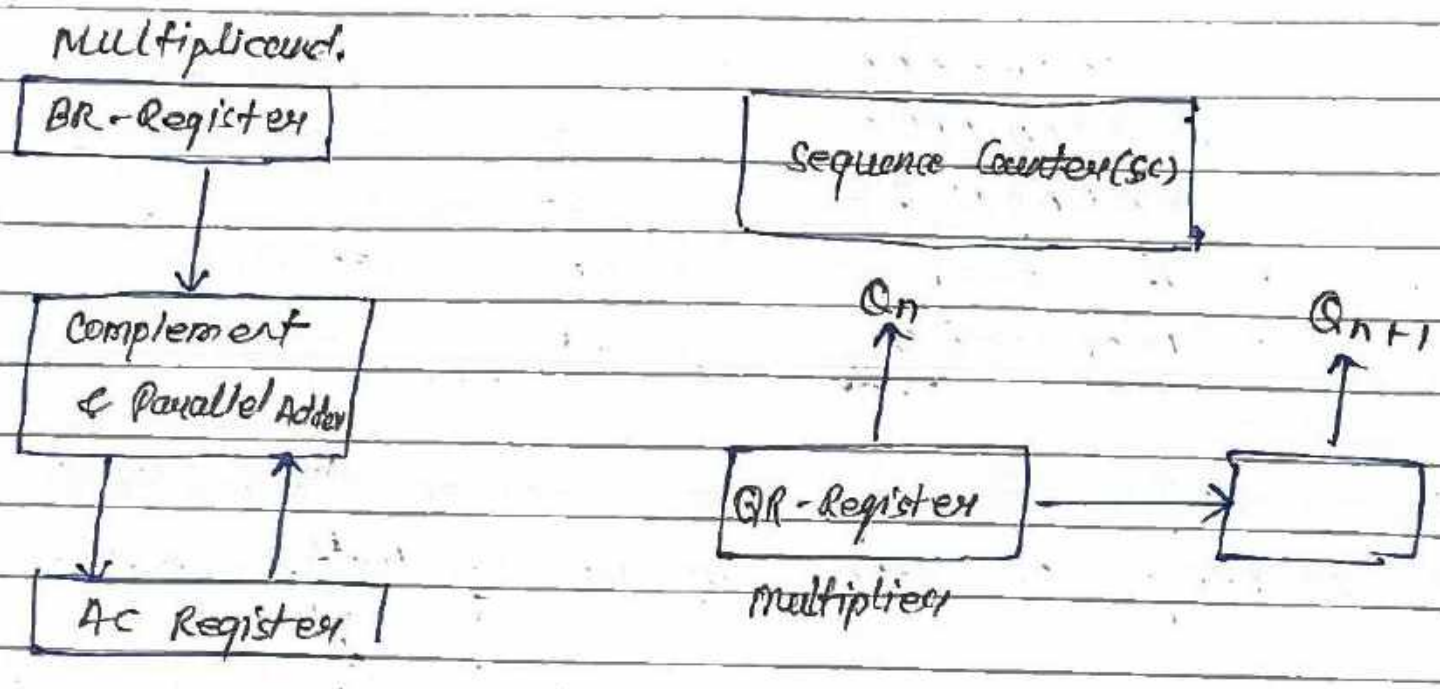
128 64 4 2 1

10001100

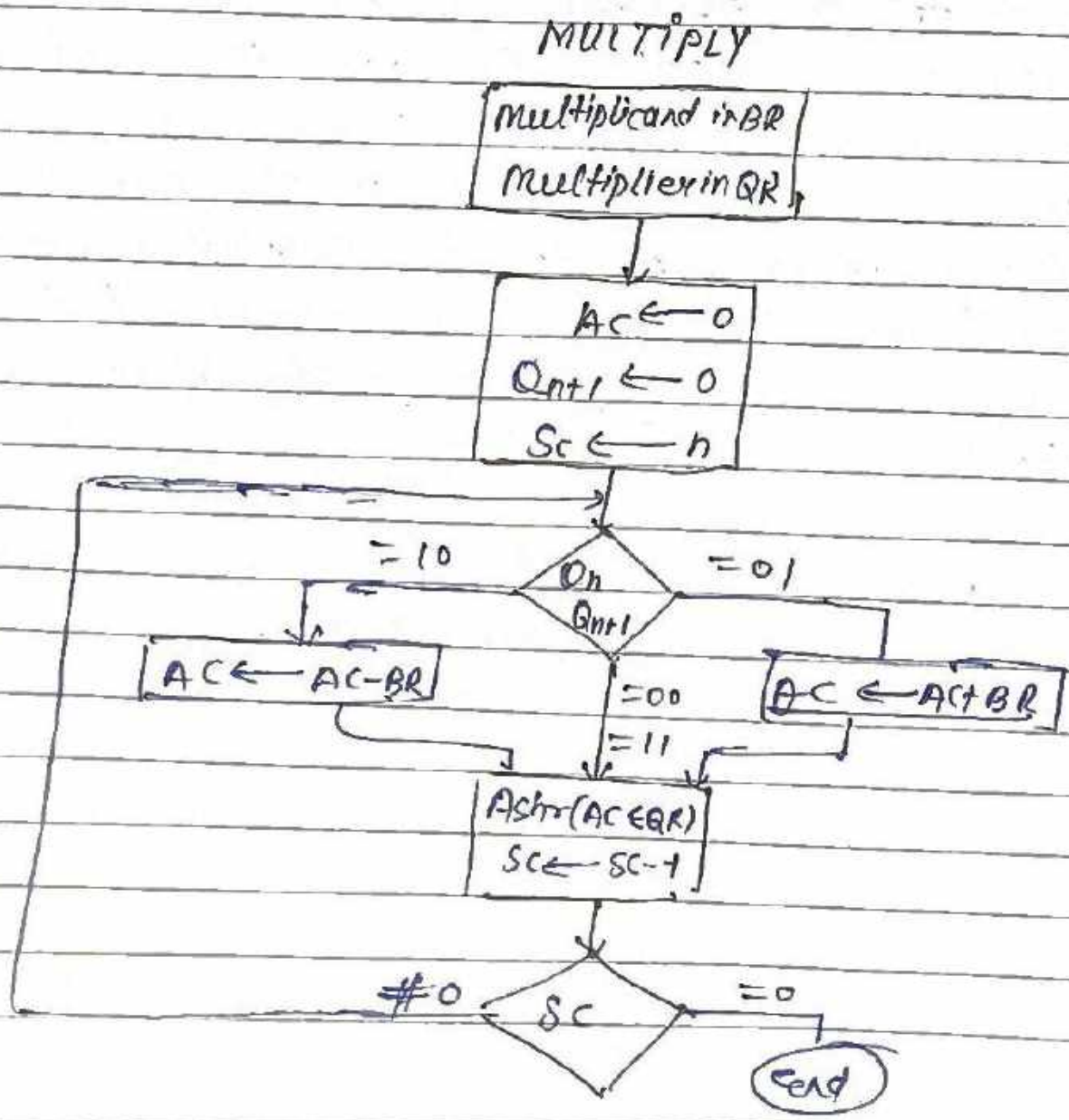
1

- Booth algorithm require examination of the multiplier bits and shifting of the partial product.
- Trial to the shifting the multiplicand may be added to the partial product or subtractive for the partial product or left unchanged according to following rule
  1. The partial product does not change when the multiplier bit is identical to previous multiplier bit
  2. The multiplicand is subtracted from the partial product upon encountering the least significant one (1) in the string of one in the multiplier
  3. The multiplicand is added to the partial product upon encountering the first zero (provided that there was a previous one) in a strings of zeros in the multiplier.

The hardware implementation of booth algorithm requires the register configuration as shown below.



An extra flip-flop Q_{n+1} is include to QR register to facilitate a double bit inspection of the multiplier. The flow chart for booth algorithm is shown below.



15 → 1101 → 0010 → -13 → 00011

1110
0111
0101

01111
+ 10000
10001 2's

Que-01:- Multiplier 7×3 using booth Algorithm.

Qn Qn+1	Multiplicand Br	AC	Multiplier BR	Qn+1	SC +1
	BR → 0111				
	Initial	0000	0011	0	100
10	AC ← A-Br	1001	0011	0	011
	AC ← AC+(-Br)				
	Ash	1100	1001	1	
11	Ash	1110	0100	1	010
10	AC ← AC+BR	0101	0100	1	001
	Ash	0010	1010	0	
00	Ash	0001	0101	0	000

AC + BR = 00010101

0111
1000
1
1001

Que - Multiply 15x-13 using booth algorithm.

01101
10010
+1
10011

-BR = 10001

Qn Qn+1	Multiplicand BR → 01111	AC	Multiplier (BR)	Qn+1	SC
	Initial	00000	10011	0	101
10	AC ← AC-Br	10001	10011	0	100
	AC ← AC+(-Br)				
	Ash	11000	11001	1	
11	Ash	11100	01100	1	011
01	AC ← AC+Br	00101	01100		010
	Ash	00101	10110	0	
00	Ash	00010	11011	0	001
10	AC ← AC-Br	10011	11011	0	000
	Ash	11001	11101	1	
11	Ash	11100	11110		

BR = 0111
1000
-BR ⇒ 1001

01010
10101
+1
10110
0001
1001
1010

-7 × 3   -BR ⇒ 10111

Q_n Q_{n+1}	Multiplicand (BR)	AC	Multiplier (QR)	Q_{n+1}	SC
	BR → 1001	0000	0011	0	100
10	AC → AC+BR	0111	0011	0	011
	Arsh	0011	1001	1	
11	Arsh	0001	1100	1	010
01	AC → AC+BR	1010	1100	1	001
	Arsh	1101	0110	0	
00	Arsh	1110	1011	0	000

-BR ⇒ 01111

11100
01111
01011

00111
10001
11000

-15 × -12

Q_n Q_{n+1}	Multiplicand (BR) BR ⇒ 10001	AC	Multiplier (QR)	Q_{n+1}	SC
	Initial	00000	10100	0	101
00	Arsh	00000	01010	0	100
00	Arsh	00000	00101	0	011
10	AC ← AC−BR / AC ← AC+(−BR)	01111	00101	0	010
	Arsh	00111	10010	1	
01	AC ← AC+BR	11000	10010	1	001
	Arsh	11100	01001	0	
10	AC ← AC+(−BR) / AC ← AC−BR	01011	01001	0	000
	Arsh	00101	10100	1	

Que-01) -14 × -9
Que-02) -13 × 9
Que-03) -18 × -19
Que-04) +8 × -12.

Qus-01 -14x-9

-BR = 01110

	Multiplicand (BR)	AC	Multiplier (QR)	Qn+1	SC
	BR → 10010				
	Initial	00000	10111	0	101
10	AC ← AC-BR AC ← AC+(-BR)	01110	10111	0	100
	Ashr	00111	01011	1	
11	Ashr	00011	10101	1	011
11	Ashr	00001	11010	1	010
01	AC ← AC+BR	10010	11010	1	001
	Ashr	11001	01101	0	
10	AC ← AC-BR AC ← AC+(-BR)	00111	01101	0	000
	Ashr	00011	10110		

AC&QR = 00011110

Qus-02 -13x9

-BR → 01101
multiplicand (BR)
BR → 10011

		AC			
	Initial	00000	01001	0	101
10	AC ← AC-BR AC ← AC+(-BR)	01101	01001	0	100
	Ashr	00110	10100	1	
01	AC ← AC+BR	11001	10100	1	011
	Ashr	11100	11010	0	
00	Ashr	11110	01101	0	010
10	AC ← AC-BR AC ← AC+(-BR)	01011	01101	0	001
	Ashr	00101	10110	1	
01	AC ← AC+BR	11000	10110	1	000
	Ashr	11100	01011	1	

AC&QR = 11100 1011

- BR → 01 0010

On → On+1	Multiplicand (BR)	AC	Multiplier (QR)	On+1	SC
	BR → 10111 0	000000	10111 0	0	~~110~~
Initial	Initial	000000	10110 1	0	11 0
10	AC ← AC+(-BR)	01 0010	10110 1	0	101
	Ash	00100 1	01011 0	1	
01	AC ← AC+BR	11011 1	0 0101 10	1	100
	Ash	111011	10101 1	0	
10	AC ← AC+(-BR)	10110 1	10101 1	0	011
	Ash	110110	11010 1	1	
11	Ash	111011	01101 0	1	010
01	AC ← AC+BR	101001	01101 0	1	001
	Ash	110100	10110 1	0	
10	AC ← AC+(-BR)	000110	10110 1	0	000
	Ash	000011	01011 0	0	

Que-5 BX-12

- BR → 11000

	Multiplicand (BR)	AC	Multiplier (QR)	On+1	SC
	BR → 01000		10100		
	Initial	00000	10100	0	101
00	Ash	00000	01010	0	100
00	Ash	00000	00101	0	011
10	AR → AC+(-BR)	11000	00101	0	010
	Ash	11100	00010	1	
01	AC → AC+BR	00100	00010	1	001
	Ash	00010	00001	0	
10	AC ← AC+(-BR)	11010	00001	0	000
	Ash	11101	00000	1	

AC & QR = 11101 00000

$-18 \rightarrow \underline{101110}$

$-19 \rightarrow 101101$

(24)

$AC \rightarrow AC + BR$

$AC \rightarrow \underline{AC + (-BR)}$

$-BR \rightarrow 010010$

Q_n, Q_{n+1}	Multiplicand (BR)	AC	Multiplier (QR)	Q_{n+1}	SC
	BR → 101110				
	Initial	000000	101101	0	110
10	AC ← AC + (−BR)	010010	101101	0	101
	Ashr	001001	010110	1	
01	AC ← AC + BR	111011	010110	1	100
	Ashr	111011	001011	0	
10	AC ← AC + (−BR)	001101	001011	0	011
	Ashr	000110	100101	1	
11	Ashr	000011	010010	1	010
01	AC ← AC + BR	110000	010010	1	001
	Ashr	111000	001001	0	
10	AC ← AC + (−BR)	001010	001001	0	000
	Ashr	000101	000100		

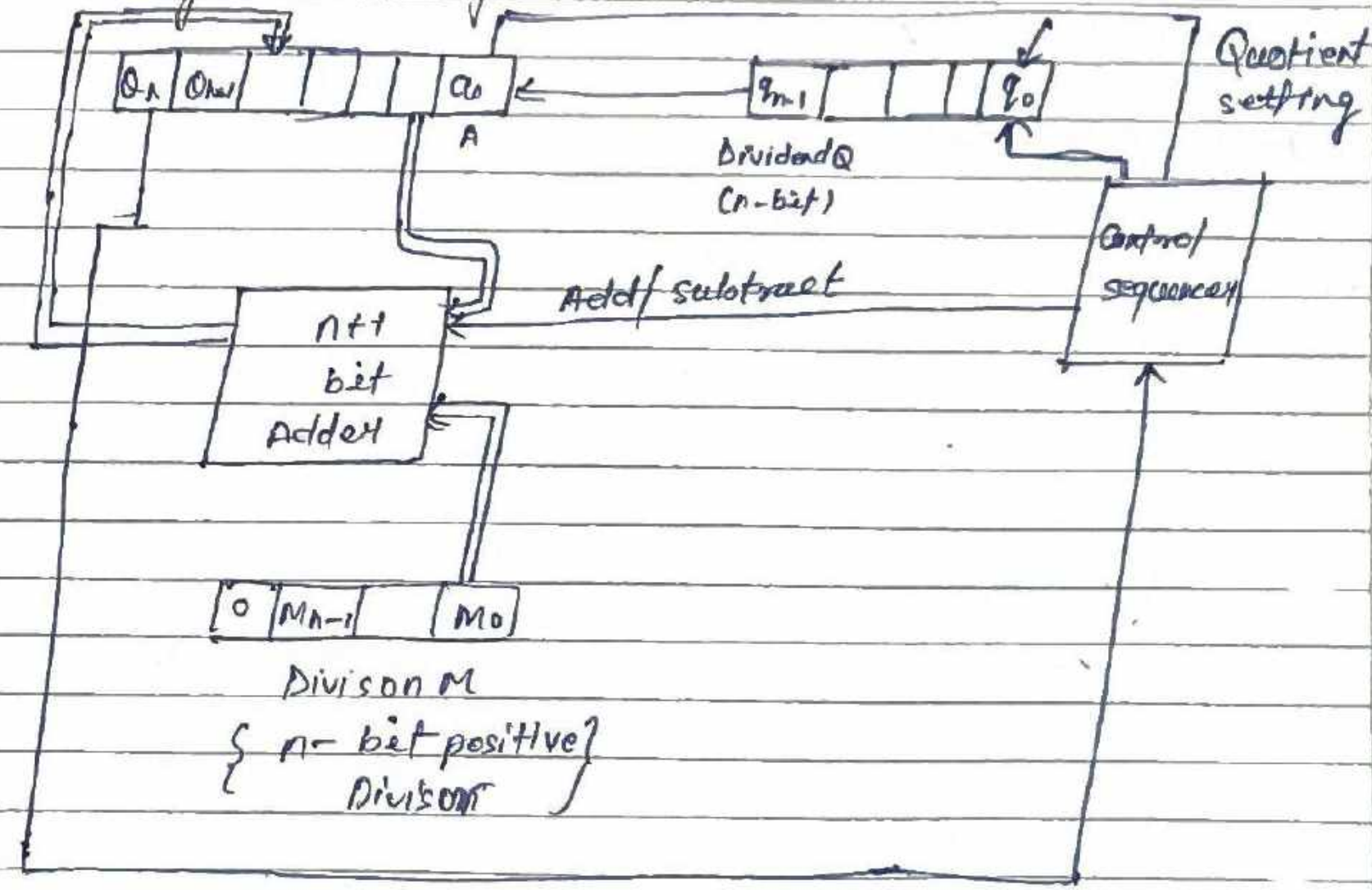
	−BR = 010010				
	BR → 101110				
	initial	000000	101101	0	110
10	AC ← AC + (−BR)	010010	101101	0	101
	− Ashr	001001	010110	1	
01	AC ← AC + BR	110111	010110	1	100
	Ashr	111011	101011	0	
10	AC ← AC + (−BR)	001101	101011	0	011
	Ashr	000110	110101	1	
11	Ashr	000011	011010	1	010
01	AC → AC + BR	110001	011010	1	001
	Ashr	111000	101101	0	
10	AC → AC + (−BR)	001010	101101	0	000
	Ashr	000101	010110	1	

Integer Division:- An algorithm which deal with two integer A & B, computes their reminder and quotient is called division algorithm.
Devision algorithm are of two types

(i) Slow Algorithm
   (a) Beast orin (a) Restoring Algorithm
   (b) Non-Restoring algorithm

(ii) Fast Algorithm.
   (a) Newton Raphson algorithm
   (b) SRT Division Algorithm.
   "Sweeney, Robertson and Tocher"

Restoring divison algorithm:-



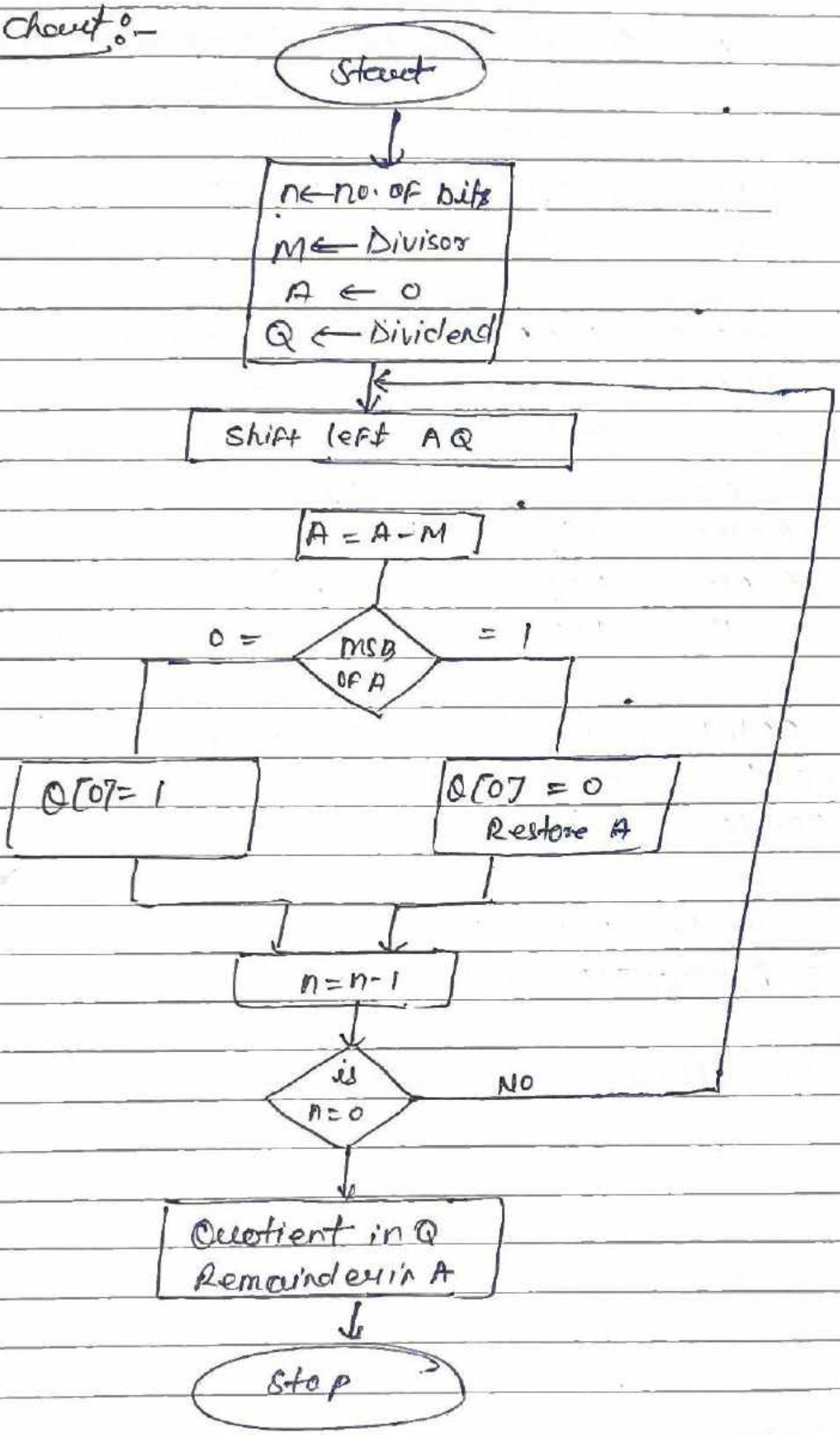
- Restoring algorithm is a traditional algorithm for performing binary division.
- In restoring division we restore the partial rimender after each subtraction if the result is negative (MSB is 1).
- above logic circuit shows the arrangement that implements the restoring devison algorithm
- An n bit positive division is loaded into register N and an an n bit positive dividend is loaded into register q at the start of operation.
- Register A is ~~set said~~ set to zero.
- After a divison is completed the n bit quotient is in register q and the remainder is in Register A.

Flow Chart :-

000

11101

Flow Chart :-



n	M	A	Q	Action / comment:
4	00011	00000	1011	Initilization
		00001	011?	Shift left A,Q
		11110	011?	A = A + (-M)
		00001	0110	MSB of A = 1. Restor A, Q[0]=0
3				n = n-1
		00010	110?	Shift left A,Q
		11111	110?	A = A + (-M)
		00010	1100	MSB of A = 1, Restore A, Q[0]=0
2				n = n-1
		00101	100?	Shift left A,Q
		00010	100?	A = A + (-M)
		00010	1001	MSB of A = 0, Q[0] = 1
1				n = n-1
		00101	001?	Shift left A,Q
		00010	001?	A = A + (-M)
0		00010	0011	MSB of A = 0, Q[0] = 1

Quotient in Q = 0011
Remainder A = 00010

Qu Devide 7 by 3 using Restoring Method.

0001
1101
1111

-M => 1101

0011
1101
000

×3

0011
+ 1101
10000

29

1101
0001
1110

0001
1101
1110

0010
1101
1111

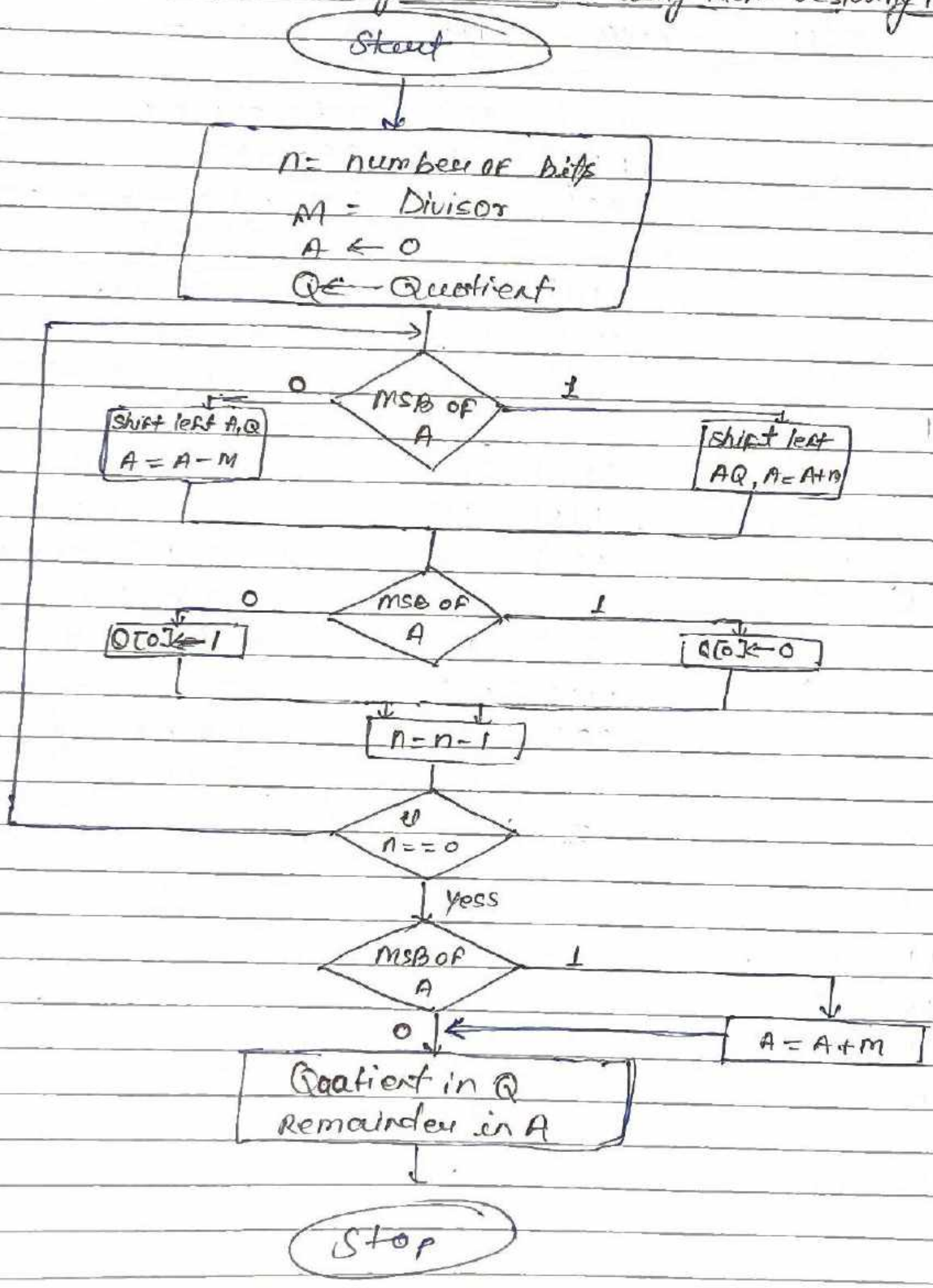
n	m	A	Q	Action/comment
4	0011	0000	1111	Initialized
3		0001	1117	Shift left A,Q
		1114	1117	AC + (-M)
		0001	1110	MSB of A=1 Restor n, Q[0]=0
2				n = n-1
		0011	1017	Shift left A,Q
		0000	1017	AC + (-M)
		0000	1015	MSB of A=0 Q[0]=1
1				n=n-1
		0001	0117	Shift left A,Q
		1110	0117	AC + (-M)
		0001	0110	MSB of A=1 Restor=A Q[0]=0
0				n=n-1
		0010	1007	Shift left A,Q
		1111	1007	AC = AC + (-M)
		0010	1000	MSB of A=1 Restor A Q[0]=0
				n=n-1

A = 0001

Q = 0100

0011
1100
+1
1101

Flow chart for integer divison using Non-restoring Method:-



00011 | 11111 | 00010

00001 | 11101 | 11110

00101 | 11101 | 00010

11110 + 00011 | 00001

11100 | 00011

Ques - 11 by 3 :-

-m → 11101

n	M	A	Q	Action/Comment.
4	00011	00000	1011	Initilization.
		00001	011?	Shift left A,Q
		11110	011!	A = A - M
		11110	0110	MSB of A = 1, Q[0] = 0
3				n = n - 1
		11100	110?	Shift left AQ
		11111	110	A = A + m
		11111	1100	MSB of A = 1 Q[0] = 0
2				n = n - 1
		11111	100?	Shift left AQ
		00010	100?	A = A + M
		00010	1001	MSB OF A = 0 Q[0] = 1
1				n = n - 1
		00101	001?	Shift left A,Q
		00010	001?	A = A - M
		00010	0011	MSB OF A = 0, Q[0] = 1
0				n = n - 1

Quotient = 0011

Remainder = 00010.

1101
+ 0011
10000

0001
1101
1110

~~11~~ 1101 → 0010
2's → 1100
-m → +1
1101

0011
1100
+1
1101

(32)
1110
0011
10001

0001
1101
1110

Devide 7 by 3

- m = 1101

n	m	A	Q	Action / Comments.
4 3	0011	0000	111	Initialization
		0001	111?	Shift left A,Q
		1110	111?	A = A - M
		1110	1110	MSB of A=1 Q[0]=0
2				n = n-1
		1101	110?	Shift left A,Q
		0000	110?	A = A + M
		0000	101	MSB of A = 0 Q[0]=1
1				n = n-1
		0001	01?	Shift left A,Q
		1111	01?	A = A - M
		1111	010	MSB of A = 1 Q[0]=0
0				n = n-1
		0001	010.	A = A + M

Quotient ⇒ 010
Remainder ⇒ 0001

Que-1	Divide	9 by 4	Restoring and Non Restoring method.
Que-2	Divide	11 by 5	
Que-3	Divide	14 by 6	
Que-4	Divide	16 by 7	
Que-5	Divide	19 by 6	

(0) $Q = 9 \Rightarrow 1001$

$M = 4 \Rightarrow 00100$

$-M \Rightarrow 11100$

Qu: 9 by 4

n	m	A	Q	Action / Comments
4	00100	00000	1001	Initialization
		00001	001?	Shift left A, Q
		11101	001?	A = A - M
		11101	0010	MSB of A = 1 Q[0] = 0
3				n = n - 1
		11010	010?	Shift left A, Q
		11110	010?	A = A + m
		11110	0100	MSB of A = 1 Q[0] = 0
2				n = n - 1
		11100	100?	Shift Left A, Q
		00000	100?	A = A + m
		00000	1001	MSB of A = 0, Q[0] = 1
1				n = n - 1
		00001	001?	Shift left A, Q
		11101	001?	A = A - M
		11101	0010	MSB of A = 1 Q[0] = 0
0				n = n - 1
		00001	0010	A = A + M

Quotient in Q -> 0010

Remainder in A -> 00001

$-m \Rightarrow 11011$
$Q = 11 \rightarrow 1011$
$m = 5 \rightarrow 00101$

Devide 11 by 5 Non-Restoring method :-

$-m \rightarrow 11011$

n	M	A	Q	Action / comment
4	00101	00000	1011	Initialization
		00001	011?	Shift left A,Q
		11100	011?	A = A+M
		11100	0110	MSB of A=1, Q[0]=0
3				n = n-1
		11000	110?	Shift left A, Q
		11101	110?	A = A+M
		11101	1100	MSB of A=1 Q[0]=0
2				n = n-1
		11011	100?	Shift left A, Q
		00000	100?	A = A+m
		00000	1001	MSB of A=0 Q[0]=1
1				n = n-1
		00001	001?	Shift left A,Q
		11100	001?	A = A-M
		11100	0010	MSB of A=1 Q[0]=0
				n = n-1
		00001	0010	A = A+M

Qoutient Q = 0010

Remainder R = 00001

(3)

Que-3 14 divide by 6 Q = 14 → 1110

Non-Restoring m = 6 → 0110 → 00110

-m 11010.

n	m	A	Q	Action / Comment
4	00110	00000	1110	Initialization.
		00001	110?	Shift left A,Q
		11011	110?	A = A-M
		11011	1100	MSB of A = 1 Q[0] = 0
3				n = n-1
		10111	100?	Shift left A,Q
		11101	100?	A = A+m
		11101	1000	MSB of A = 1 Q[0] = 0
2				n = n-1
		11011	000?	Shift left A,Q
		00001	000?	A = A+m
		00001	0001	MSB of A = 0, Q[0] = 1
1				n = n-1
		00010	0010?	Shift left A,Q
		11100	001?	A = A-M
		11100	0010	MSB of A = 1, Q[0] = 0
				n = n-1
		00010	0010	MSB of A = 1 A = A+M

Quotient Q = 0010

Remainder in A = 00010

Qu-4 Divide 16 by 7 Non storing Method.

Q = 16 → ~~10~~ 1110

M = 7 → 111 → 00111

-m → 00111 → 1's of M

2's → 11000

+1

11001

-m = 11001

n	m	A	Q	Action / Comments
4	00111	~~00000~~	1110	Initialization
		00001	110?	Shift left A,Q
		11010	110?	A ⇒ A - M
	3	11010	1100	MSB of A = 1 Q[0] = 0 n = n-1
		10101	100?	→ Shift left A,Q,
		11000	100?	→ A → A+M
	2	11000	1000	→ MSB of A = 1 Q(0) = 0 n=n-1
		10001	000?	Shift left A,Q
		11000	000?	A → A+M
		11000	0001	MSB of A=0, Q[0] = 1
	1			n = n-1
		10000	001?	Shift left A,Q
		11001	001?	A ⇒ A+M
		11001	0010	MSB of A = 1, Q[0] = 0
	0			n = n-1
				MSB of A = 1, A → A+M

(5) (37)

19 by 6

Q = 19 → 10111

m = 000110 ⟶ −m ⟶ 1's of m → 111001

−m = 111010. +1

−m → 111010

n	M	A	Q	Action / comments
5	000110	000000	10111	Initialization.
		000001	0111 ?	Shift left A,Q
		111011	0111 ?	A → A − m
		111011	01110	MSB of A = 1 Q[0] = 0
4				n = n−1
		110110	1110 ?	Shift left A,Q
		111100	1110 ?	A → A + m
		111100	11100	MSB of A = 1 Q[0] = 0
3				n = n−1
		111001	1100 ?	Shift left A,Q
		111111	1100 ?	A → A + m
		111111	11000	MSB of A = 1 Q[0] = 0
2.				n = n−1
		111111	1000 ?	Shift left A,Q
		000101	1000 ?	A → A + m
		000101	10001	MSB of A = 0 Q[0] = 1
1				n = n−1
		001011	0001 ?	Shift left A,Q
		000101	0001 ?	A → A − M
		000101	00011	MSB of A = 0 Q[0] = 1

(6) × (30)

Restoring Metho

9 by 4 Q = 9 → 1001

M = 4 → 00100

-M → 11010

n	M	A	Q	Action / Comment
4	00100	00000	1001	Initialization
		00001	001?	Shift left A,Q
		11011	001?	A → A−M
		00001	0010	MSB of A=1, restor A, Q[0]=0
3				n = n−1
		00010	010?	Shift left A,Q
		11100	010?	A → A−M
		00010	0100	MSB of A=1, restor A, Q[0]=0
2				n = n−1
		00100	100?	Shift left A,Q
		11110	100?	A → A−M
		00100	1000	MSB of A=1, restore A Q[0]=0
1				n = n−1
		01001	000?	Shift left A,Q
		00011	000?	A → A−M
		00011	0000	MSB of A=0, Q[0]=1
				n =

11011 ~
+1
11000

11011
+1
11100

(1) 11 by 5

$-M = 11011$

A	M	A	Q	action / comments
5	01011	00000	1011	Initialization
		00001	011?	shift left A, Q
		11100	011?	A = A − M
		00001	0110	MSB of A = 1 restore A Q[0] = 0
3				n = n − 1
		00010	110?	shift left A, Q
		11101	110?	A = A − M
		00010	1100	MSB of A = 1 restore, A Q[0] = 0
2				n = n − 1
		00101	100?	shift left A, Q
		00000	100?	A = A − M
		00000	1001	MSB of A = 0 Q[0] = 1
1				n = n − 1
		00001	001?	shift left
		11100	001?	A = A − M
		00001	0010	MSB of A = 1 restor A, Q[0] = 1

(2) 19 by 6

$-m \Rightarrow 11010$

n	m	A	Q	Action comments
4	00110	00000	1110	Initialization
		00001	110?	shift left A, Q
		11011	110?	A = A − M
		00001	1100	MSB of A = 1, Q[0] = 0 restore A
3				n = n − 1
		00011	100?	shift left A, Q
		11101	100?	A = A − M
		00011	1000	MSB of A = 1 Q[0] = 0 restor, A
2				n = n − 1

00111	000?	Shift left A,Q	
00001	000?	A = A − M	
00001	0001	MSB of A = 0 Q[0] = 1	
		n = n − 1	
000010	001?	Shift left A,Q	
11100	001?	A = A − M	
00010	0010	MSB of A = 1 Q[0] = 0 restore A	
		n = n − 1	

Q = 0010

A = 00010

(9) 16 by 7

① restoring −m → 111001

n	M	A	Q	Action / comment
5	000111	000000	10000	Initialization
		000001	0000?	Shift left A,Q
		111010	0000?	A = A − M
		000001	00000	MSB of A = 1 Q[0] = 0 restore A
4				n = n − 1
		000010	00000?	Shift left A,Q
		111010	0000?	A = A − M
		000010	00000	MSB of A = 1 Q[0] = 0 restore, A
3				n = n − 1
		00010	0000?	Shift left A,Q
		111101	0000?	A − M
		000100	00000	MSB of A = 1 Q[0] = 0 restor A
2				n = n − 1
		001000	0000?	Shift left
		000001	0000?	A = A − M
		000001	00001	MSB of A = 0 Q[0] = 1
1				n = n − 1

(9) (41)

	000010	0001?	Shift left A,Q
	111011	0001?	A = A - M
	000010	00010	MSB of A = 0 Q[0] = 0 restore A
0			n = n - 1

(10)

19 bytes −m = 111010

n	M	A	Q	Action / Comments
5	000110	000000	10011	Initialization
		000001	0011?	shift left
		111011	0011?	A = A - M
		000001	00110	MSB of A = 1 Q[0] = 1 restore - A
4				n = n - 1
		000010	0110?	Shift left A,Q
		111100	0110?	A = A - M
		000010	00100	MSB of A = 1 Q[0] = 0 restor = A
3				n = n - 1
		000100	1100?	Shift left A, Q
		111110	1100?	A = A - M
		000100	11000	
2				n = n - 1
		001001	1000?	Shift left A,Q
		000011	1000?	A = A - M
		000011	10001	MSB of A = 0 Q[0] = 1
1				n = n - 1
		000111	0001?	Shift left A,Q
		000001	0001?	A = A - M
		000001	00011	MSB of A = 0 Q[0] = 1
0				n = n - 1

IEEE (Floating Point Representation)

① Single Precision (32 bits)
② Double Precision (64 bits)

( ) Any base ⟶ ( )$_2$ ⟶ Normalised Form.

Single Precision.

① $(54)_{10}$

2	54	0
2	27	1
2	13	1
2	6	0
2	3	1
	1	

$(54)_{10} \longrightarrow (110110)_2$

Normalised form = 1.10110×2^5

Exponent = E' + 127
= 5 + 127
Exponent = 132

2	132	0
2	66	0
2	33	1
2	16	0
2	8	0
2	4	0
2	2	0
	1	

Exponent = 10000100
mantisa = 10110

Sign	Exponent	mantisa.
0	10000100	10110.....0

1 bit ← 8 bit → ← 23 bits →

$(35)_{10}$

2	35	1
2	17	1
2	8	0
2	4	0
2	2	0
	1	

$(35)_{10} \longrightarrow 100011$

Normalised form = 1.00011×2^5

Exponent = E'+127

= 5+127

Exponent = 132

2	132	0
2	66	0
2	33	1
2	16	0
2	8	0
2	4	0
2	2	0
	1	

Exponent = 10000100

mantisa = 00011000

Sign	Exponent	Mantisa
0	10000100	00011.....0
← 1 bit →	← 8 bit →	← 23 bit →

$(-54)_{10}$

2	54	0
2	27	1
2	13	1
2	6	0
2	3	1
	1	

$(54) \longrightarrow 110110$

Normalised form 1.10110×2^5

Exponent = E'+127

= 5+127

=132

2	132	0
2	66	0
2	33	1
2	16	0
2	8	0
2	4	0
2	2	0
	1	

Exponent = 10000100

mantisa = 10110

sign	exponent	mantisa
1	10000100	10110.....0
1 bit	8 bit	23 bit

(2)

$(-234.725)_{10}$

2	234	0
2	117	1
2	58	0
2	29	1
2	14	0
2	7	1
2	3	1
	1	

$0.725 \times 2 = 1.450 \Rightarrow 1$

$.450 \times 2 = 0.90 \Rightarrow 0$

$.90 \times 2 = 1.8 \Rightarrow 1$

$.8 \times 2 = 1.6 \Rightarrow 1$

$(-234.725)_{10} \longrightarrow (11101010.1101)_2$

Normalised form $\Rightarrow$ 1.11010101101×2^7

Exponent = $E' + 127$

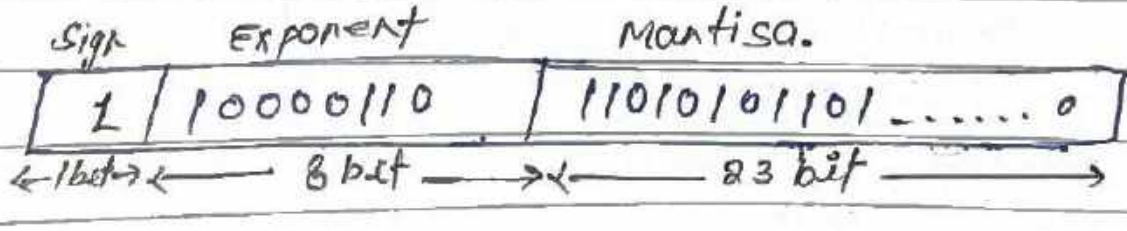
$= 7 + 127$

Exponent = 134

2	134	0
2	67	1
2	33	1
2	16	0
2	8	0
2	4	0
2	2	0
	1	

Exponent = 10000110

Mantisa = 11010101101



(0.25)10:

Unnormalized form → 0.1×2^{-6}

Exponent = E+127

= -6+127

= 121

Exponent = 1111001

Mantisa = 1010

Sign	Exponent	Mantisa
0	01111001	1010 0 ... 0

1 bit ← 8 bit → ← 23 bits →

2 | 121	1
2 | 60	0
2 | 30	0
2 | 15	1
2 | 7	1
2 | 3	1
	1

Double Precision :-

$(-25)_{10}$

2 | 25	0
2 | 12	1
2 | 6	0
2 | 3	1
2 | 1	1
	1

$(-25)_{10} \rightarrow (11001.0)_2$

Normalized form (1.1001×2^4)

Exponent = E+1023

= 4+1023

= 1027

2 | 1027	1
2 | 513	1
2 | 256	0
2 | 128	0
2 | 64	0
2 | 32	0
2 | 16	0
2 | 8	0
2 | 4	0
2 | 2	0
	1

Exponent = 10000000011

mantisa 1001 0 ... 0

Sign	Exponent	Mantisa
1	10000000011	1001 0 ... 0

1 bit | 11 bits | 52 bit

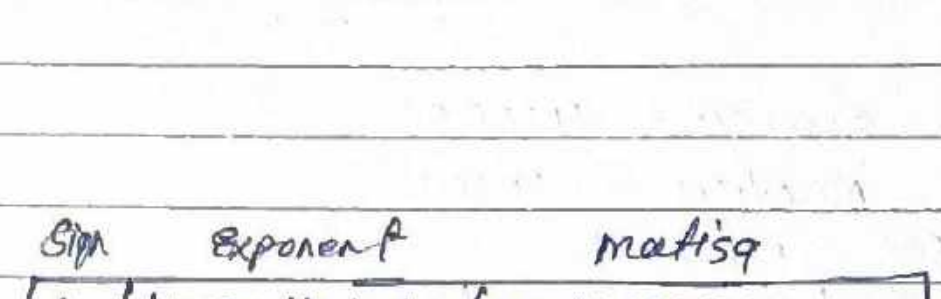
$1.00010100 \times 2^{-10}$

Exponent = E' + 1023

= -10 + 1023

Exponent = 1013

2	1013	1
2	506	0
2	253	1
2	126	0
2	63	1
2	31	1
2	16	0
2	8	0
2	4	0
2	2	0
	1	



Exponent 10000110101

matisa