No.

DATE

Multiplication of positive numbers
Binary Multiplier

Eg: 1101 (13) 1011 (11)

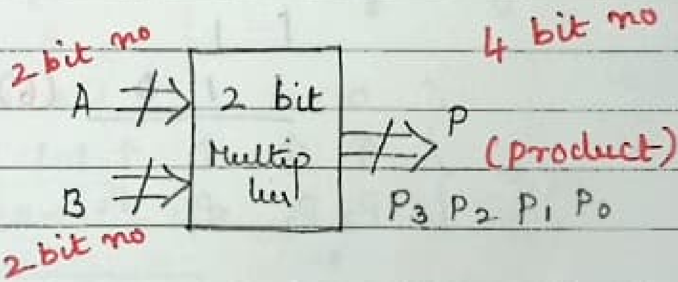
Multiply 2 bit numbers

$$
\begin{array}{cccccccc}
 & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 0 & 1 & 1 \\
\hline
 & & & & 1 & 1 & 0 & 1 \\
 & & & 1 & 1 & 0 & 1 & \\
 & & 0 & 0 & 0 & 0 & & \\
 & 1 & 1 & 0 & 1 & & & \\
\hline
\boxed{1} & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\hline
128 & 64 & 32 & 16 & 8 & 4 & 2 & 1
\end{array}
$$

128 + 8 + 4 + 2 + 1 = 143

$$
\begin{array}{r} 128 \\ 15 \\ \hline 143 \end{array}
$$

$$
\begin{array}{cccc}
 & & 2^1 & 2^0 \\
A & & A_1 & A_0 \\
B & & B_1 & B_0 \\
\hline
C_2 & C_1 & A_1B_0 & A_0B_0 \\
 & A_1B_1 & A_0B_1 & \\
\hline
C_2 & A_1B_1 & A_1B_0 + & A_0B_0 \\
 & C_1 & A_0B_1 & \\
 & C_2 & C_1 & \\
\hline
\end{array}
$$



Half Adder

$X + Y$

Sum $= X \oplus Y$

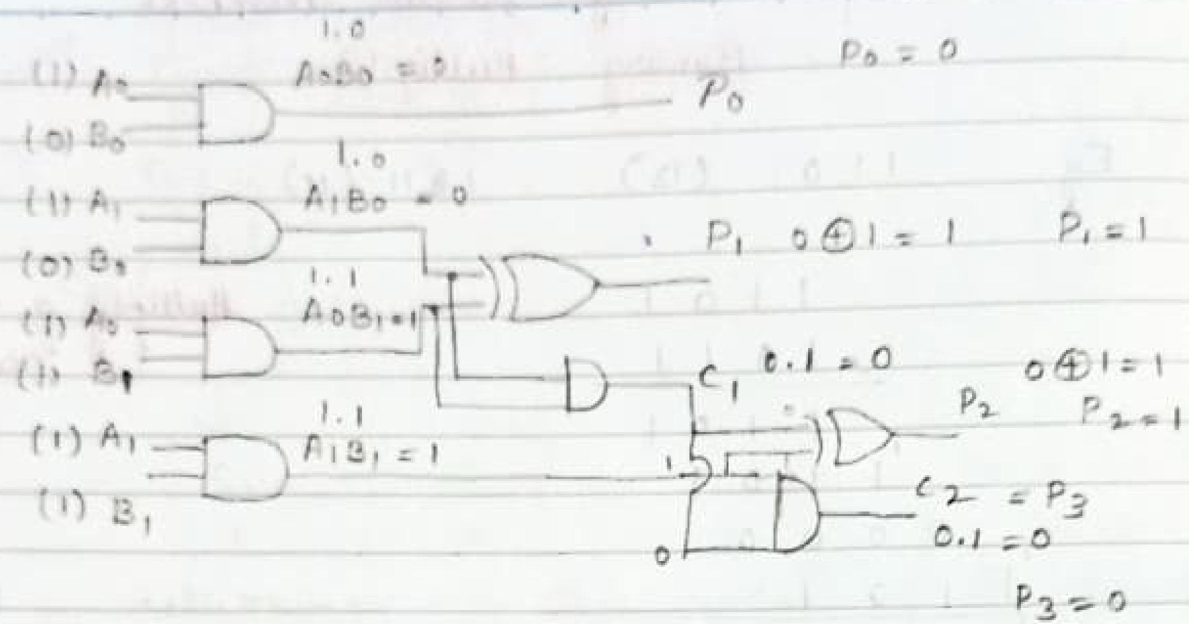
Carry $= X \cdot Y$

$P_0 = A_0B_0$

$P_1 = A_1B_0 + A_0B_1$ (HA)

$P_2 = A_1B_1 + C_1$ (HA)

$P_3 = C_2$



A_1 A_0

A 1 1 (3)

B B_1 1 0 B_0 (2)

0 0

1 1

0 1 1 0 (6)

↑ ↑ ↑ ↑

P_3 P_2 P_1 P_0

0110 (6)

~~01~~ 1 0

~~01~~ 1 0

0 0

1 0

1 0 0

Shift and Add Multiplier

Multiply 11 (Multiplicand) and 13 (Multiplier) using add Shift method

Multiplier (Q)

11 × 13 = 143

Multiplicand (M)

(AQ) product

N = 4

N − 1

N	M	C	A	Q (perform Add)	Operation
4	1011	0	0000	1101	Initialization
					First cycle
		0	M+A 1011	1101	Add M with A
		0	0101	1110	A = A+M
					Shift Right CAQ
3		0	0010	1111	Second cycle
					Shift Right CAQ
2		0	M+A 1101	1111	Third cycle
					Add M with A
		0	0110	1111	Shift Right CAQ
					Fourth cycle
1		1	M+A 0001	1111	Add M with A
		0	1000	1111	Shift Right CAQ

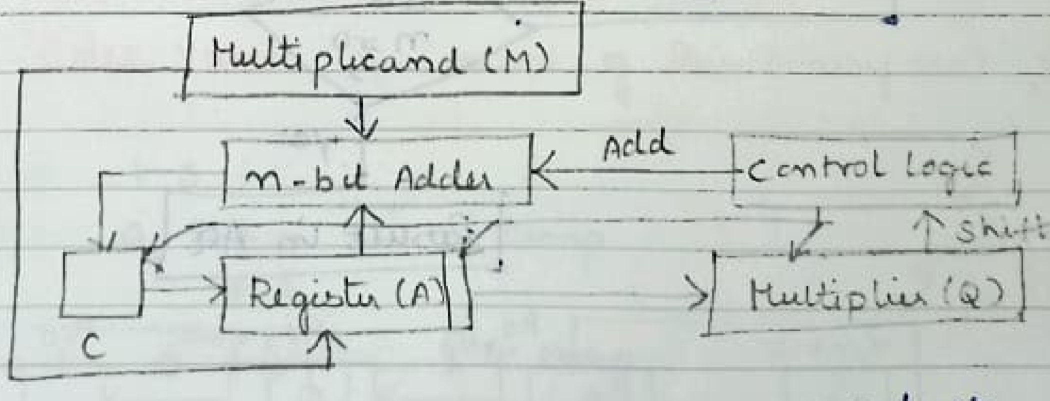
0010
1011
1101

0110
1011
10001

128 64 32 16 8 4 2 1

1000 1111

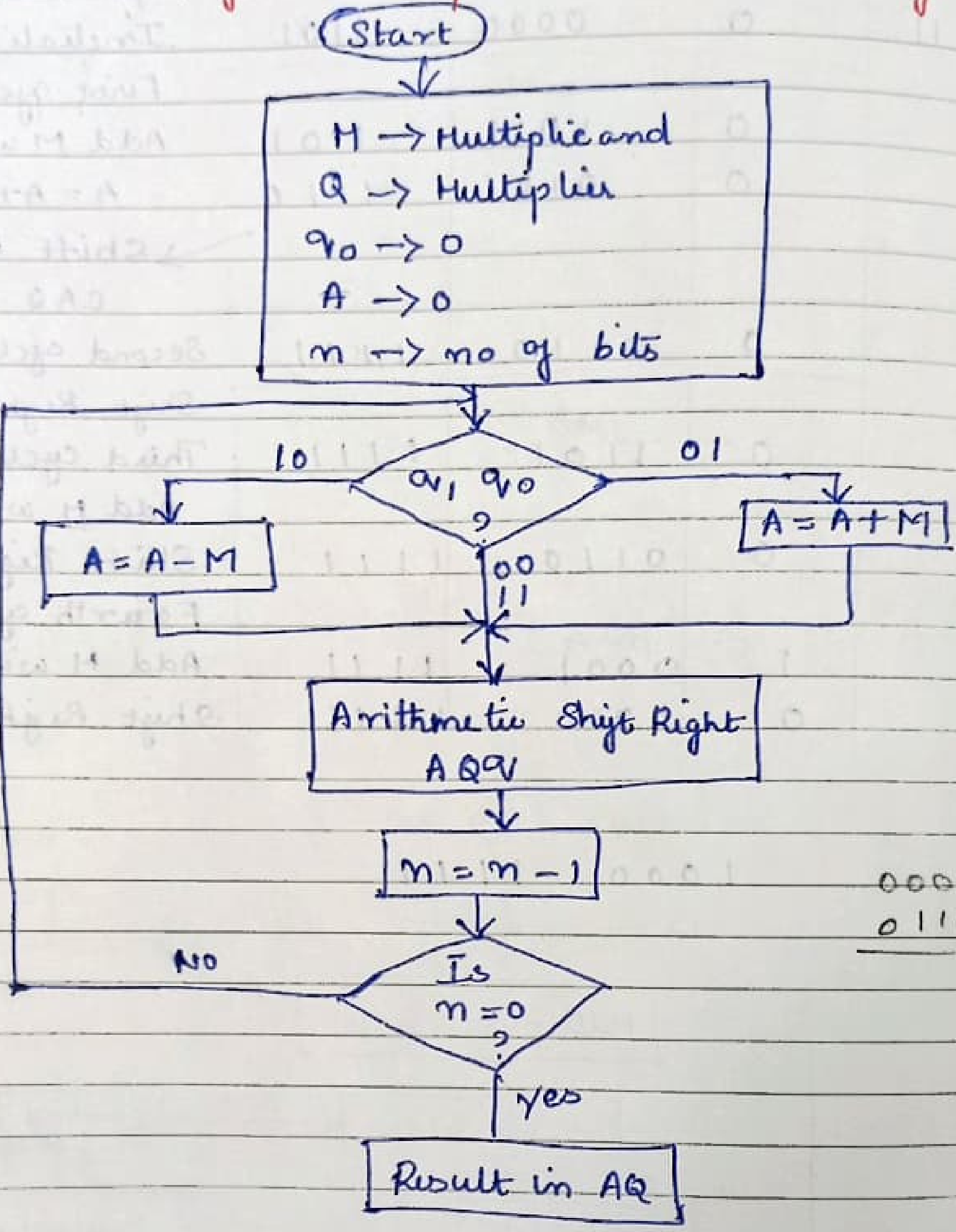
128+8+4+2+1 = 143



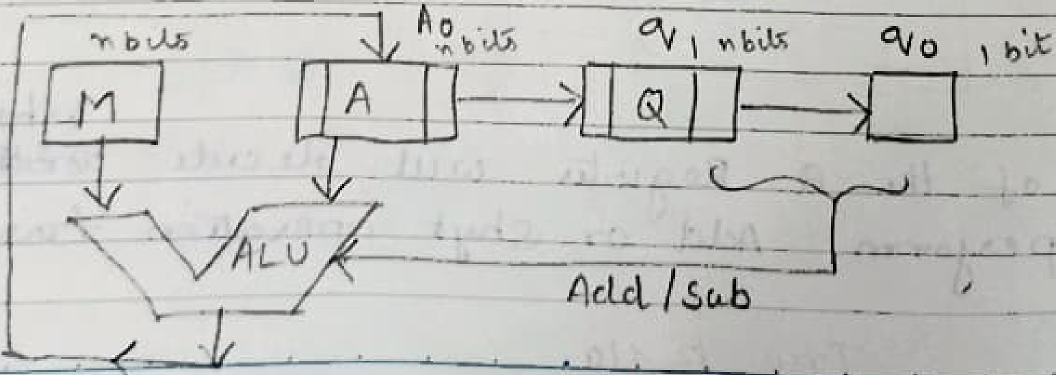
LSB of the Q Register will decide ~~whether~~ whether to perform Add or Shift operation first

Try 12×10.

Signed Multiplication – Booth Alg



0000
0111



Hardware structure – Implementing Booth's Algo

A-(-M)
A+M
0000+
0111
0111

NO.
DATE

Eq: (-7) x (+3) = (-21)

(-7) → Multiplicand
(+3) → Multiplier
(-21) → product

M = -7

2's of 0111_2

1001_2

$-M = 0111_2$

Tracing Table

holds single bit

n	A	Q	q_0	Action / Comments
4	0000	0011	0	Initialization
	0111	0011	0 → LSB is lost	A = A - M
3	0011	1001	1	ASR AQq_0
2	0001	1100	1	ASR AQq_0
	1010	1100	1	A = A + M
1	1101	0110	0	ASR AQq_0
0	1110	1011	0	ASR AQq_0

AQ - product

1001
0001
1010

MSB - 1 (value is -ve)
do 2's complement of the outcome
MSB - 0 (dec value equivalent of this binary no)

1110 1011
0001 0100 - 1's comp
+1
0001 0101 - 2's comp
128 64 32 16 8 4 2 1

16 + 4 + 1 = 21

(-21)

Booth Recording Algo

1 1 1 1 1 0 → imaginary zero added

(1 1) → 0
(1 0) → +1
(0 1) → −1
(0 0) → 0

−1 0 0 0 +1

+1 0 0 0 −1

$\boxed{01111}$

16	8	4	2	1
0	1	1	1	1

15

2^4	2^3	2^2	2^1	2^0
+1	0	0	0	−1
16×1	8×0	4×0	2×0	1×−1
16	0	0	0	−1

$16 + (-1) = 15$

Advantage

when we have more Zero's number of additions will be less

Shifts cannot be reduced

Eg: 1 1 0 1 1 0 1 1 0

0 −1 +1 0 −1 +1 0 −1

−128	64	32	16	8	4	2	1
1	1	0	1	1	0	1	1

$-128 + 64 + 16 + 8 + 2 + 1 = -37$

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

$= -64+32-8+4-1$

$= -32-8+3$

$= -40+3 = -37$

Eg1: 0 1 1 1 1 0

0 1 1 1 1 0 0 0

+1 0 0 0 -1 0

32	16	8	4	2	1
0	1	1	1	1	0

30 ✓

32	16	8	4	2	1
+1	0	0	0	-1	0

$32+(-2) = 30$ ✓

0 1 1 1 1 1 0

+1 0 0 0 0 -1

32

32

Eg2: 0 1 1 0 0 1 0 0

0 1 1 0 0 1 0 0 0

+1 0 -1 0 +1 -1 0 0

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

$64+32+4 = 100$ ✓

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

$128+(-32)+8+(-4)$

$96+4 = 100$ ✓

1 1 1 1 0 1 0 1 0

0 0 0 -1 +1 -1 +1 -1

16 8 4 2 1

$-16+8-4+2-1$

$= -21+10$

$= -11$

NO.

DATE

Eg 3: 0 0 1 1 1 1 0 1 1 0 0 1

0 0 1 1 1 1 0 1 1 0 0 1 0

0 +1 0 0 0 -1 +1 0 -1 0 +1 -1

2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	1	1	1	1	0	1	1	0	0	1

$512 + 256 + 128 + 64 + 16 + 8 + 1$

985

2048	1024	512	256	128	64	32	16	8	4	2	1
0	+1	0	0	0	-1	+1	0	-1	0	+1	-1

$1024 + (-64) + 32 + (-8) + 2 + (-1)$

$= 985$

Booth Multiplication

1. It reduces the number of additions [most of the time]
2. -ve multiplier, +ve multiplier are treated same

No.

DATE

0 1 1 (3) 3 = 4 − 1
0 1 1 × (3) → Multiplier is written in a different form

~~0 1 1~~ .

~~0 1 −1~~

~~2^2 2^1 2^0~~

~~0 1 −1~~

0 1 1   1 0 −1

2^2 2^1 2^0
1 0 −1

4 + 0 + (−1)
4 − 1 = 3

0 1 1 x
1 ×
0 1 1 x

0 1 1 x
−1
−x

		0	1	1	3	
		1	0	−1	3 ×	
1	1	1	1	0	1	9
0	0	0	0	0		
0	0	1	1			
0	0	1	0	0	1	
2^5	2^4	2^3	2^2	2^1	2^0	

8 + 1 = 9

(0 1 1) − (0 1 1)

1 0 0
+1   2's comp
1 0 1   − 3

Eg: 1 0 1 1 (-5)
1 1 1 1 (-1)

8	4	2	1
0	0	0	-1

$-1 = 0\ 0\ 0\ -1$

$0+0+0+(-1)$
$= -1$

```
        1 0 1 1        -5
        0 0 0 -1       -1  x
  -----------------    ----
0 0 0 0 1 0 1           5
0 0 0 0 0 0
0 0 0 0 0
0 0 0 0
-----------------
0 0 0 0 1 0 1   (5)
```

$x \times -1 = -x$

```
 1 0 1 1
 0 1 0 0
     +1
 -------
 0 1 0 1
```

```
0 1 0 0
      1
-------
0 1 0 1
```

2 bits Combination

6 bits Multiplier means recoded also 6 bits

NO.
DATE . .

Fast Multiplication

Speed up the multiplication operation

Bit - Pair Recoding of Multipliers

eg: $6 \quad 6/2 = 3$

n bits in multiplier recoded value $n/2$ bits

Multiplier bit - pair		Multiplier bit on the right	Multiplicand detected at pos i
$i+1$	i	$i-1$	
0	0	0	$0 \times M$
0	0	1	$+1 \times M$
0	1	0	$+1 \times M$
0	1	1	$+2 \times M$
1	0	0	$-2 \times M$
1	0	1	$-1 \times M$
1	1	0	$-1 \times M$
1	1	1	$0 \times M$

eg:

$$\underbrace{\overset{i+1}{1}\ \overset{i}{1}\ \overset{i-1}{1}}_{0}\quad \underbrace{\overset{i+1}{1}\ \overset{i}{0}\ \overset{i-1}{1}}_{-1}\quad \underbrace{\overset{i+1}{1}\ \overset{i}{0}\ \overset{i-1}{\boxed{0}}}_{-2}$$

1 1 1 0 1 0 0

2 | 27 - 1
2 | 13 - 1
2 | 6 - 0
2 | 3 - 1
1

Eq

	A	B
Multiply	-11	and $+27$

10100
1
10101

-11
$A = 01011$
$B = 11011$
27

A = 1 1 0 1 0 1 (2's comp of 11)
B = 0 1 1 0 1 1 (+27)

Add sign bit — (1) + (0) in MSB

Recording can be done for multiplier

$$\underbrace{0\ 1\ 1}_{+2}\ \underbrace{0\ 1\ 1}_{-1}\ \underbrace{1\ \boxed{0}}_{-1}$$

Multiplier / Multiplicand

n bits – output 2n

```
            1  1  0  1  0  1   x
              +2    -1    -1
  -------------------------------
  0 0 0 0 0 0 0  0 1 0 1 1   (2's comp)
  0 0 0 0 0 0 1  0 1 1       (2's comp) Multiplicand
  1 1 0 1 0 1 0              (Multiplicand x 10)
  -------------------------------
  1 1 1 0 1 1 0 1 0 1 1 1
  -------------------------------
```

(-297)

+2 = 10

Multiplicand x 10

```
    1 1 0 1 0 1
             1 0
  -------------
    0 0 0 0 0 0
  1 1 0 1 0 1
  -------------
  1 1 1 0 1 0 1 0
```

111

NO.

DATE

2. Multiply +13 & −6

0110
1001
1
1010

Multiplicand = +13 ⟶ 01101
Multiplier = −6 ⟶ 11010 (2'S comp)

↑ Last bit is 1 extend by 1

$\underbrace{1\ 1\ 1}_{0}\ \underbrace{1\ 0\ 1}_{-1}\ \underbrace{1\ 0\ \boxed{0}}_{-2}$

1101
0110

01101
11010

```
          0 1 1 1 0 1
          0    -1   -2
    ---------------------
1 1 1 1 1 0 0 1 1 0   (2'S comp of Multiplicand x10)
1 1 1 1 0 0 1 1
0 0 0 0 0 0
---------------------
X 1 1 1 0 1 1 0 0 1 0
Ignore
---------------------      (−78)
```

$$\begin{array}{r|l} 2 & 78 \\ 2 & 39 - 0 \\ 2 & 19 - 1 \\ 2 & 9 - 1 \\ 2 & 4 - 1 \\ 2 & 2 - 0 \\ 2 & 1 - 0 \end{array}$$

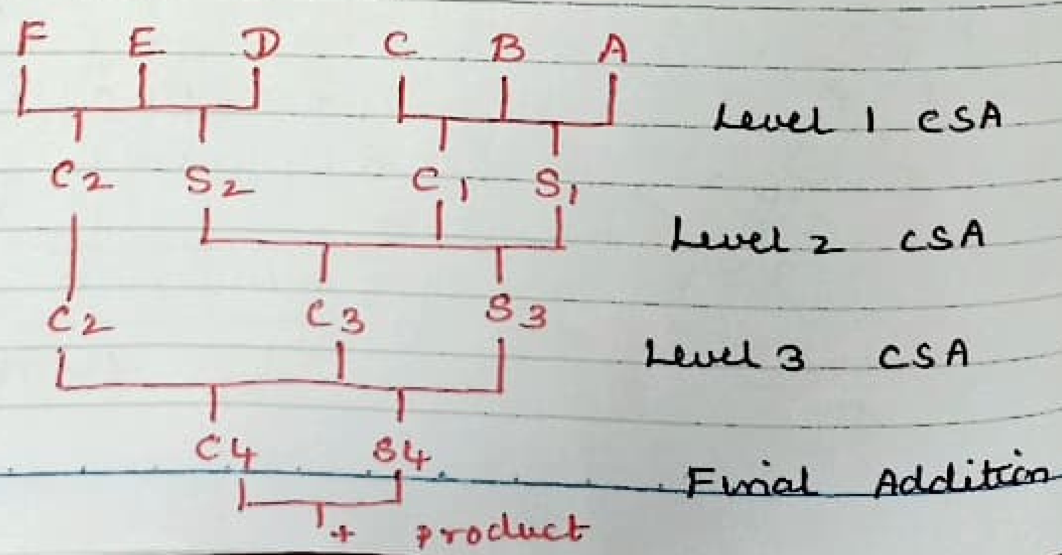
```
  1001110   (78)
  0110001   (take 2'S comp of 78)
        1
---------
  0110010
---------
  (−78)
```

Carry Save Addition (CSA) of Summands

used to compute sum of 3 or more binary numbers

CSA or 3-2 Adder very fast and cheap adder does not propagate carry bits

```
          1 0 1 1 0 1          (45)  M
          1 1 1 1 1 1          (63)  Q
     --------------------
          1 0 1 1 0 1              A
        1 0 1 1 0 1                B
      1 0 1 1 0 1                  C
    1 0 1 1 0 1                    D
  1 0 1 1 0 1                      E
1 0 1 1 0 1                        F
---------------------------
1 0 1 1 0 0 0 1 0 0 1 1        (2835) product
```



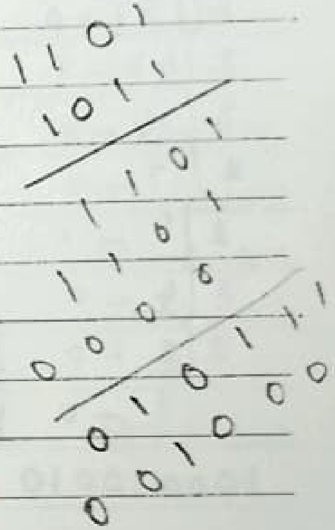
1 0 1 1 0 1
1 1 1 1 1 1

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

1 1 0 0 0 0 1 1 S_1
0 0 1 1 1 1 0 0 C_1

1 0 1 1 0 1 D
1 0 1 1 0 1 E
1 0 1 1 0 1 F

1 1 0 0 0 0 1 1 S_2
0 0 1 1 1 1 0 0 C_2



1 1 0 0 0 0 1 1
0 0 1 1 1 1 0 0
1 1 0 0 0 0 1 1

1 1 0 1 0 1 0 0 0 1 1 S_3
0 0 0 0 1 0 1 1 0 0 0 C_3
0 0 1 1 1 1 0 0 C_2

0 1 0 1 1 1 0 1 0 0 1 1 S_4
0 1 0 1 0 1 0 0 0 0 0 C_4

1 0 1 1 0 0 0 1 0 0 1 1 product

Integer Division

```
      21
13 | 274
     26
     --
      14
      13
      --
       1
```

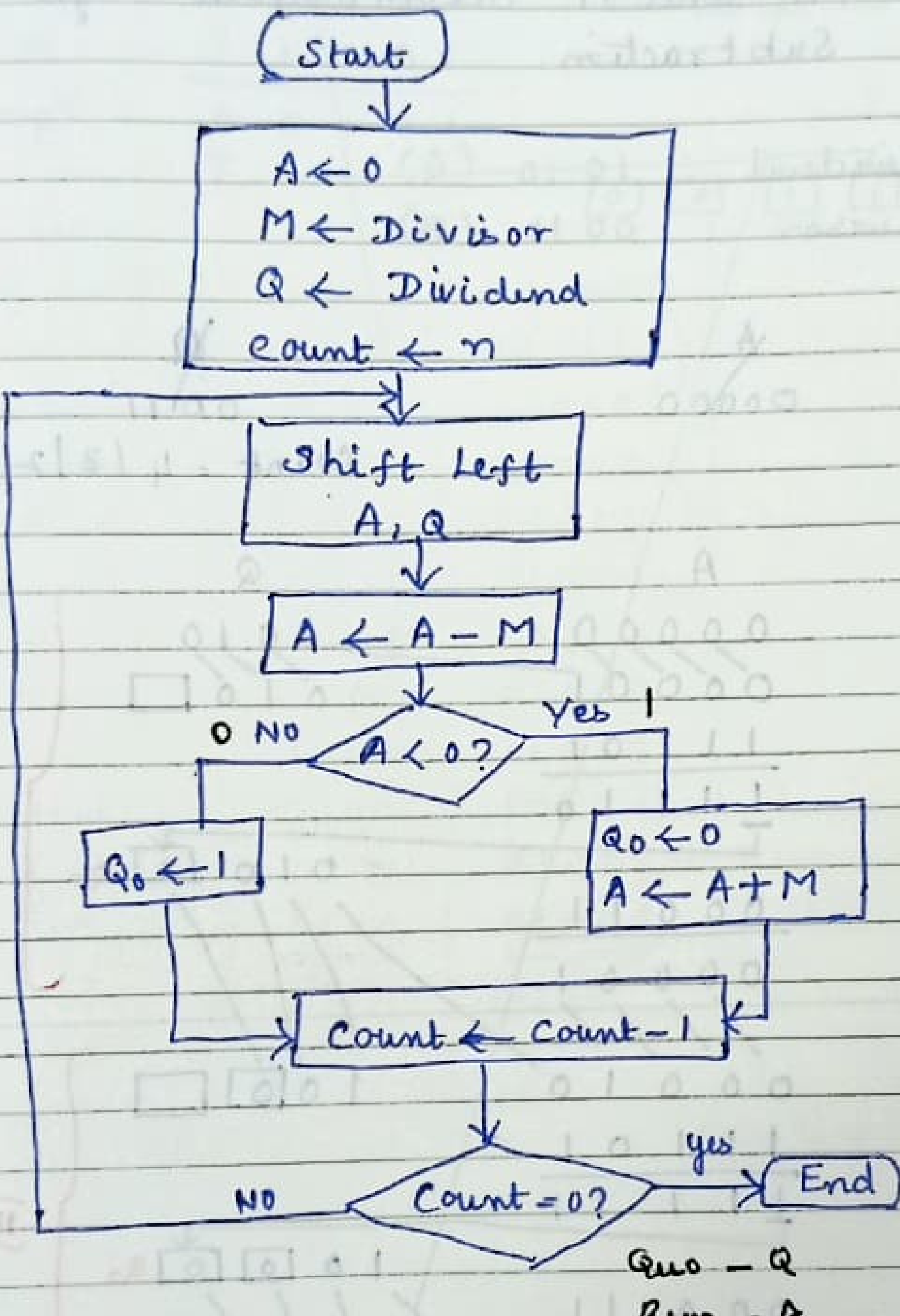
```
2 | 274
2 | 137 - 0
2 | 68 - 1
2 | 34 - 0
2 | 17 - 0
2 | 8 - 1
2 | 4 - 0
2 | 2 - 0
    1 - 0
```

100010010

```
             10101
1101 | 100010010
       1101
       -----
        10000
         1101
         -----
          1110
          1101
          ----
             1
```

```
            1101
1011 | 10010011
       1011
       ------
       0011100
          1011
          -------
          001111
             1011
             ----
              100
```

Restoring Division



n bit +ve divisor loaded − reg M
n bit +ve dividend loaded − reg Q

Reg A is set to 0

After division operation − n bit quotient reg Q
remainder − reg A

extra bit position at the left end of both A and M accommodates sign bit during subtraction.

Eg: dividend : 1010 (Q)
divisor : 0011 (M)

A: 00000 M: 00011

count = 4 | 3 | 2 | 1

00011
11101

	A	Q	
Initial	00000	1010	I
Shift Left	00001	010$\square$	
A ← A − M (2's Comp)	11101		
	11110		
		010$\boxed{0}$ Q_0	
A ← A + M	00011		
	00001		
Shift Left	00010	10$\boxed{0}$$\square$	II
A ← A − M (2's Comp)	11101		
	11111		
		10$\boxed{0}$$\boxed{0}$ Q_0	
A ← A + M	00011		
	00010		
Shift Left	00101	0$\boxed{0}$$\boxed{0}$$\square$	III
A ← A − M (2's comp	11101		
	00010		
		$\boxed{0}$$\boxed{0}$$\boxed{0}$$\boxed{1}$ Q_0	

00100 0001

Shift left   0 0 1 0 0 $\boxed{0}\ \boxed{0}\ \boxed{1}\ \boxed{\ }$

$A \leftarrow A - M$ 1 1 1 0 1

(2's comp)   0 0 0 0 1

$\boxed{0}\ \boxed{0}\ \boxed{1}\ \boxed{1}$ q_0

2)11
9
2

011 | 1011
011
101
011
10 (2)

Eg: 11 (Dividend) / 3 (Divisor) ⟹ 3 (Quo) & 2 (Rem)

$-M = 11100$
$+1$ 2's comp
11101

Tracing Table

n	M	A	Q	Action / operation
4	00011	00000	1011	Initialization
		00001	011□	Shift Left AQ
		11110	011□	A = A − M
3		00001	0110	$Q_0 = 0$ Restore A
		00010	110□	Shift Left AQ
		11111	110□	A = A − M
2		00010	1100	$Q_0 = 0$ Restore A
		00101	100□	Shift Left AQ
		00010	100□	A = A − M
1		00010	1001	$Q_0 = 1$
		00101	001□	Shift Left AQ
		00010	001□	A = A − M
0		$\underbrace{00010}_{A(2)}$	$\underbrace{0011}_{Q(3)}$	$Q_0 = 1$

For M and A number of bits will be n+1

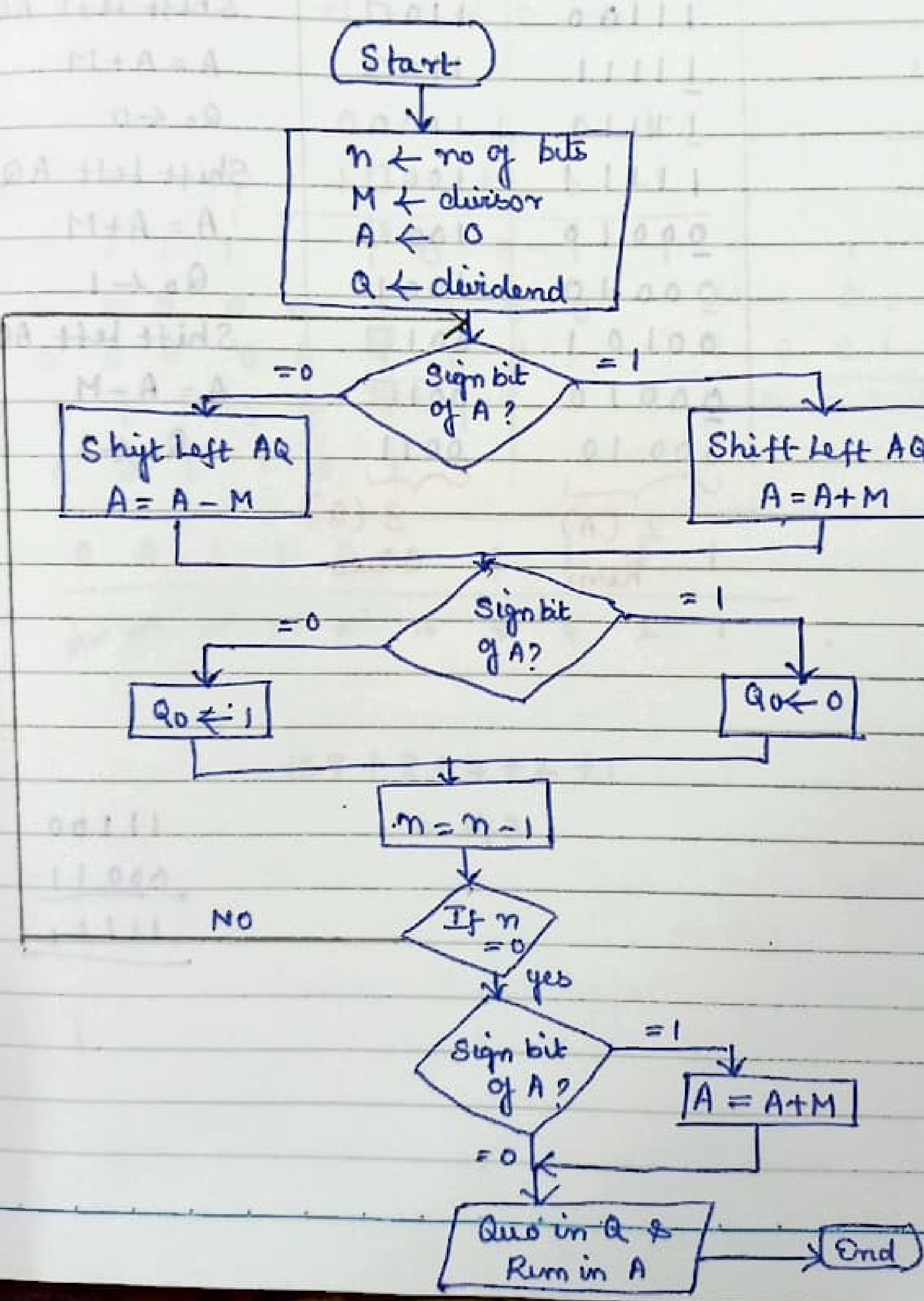
NO. DATE

Non-Restoring Division

n bit +ve divisor - reg M
n bit +ve dividend - reg Q

Reg A is set to 0

After division operation is complete
n bit Quo - reg M, n bit rem - reg A



No.

DATE

Eg: 11 (Dividend) / 3 (Divisor) ⇒ 3 (Quo) & 2 (Rem)

Tracing Table $\qquad -M = 11101$

n	M	A	Q	Action / operation
4	00011	00000	1011	Initialization
		00001	011□	Shift Left AQ
		11110	011□	A = A − M
3		11110	0110	$Q_0 \leftarrow 0$
		11100	110□	Shift Left AQ
		11111	110□	A = A + M
2		11111	1100	$Q_0 \leftarrow 0$
		11111	100□	Shift Left AQ
		00010	100□	A = A + M
1		00010	1001	$Q_0 \leftarrow 1$
		00101	001□	Shift Left AQ
		00010	001□	A = A − M
0		00010	0011	$Q_0 \leftarrow 1$

2 (A) Rem $\qquad$ 3 (Q) Quo

$$\begin{array}{r} 11100 \\ 00011 \\ \hline 11111 \end{array}$$

Floating Point numbers and operations

Sign bit
-ve 1
+ve 0

Booth Multiplication

Eg: 0 1 1 0 1 (+13) X 0 1 0 1 1 (+11)

0 1 0 1 1 [0

+1 -1 +1 0 -1

1 0 0 1 0
+1
1 0 0 1 1

					0	1	1	0	1
					+1	-1	+1	0	-1
1	1	1	1	1	1	0	0	1	1
0	0	0	0	0	0	0	0	0	
0	0	0	0	1	1	0	1		
1	1	1	0	0	1	1			
0	0	1	1	0	1				
0	0	1	0	0	0	1	1	1	1
512	256	128	64	32	16	8	4	2	1

$128 + 8 + 4 + 2 + 1$
$= 143$

Eg: 100 11 (-13) x 0 1 0 11 (11)

0 1 0 1 1 [0

+1 -1 +1 0 -1

```
 0 1 1 0 0
        +1
 ---------
 0 1 1 0 1
```

```
                1   0   0   1   1
               +1  -1  +1   0  -1
          -----------------------
0  0  0  0  0   0   1   1   0   1
0  0  0  0  0   0   0   0   0
1  1  1  1  0   0   1   1
0  0  0  1  1   0   1
1  1  0  0  1   1
-----------------------------------
1  1  0  1  1   1   0   0   0   1
```

= -143

Floating point number Representation (FPR)

$\pm$ Significant $\times$ Base$^{\pm \text{Exponent}}$

$0.00000000005 = 0.5 \times 10^{-10}$
$50000000000 = 5 \times 10^{10}$

Normalization rules of FP number

1. The integer part should be zero.
2. $0.d_1 d_2 \ldots d_n \times B^{\pm E}$ then $d_1 > 0$ and all $d_i \geq 0$, $i=2$.

$0.123 \times 10^4 = 0.0123 \times 10^5 = 1.23 \times 10^3$

Two representation techniques

1. Single precision (32 bit)
2. Double precision (64 bit)

C prg Float data type - 4 bytes
$4 \times 8 = 32$ bits
double data type - 8 bytes
$8 \times 8 = 64$ bits

Signifiant also called mantissa

Single precision

←— 32 bits —→

1	8	23 bits

23 bits → Truncated Significant

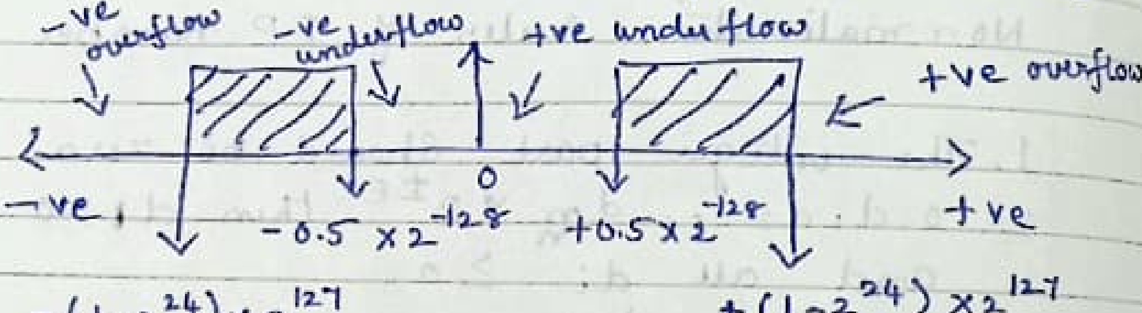
1 → Sign of the Significant

$0 \rightarrow +ve$

$1 \rightarrow -ve$

8 → Biased Exponent
128 will be added with the exponent

So range -128 to 127 will be shifted to 0 to 255



Double precision

←— 64 bits —→

1	11	52 bits

$$\begin{array}{r} 1.1100 \\ 0.0110 \\ \hline 10.0010 \end{array}$$

Floating point Arithmetic - Add & Subtract

Steps to add / subtract two floating point numbers

1. Compare the magnitudes of the two exponents and make suitable alignment to the number with the smaller magnitude exponent.

2. perform the Addition / Subtraction

3. Perform normalization by shifting the resulting mantissa and adjusting the resulting exponent.

Eg1: Add 1.1100×2^4 and 1.1000×2^2

1. Alignment: 1.1000×2^2 has to be aligned to 0.0110×2^4

2. Addition: Add two numbers to get 10.0010×2^4

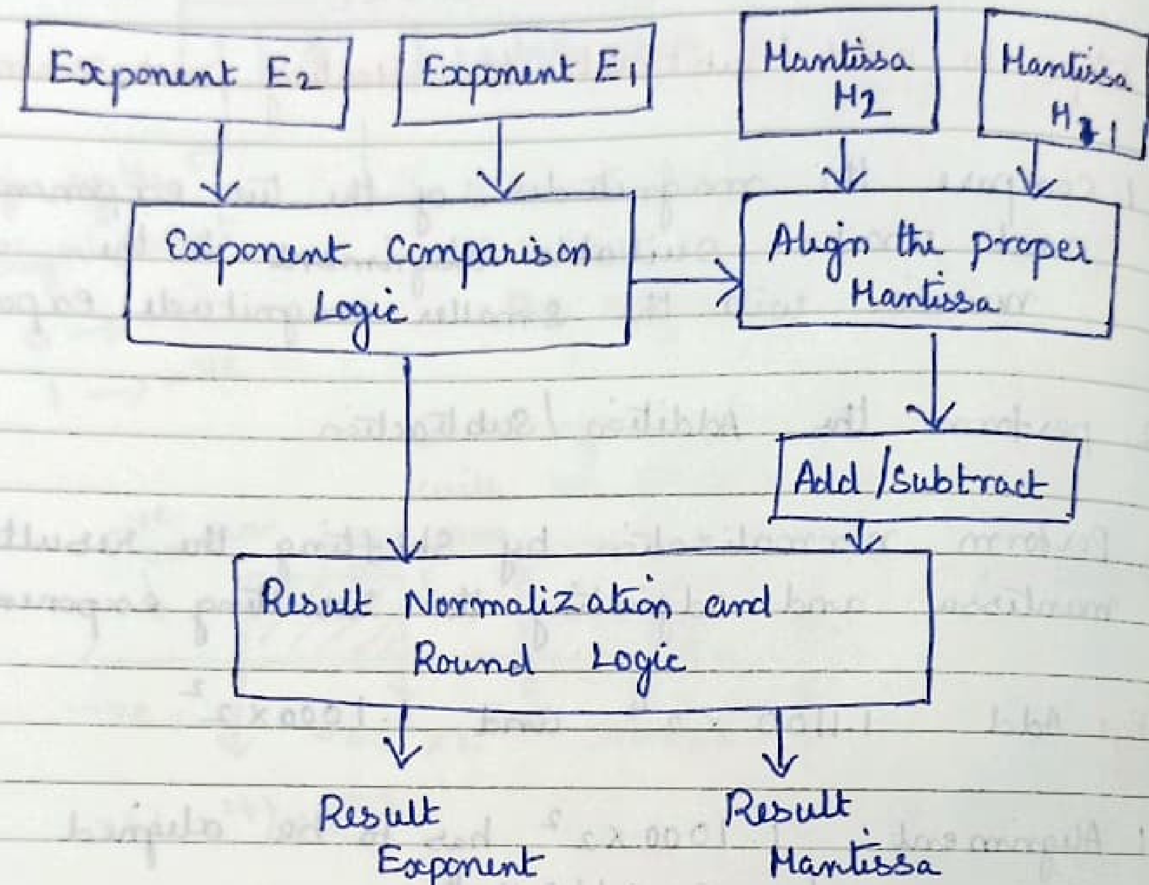
Add
$$\begin{array}{r} \cancel{1.1000} \\ \cancel{0.0110} \\ \hline \cancel{10} \end{array} + \qquad \begin{array}{r} 1.1100 \\ 0.0110 \\ \hline 10.0010 \\ \hline \end{array} +$$

3. Normalization: Final normalized result 0.1000×2^6 (Assume 4 bits are allowed after radix point)

No.

DATE

Addition / Subtraction of F.P numbers



Eg 2: Add -9.999×10^{1} and 1.610×10^{-1}

1. Alignment: Shift the number with smaller exponent
   1.610×10^{-1} as 0.016×10^{1}

2. Addition

$$\cancel{\begin{array}{r} -9.999 \\ +\ 0.016 \\ \hline 10.015 \end{array}} \times 10^{1} \qquad \begin{array}{r} -9.999 \\ +\ 0.016 \\ \hline -9.983 \end{array}$$

NO.

DATE

3. Normalization :

$$\cancel{0.1001 \times 10^3} - 0.9983 \times 10^2$$

Represent a number in IEEE 754 32 bit floating point number notation

263.3

Convert 263 into binary representation

2 | 263	
2 | 131	− 1
2 | 65	− 1
2 | 32	− 1
2 | 16	− 0
2 | 8	− 0
2 | 4	− 0
2 | 2	− 0
1	− 0

263 : 100000111

0.3 × 2	0.6	0
0.6 × 2	1.2	1
0.2 × 2	0.4	0
0.4 × 2	0.8	0
0.8 × 2	1.6	1
0.6 × 2	1.2	1
0.2 × 2	0.4	0
0.4 × 2	0.8	0
0.8 × 2	1.6	1
0.6 × 2	1.2	1
		0
		0
		1
		1
		⋮

1. 263.3 = 100000111.0100110011 ...

2. Represent the binary form in scientific notation

Shift the decimal point to the left.
for Each to the left need to multiply the number by 2.

Scientific Notation:

$\underbrace{1.0000011101001100 11}_{\text{mantissa}} \ldots \times 2^{8}$

[3] write it in IEEE 754 format

The format Suggest - 1st bit Should be the Sign bit

Single precision (32 bit)

-ve - 1
+ve - 0

$\underbrace{X}_{\text{1 Sign bit}}$ $\underbrace{XXXX\ XXXX}_{\text{8 exponent bits}}$ $\underbrace{XXXX \ldots XX}_{\text{23 fraction bits}}$

0

exponent bias for Single precision 127

127+8 = 135

10000111

combine four bits together and write

combine all bits together

00000 1110100110011...

263.3 ⟹ 0100 0011 1000 0011 1010 0110 0110 0110...

Final representation of 32 bits

2. Represent a number 1259.125 in IEEE 754 32 bit floating point number notation

1. convert decimal to binary

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

1259: 10011101011

0.125 x 2 =	0.25	0
0.25 x 2	0.5	0
0.5 x 2	1.0	1
0 x 2	0	0
		⋮

0.125 : 0010....

1259.125 = 10011101011.0010...

2. Represent the binary form in Scientific notation

Single precision : $(1.N)\ 2^{E-127}$

Double Precision : $(1.N)\ 2^{E-1023}$

$1.00111010110010 \ldots \times 2^{10}$

3. write it in IEEE 754 format

$\underbrace{x}_{\text{1 sign bit}}$ $\underbrace{xxxxxxxx}_{\text{8 exponent bits}}$ $\underbrace{xxxxxx\ldots xx}_{\text{23 fraction bits}}$

$1.00111010110010 \ldots \times 2^{10}$

1. Number consider is a +ve number so
Sign bit is <u>0</u> (Sign bit)

2. Exponent bias for single precision is
127 and Exponent value is 10
So $127 + 10 = 137$
Convert 137 to binary
<u>1000 1001</u> (Exponent)

3. 23 fraction bits
<u>00111010110010000000000</u> (23 fraction bits)

x	xxxxxxxx	xxxx....xx
0	10001001	00111010110010....

Combine four bits together and write
0100 0100 1001 1101 0110 0100 0000 0000

over all 32 bit
representation

No.

DATE

Microprogrammed ctrl org

computer contains – CPU, Io devices, memory

CPU → ALU, Reg, control unit

control unit → resp for generating ctrl signals

data will be transferred bt reg

instructs ALU to perform operation

control unit design
- Hardwired – H/w comp flip flops, decoder, encoder, finite State m/c
- Microprogrammed → perform only read op not write

control Mem (ROM) stores microprogram

Micro prg is the collection of micro instr

Each micro instr contains one/more micro operations

once the cu is designed – no possibility to modify the cu.

CAR (control Address reg) – address of the microinstr

CDR (control Data reg) – instr that is to be executed

It contains extra bits inorder to generate the next address info. → status bits – zero, sign, overflow } bits

micro instr contains the collection of ctrl words

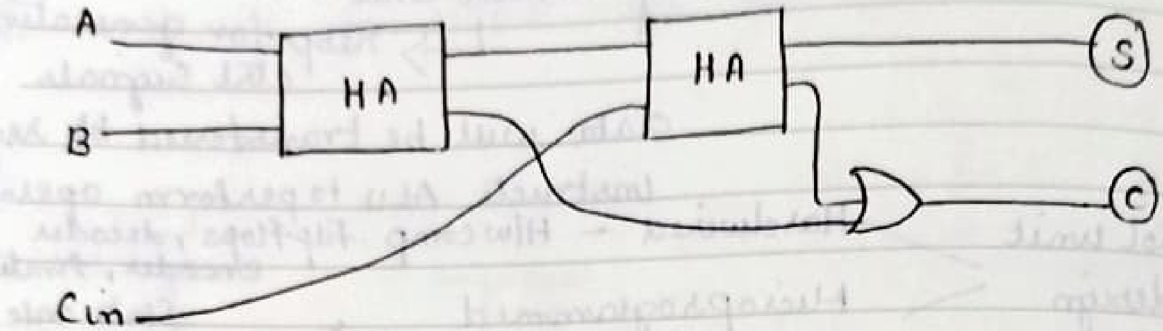
CDR also called as pipeline reg – executing multiple tasks simul in less amt of time

```
  1111010   00000010        00000010   1000
  0000101   11111110
 ___________________
  0000101   111111110       512 + 128 + 64 + 32
                                + 16 + 8 + 4 + 2
 512 256 128 64 32 16 8 4 4 2 1               + 1
                                          = 717
```

No.

DATE

Full Adder



input			output	
A	B	Cin	S	C
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

20

5×4 = 20

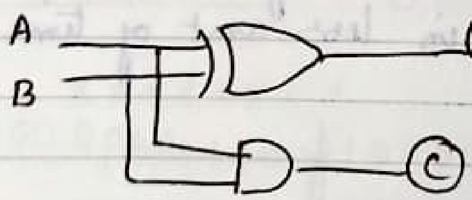
18

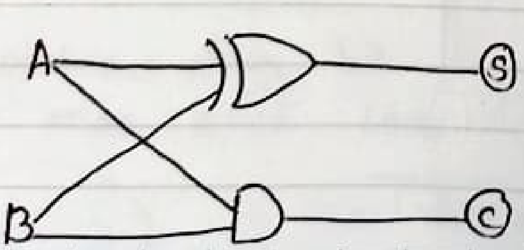
2×9 =

21

3

Half Adder





input		output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Addressing Modes

```
Data Segment
  Data1 db 23h
  Data2 dw 1234h
  Data3 db 00h
  Data4 dw 0000h
  Data5 dw 2345h, 6789h
Data Ends
Code Segment
  Assume cs:code, DS:data
  Start:
    Mov ax, data
    Mov ds, ax

    Mov AL, 25h          } immediate mode
    Mov ax, 2354h        }

    Mov Bx, Ax           } reg mode
    Mov CL, AL           }

    Mov al, data1        }
    Mov ax, data2        } direct mode /
                         } absolute mode
    Mov data3, AL        }
    Mov data4, ax        }

    Mov bx, offset data5 } Register indirect
    Mov ax, [bx]         }

    Mov DI, 02h          } indexed mode
    Mov ax, [bx+DI]      }
Code ends
end Start
```

611
612
638 620
637 623
640 625
646 628
650 629
652 632
654 633
655
658 674
672

00001
11101
11110

01101 0

00010
11101
11111