module - 3

Fixed Point & Floating Point formats

Multiplication :-

* Manual multiplication requires the 2 - Actions :
  i) generation of Partial products
  ii) summation of Partial Products.
* multiplication process is controlled by the multiplier so, partial products are generated based on the multiplier bits that is :
  - when multiplier bit is '1' then partial product is multiplicand otherwise '0'
  - After the generation of partial products summation operation is provided to produce the final product

eg: multiplicand * multiplier

```
   1111   *  1111
  ------
   1111
  1111
 1111
1111
--------
11100001
--------
```

Partial Products

4 → 100 → carry (2)
6 → 110 → carry (3)
5 → 101 → carry (2)

n bit * n bit = 2n bit

Que:- Consider the following multiplication process and identify the value of w, y & z variables:

$(1\,0\,w\,1\,z)_2 * (15)_{10} = (y\,0\,1\,0\,1\,1\,0\,0\,1)_2$

$(16 + 4w + 2 + 1z) * 15 = 256y + 64 + 16 + 8 + 1$

$(18 + 4w + 2) * 15 = 89 + 256y$

$270 + 60w + 15z = 89 + 256y$

$1\,0\,w\,1\,z * 1111$

$1\,0\,w\,1\,z$

$1\,0\,w\,1\,z$

$1\,0\,w\,1\,z$

$1\,0\,w\,1\,z$

$1\,0\,1\,0\,1\,1\,0\,0\,z$

$y = 1 \quad w = 1$

$1 + w + 1 + 1 = 0$

If $w = 0$

$1 + 0 + 1 + 1 = 3\,(11)$

If $w = 1$

$1 + 1 + 1 + 1 = 4 = 100$

$(y\,0\,1\,0\,1\,1\,0\,0\,1)$

$z = 1$

★ Note:-

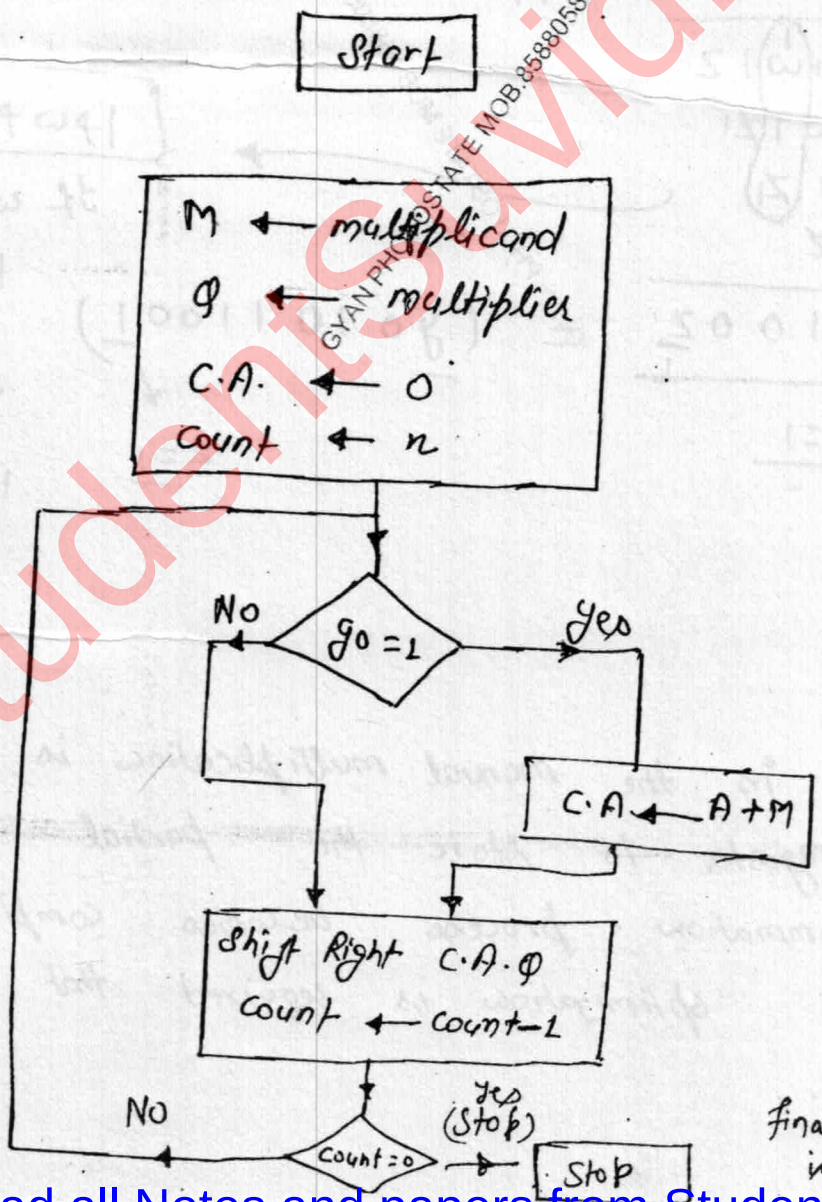
Limitation in the manual multiplication is, It requires more register to store the partial products and summation process becomes complex in the H/w so optimization is required that is Accumulated addition.

unsigned multiplication :-

* when the multiplication takes place between 2-n bit number then Result will be expressed in a 2n bit long. therefore Register pair is required to store the result.

n bit * n bit = 2n bit

- Optimized multiplication process is described in the following flow-chart :-



Final Product is in AQ Reg.

eg: 11 * 13 ⇒ 1011 * 1101

M = 1011 , Count = 4

C	A	Q	
0	0000	1101	; initial
0 → 0	1011	1101	; C, A ← A+M (0000 + 1011)
0 → 0	0101	1110	; Shift (lost = q_0) Count = 3
0	0010	1111	; Shift Count = 2
0 → 0	1101	1111	; C, A ← A+M (0010 + 1011)
0	0110	1111	; Shift Count = 1
0 → 1	0001	1111	; C, A ← A+M (0110 + 1011)
0	1000	1111	; Shift Count = 0

Final Product is in A Q Reg Pair:-

i.e. 128 8421

1000 1111 [143]

*

multiplier	Operation
0	Shift
1	Add & Shift

Date 31/08/2017

Signed multiplication (Booth's multiplication)
or
Bit-Pair multiplication :-

:- Signed multiplication is controlled by the multiplier bit-pairs. In the bit-pair formation one assumption is required to get the initial bit-pair i.e. $\{q_{-1} = 0\}$

eg:- multiplier :- $q_3\ q_2\ q_1\ q_0$

Bit-pairs:

multiplier bit-pair	pair with $q_{-1}(0)$
q_0	$-q_0$
q_1	$-q_1$
q_2	$-q_2$
q_3	

:- Different operations are assigned to a bit-pair combination to perform the multiplication i.e.

Bit pair	operation	Recoding
00	ASR	0
01	+ & ASR	+1
10	− & ASR	−1
11	ASR	0

Note:- Arithmatic shift Right

Que :- Consider the following Booth's multiplication process :

multiplicand : 1010 1101 1011

multiplier : 1001 1010 1001

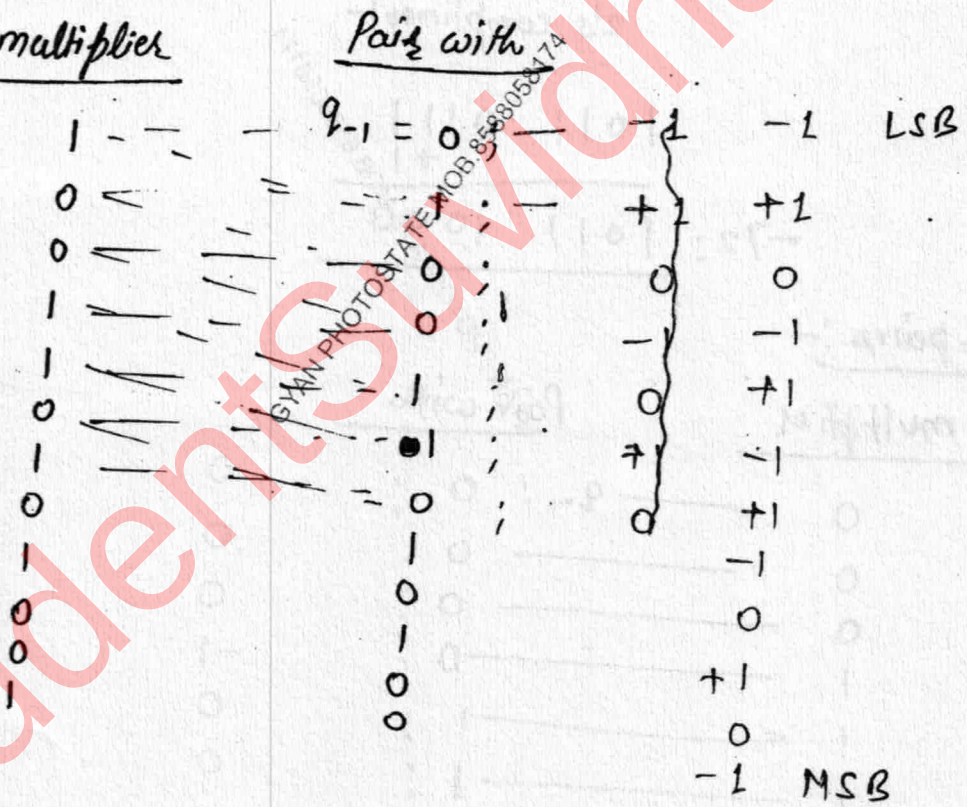
a) How many Arithmatic operations are required in the multiplication

b) what is the Recoded multiplier

Soln :-

Bit-pairs :-

multiplier



Recoded multiplier :

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

9 Arithmatic

Que:- Consider the following 8 bit multiplication

Process :

$(-121) * (-72)$

what is the Recoded multiplier :

Sol?:- multiplier :-

$-72 \longrightarrow$ 8 bit code

72 : 0100 1000

↓

2's compliment

1011 0111

+1

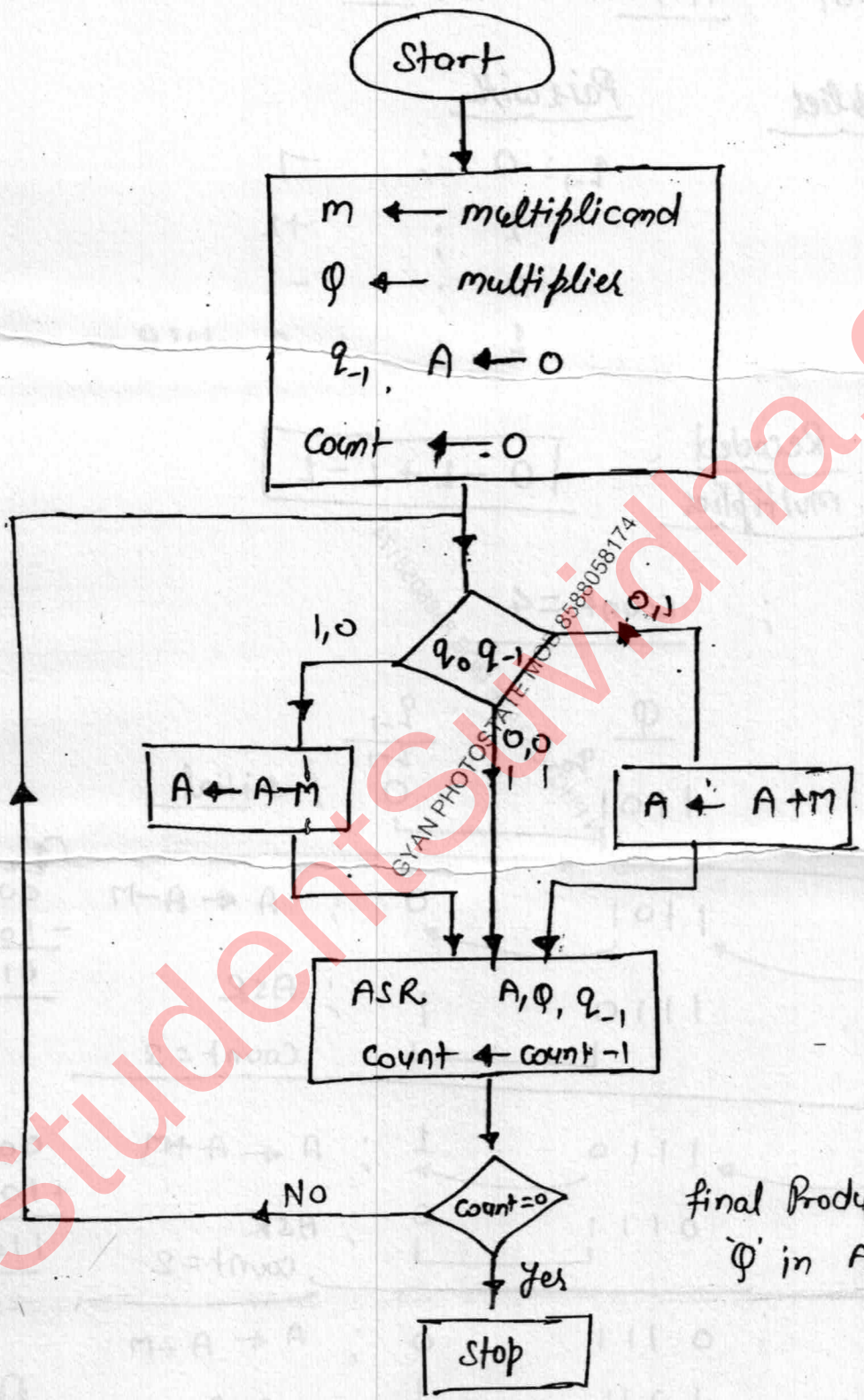
-72: 1011 1000

Bit-pairs :-

multiplier	Pair with	
0	a_{-1} : 0	0
0	0	0
0	0	0
1	0	−1
1	1	0
1	1	0
0	1	+1
1	0	−1

Recoded multiplier :- $\boxed{-1+100-1000}$

:- optimized Signed multiplication process is described in the flow-chart :-



final Product 'Q' in AQ Register Pair

eg: (-7) * (-3) = +21

↓ ↓

Q → 1001 '1101 ← multiplier

multiplier	Pair with	
q_0: 1	q_{-1}: 0 ;	-1
0	1 ;	+1
1	0 ;	-1
1	1 ;	0 MSB

Recoded multiplier : | 0 -1 +1 -1 |

M : 1001 ; Count = 4

A	Q	q_{-1}	
0000	1101	0	; Initial
0111	1101	0	; A ← A-M
0011	1110	1	; ASR
			Count = 3
1100	1110	1	; A ← A+M
1110	0111	0	; ASR
			Count = 2
0101	0111	0	; A ← A-M
0010	1011	1	; lost ASR
			Count = 1
0001	0101	1	; ASR
			Count 0

0000 - 1001 = 0111

0011 + 1001 = 1100

1110 + 1001 = 0101

2's compliment

"Floating Point Data format":-

:- Representation of very large & very small data consumes more memory space.

Eg: +978 000 000 0000 ; very large.

+0.000000 000 00 978 ; very small

★ To Represent the large & small fraction of the data within a less amount of m/y space, floating point format is used.

:- General Form of a floating point of data is:

$$\pm m * B^{\pm e}$$

± ; Sign

m ; mantissa / significant (fraction)

B ; Base / Radix

e ; exponent.

To Represent the large and small fraction of a data in a floating point format, common format is used that is:

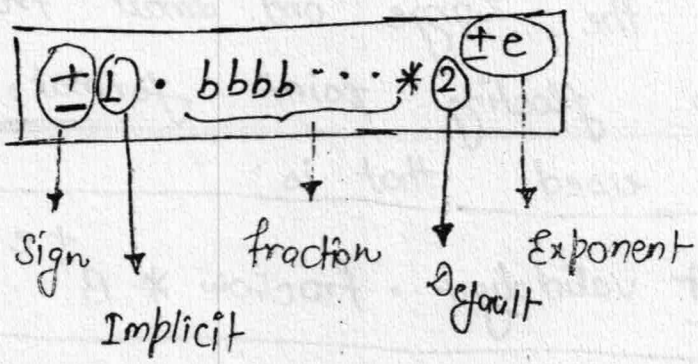
$$\pm \text{valid digit} . \text{fraction} * B^{\pm e}$$

: when the Base of a data is greater than '2' then valid digit will be varies so floating point data structure is changed to store the valid digit. i.e

Base	valid digit
B = 3	{1, 2}
B = 4	{1, 2, 3}
B = 5	{1, 2, 3, 4}
⋮	⋮

★ To store the Data without change the Structure, make sure that Base is always restricted to '2'. therefore valid digit become implicit i.e '1'. Hence this Bit is need not be Stored in the m/y.

★ General form of a Normalization (common format) is :



★ In this format 'Mantissa Alignment process' is used to adjust the decimal point.

:- Right Alignment : Right Alignment increament the Exponent

:- Left Alignment : Left Alignment decreaments the Exponent.

→ Mantissa Alignment :

eg:- -1101011.11011

↓

mantissa align to Right upto 6 times

↓

$$-1.101011 * 2^{+6}$$

eg: -0.0000000000101111

↓

mantissa align to left 12 times

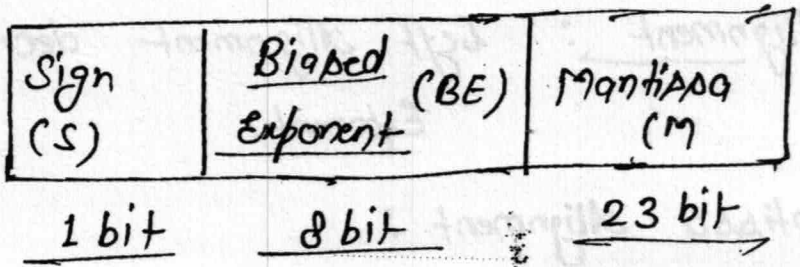
↓

$$-1.01111 * 2^{-12}$$

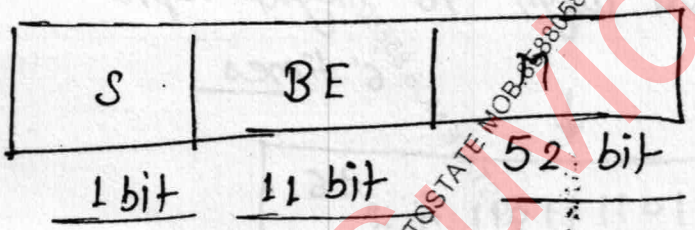
★ when the Data is present in the Normal format then it is stored in the m/y by uping the floating point format.

According to IEEE-754 standard 2 kinds of formats are present.

1) "Single Precision" (32 bit format) :-



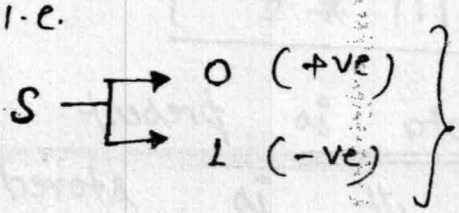
2.) Double Precision (64 bit format) :-



* Floating Point data always stored in the memory with 3-fields of information :

1.) Sign field :-

This field is use to indicate the sign of a data. i.e.



2.) Biased Exponent field :-

This feild is used to represent the Exponent value in the data.

* Exponent is always stored in the m/y in a Biased format i.e.

* Biased Exponent (BE) = AE + Biap

Biap is a maximum possible positive exponent, depends on the format i.e.

BE field size	Range of exponent	Biap
6 bit	$\{-(2^{6-1}) \text{ to } +(2^{6-1}-1)\}$ $\{-32 \text{ to } +31\}$	+31
9 bit	$\{-256 \text{ to } +255\}$	(+255)
n bit	$\{-(2^{n-1}) \text{ to } +(2^{n-1}-1)\}$	$+(2^{n-1}-1)$

* Biap concept is used to preserve the sign of a Exponent : i.e.

{ when (BE > Biap) then AE is +ve
when (BE < Biap) then AE is -ve }

Note :-

To Retrieve the exponent from the m/y, we need to subtract the Biap from the Biase Exponent

to get the Actual exponent.

i.e. $\boxed{AE = BE - Biap}$.

Eg:- format:

S	BE	M
1bit	6bit	13 bit

Data: $+1.10101 * 2^{+9}$

① Store: $\boxed{BE = AE + Biap}$

$\therefore Biap = +(2^{n-1}-1)$

$= +(2^{6-1}-1)$

$= +31$

AE: 001001

Biap: 011111

101000

② Retrieve:- $\boxed{AE = BE - Biap}$

BE: 101000

011111

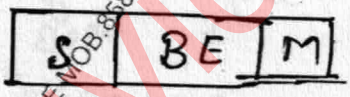
001001 [+9]

* when the format is designed with a "Excess Biap". then Biap is the centre of the Exponent Range i.e.

$$\boxed{\text{Biap} = \frac{2^n}{2}} \Rightarrow \{ n: \text{BE field size} \}$$

* In this format we need to take the compliment of a Biased Exponent to get the Actual Exponent rather than the subtraction operation.

Eg:- format:



1 bit 6 bit 13 bit

↳ Excess Biap

Data: $+1.1010111 * 2^{+9}$

① Store:- BE = Excess Biap

$= \left(\frac{2^n}{2}\right)$

$= \frac{2^6}{2} = \boxed{32}$

AE: 001001

Biap: 100000

101001

② Retrieve :-

$$AE = BE - Bias$$

BE : 10100

Bias : 100000

AE : 00100 [+9]

Alternative :- AE = compliment of a ~~Bia~~ MSB of a "BE"

BE : 10100

↓

compliment of MSB

↓

AE : 00100

= [+9]

3.) Mantissa field :-

:- This field is used to represent the fraction part of a data.

:- Mantissa is always stored in the memory in a normalized format i.e.

$$\boxed{1.bbbb\ldots.} \quad b : 0/1$$

(1) — implicit

bbbb.... — fraction

:- In the above format only the fraction part is stored in the mantissa field because

So, valid digit become Implicit so need not be stored in the memory.

Note :-

To Retrieve the data from the memory 'De-normalization' (Sub-Normal) Concept is used to adjust the result i.e.

DE- Normalization :-
(Sub Normal)

From the memory

$\pm .bbbb \ldots * 2^{\pm e}$

"mantissa align to right upto 1 time to get the implicit ~~stored~~ valid digit as a fraction"

$\pm 0.1bbbb \ldots * 2^{\pm e+1}$

$\boxed{\pm 0.1bbbb \ldots * 2^{\pm e'}}$

eg:- format :

S	BE	M

1 bit 6 bit 13 bit

↳ Excess-Biap

Data: $(+)1.10|0111 * 2^{(+9)}$

AE : 00|001

Biap: 100000

BE : 10|001

Store:-

0	10	0 111	10	001

↓

Read the Data

↓

S: 0 (+ve)

AE : BE − Biap

(+9)

M : 10|111

↓

$+.10|0111 * 2^{+9}$

↓

Mantissa align right upto 1 time

↓

$+ 0.(1)0|0111 * 2^{+9+1}$

↓

implicit

$+ 0.11|0|0111 * 2^{+10}$

Que:- Consider a Hypothetical format used to represent :

format:

S	BE	M
1 bit	7 bit	12 bit

Data : $-13.25 * 2^{+11}$

a) what is its Hexa decimal equivalent when the Data is stored in the m/y

a) without Normalization

b) with Normalization

Soln :-

① Sign : = -ve

= 1

② Biap Exponent : = AE + Bias

Biap = Normal Biap

$= +(2^{n-1}-1)$

$= (2^{7-1}-1)$

$= 63$

AE : 0 00 1011 ← 11

Biap : 0 11 1111

100 1010

③ Mantissa :

(13.25) = $(X)_2$

↓ ↓

1101 01

X = 1101.01

a) Data without Normalization :-

$-1101.01 * 2^{1001010}$

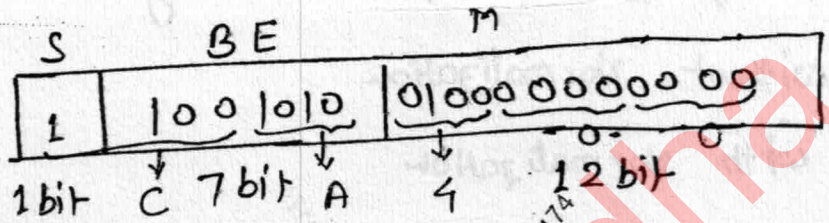
↓

Stored in m/y

↓

format required

↓



00000000001 ✗ 0's padding at MSB

01000000000 ∴ 0's padding at LSB ✓ (fraction mantissa)

b) Data with Normalization :-

$-1101.01 * 2^{1001010}$

↓

$(1.bbbb....)$

↓

mantissa align to right upto 3 times

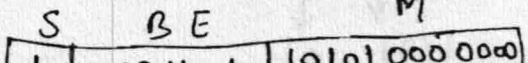
↓

$-1.10101 * 2^{1001010+3}$

$-1.10101 * 2^{1001101}$

↓

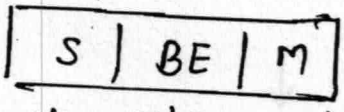
Store in m/y



→ (CDA 80)

Que:- Consider the following Hypothetical format used to represent the data:

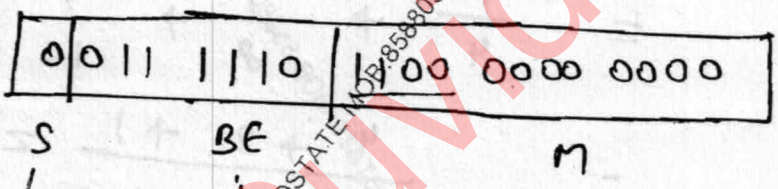
format:



Data in the m/y: 0x 3EC00

↳ Hexa decimal

a) what is the actual Data associated with above code in Decimal ?



1) Sign (S) = 0

2) AE = BE - Bias

Bias = $\frac{2^7}{2} = \frac{2^7}{2} = \frac{128}{2} = 64$

Excess Bias

0 111 1110

or

Compliment of MSB

1 111 1110

[-ve] value

[-2]

1 111 1110 → 2's → 0 000 0001 + 1 = 0 000 10

-ve

[-2]

3) Mantissa :- 1100...

Data from m/y

$+ .11 * 2^{-2}$ → De-Normalization → mantissa align to right upto 1 Digit to the left

$+ 0.111 * 2^{-1}$ ← $+ 0.(1)11 * 2^{-2+1}$

To Report the Decimal value, make the Exponent as '0' by align to right upto 1 time

$$+ 0.0111 * 2^{-1+1}$$

$$+ 0.0111 * 2^{0}$$

$$+ 0.0111$$

$$-1\text{-}2\text{-}3\text{-}4$$

$$\text{Data} = (1 * 2^{-2}) + (1 * 2^{-3}) + (1 * 2^{-4})$$

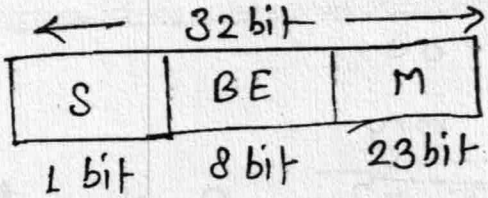
$$= \frac{1}{4} + \frac{1}{8} + \frac{1}{16}$$

$$= \frac{4 + 2 + 1}{16} = \frac{7}{16} = 0.43$$

Range of Floating Point Data :-

* Floating Point Data Range is always depends on the floating point format.

:- Let us consider Single Precision floating point format to report the Range. i.e.



1) Sign :

S → 0 (+ve)
→ 1 (-ve)

② Exponent Range :-

- Depends on the BE field
- BE = AE + Bias
- Bias = Normal Bias

$$= +(2^{n-1}-1)$$

$$= +(2^{8-1}-1)$$

$$= +127$$

a) Min BE = All 0's in BE

= 00000000

= 0

∴ AE = BE − Bias

= 0 − 127

= −127

b) Max BE = All 1's in BE

= 11111111

= 255

∴ AE = BE − Bias

= 255 − (+127)

= +128

∴ Exponent Range : {−127 to +128}

③ Mantissa Range :-

- Depends on the ~~Range~~ Normal mantissa

i.e (1.M) → mantissa

a) min m : = All 0's in 'm'

= 0000 ... (23) 0's

= 0

∴ Normal m = 1.M

= 1.0

= 1

b) max m : = All 1's in 'm'

= 111 ... (23) 1's

∴ Normal m = 1.M

= 1.111 ... (23) 1's

↓

mantissa align to left upto 23 times to convert the fraction as an integer

↓

1111 .. (24) 1's * 2^{-23}

↓

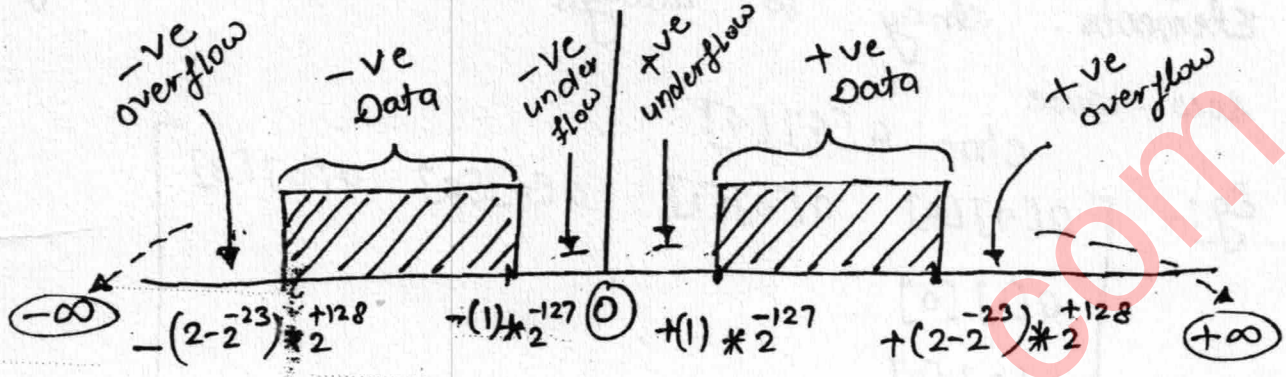
$(2^{24}-1) * 2^{-23}$

↓

$(2 - 2^{-23})$

∴ Range mantissa : { 1 to $(2-2^{-23})$ }

(4) Range of 32 bit floating point floating Point Data $\{ \pm M * B^{\pm e} \}$ is :



Round to '0'

Round to '$+\infty$'

Round to '$-\infty$'

Round to 'Nearest'

Spatial Locality in Memory System :-

- Array is a ordered set of Homogeneous data elements. Array is always stored in the m/y row-wise.

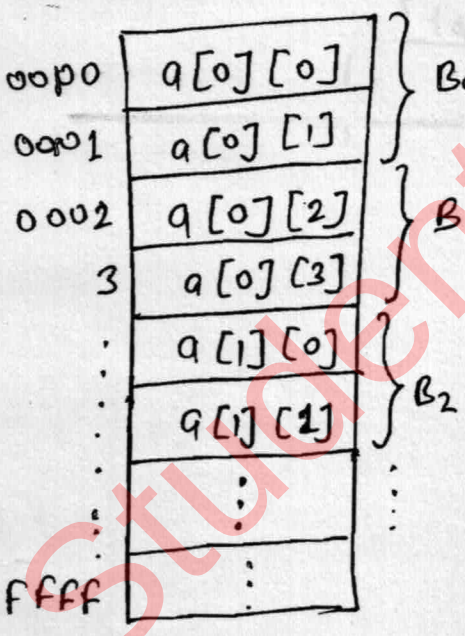
Eg :- char a[4][4]

$$\begin{bmatrix} a[0][0] & a[0][1] & a[0][2] & a[0][3] \\ a[1][0] & & & \vdots \\ a[2][0] & & & \vdots \\ a[3][0] & \cdots\cdots\cdots & & a[3][3] \end{bmatrix}$$

memory storage :-

Block size = 2B

(By default storage Row-wise)



Array elements are accessed from the memory in two-ways :-

1) Row-major Indexing :-

i.e. a[0][0], a[0][1], a[0][2], a[0][3], ...

C.M. Empty.

Accessing :-

a[0][0] : M; $\underbrace{a[0][0]\ a[0][1]}_{B_0}$ → CM

a[0][1] : Hit

a[0][2] : M; $\underbrace{a[0][2]\ a[0][3]}_{B_1}$ → CM

a[0][3] : Hit

H = 50 %

{Hit follows Miss}

2) Coloum-major indexing :-

i.e. a[0][0], a[1][0], a[2][0], a[3][0]...

CM. empty

Accessing :-

a[0][0] : M; $\underbrace{a[0][0]\ a[0][1]}_{B_0}$ → CM

a[1][0] : M; $\underbrace{a[1][0]\ a[1][1]}_{B_2}$ → CM

a[2][0] ; M ;

a[3][0] ; M :

H = 0 %

Analysis :-

Storage sequence	Accessing sequence	Spatial Locality
Row-wise	Row-wise	yes
Row-wise	Col-wise	No
Col-wise	Row-wise	No
Col-wise	Col-wise	yes

Analysis :-

- Element size = 1B
- Block size = 2B

Elements / Block = $\frac{2}{1}$ ⇒ 2

Row size = 4 Elements

Blocks / Row = $\frac{4}{2}$ = 2

or

(# misses / Row)

misses / array ⇒ # rows in array * # miss / row

⇒ 4 * 2 miss

⇒ 8 misses