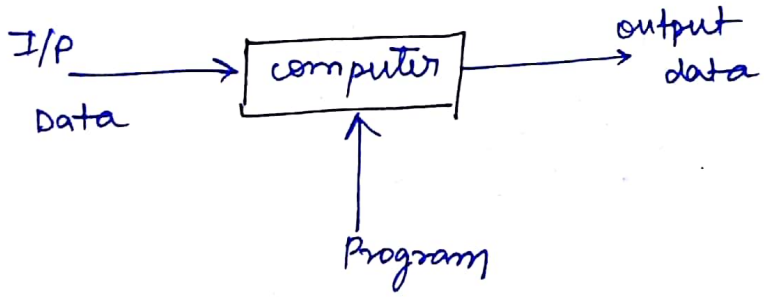
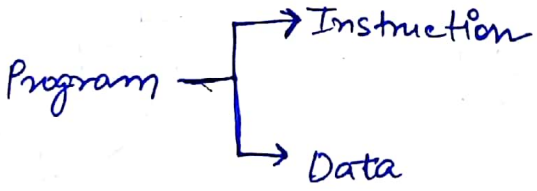
Computer Organization

Keywords :

(i) **Computer** - It is a computational machine used to process the data under the control of a program. Therefore the computer system fuctionality is program execution.



(ii) **Program :-** It is sequence of instructions along with the data.

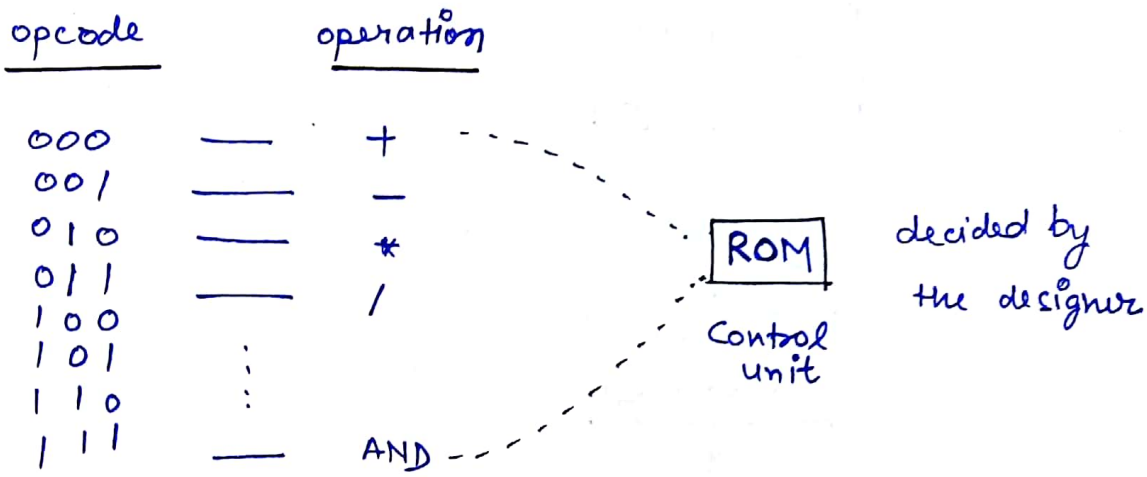


(iii) **Instruction :-** It is a binary code which is designed inside the processor to perform some taksk.

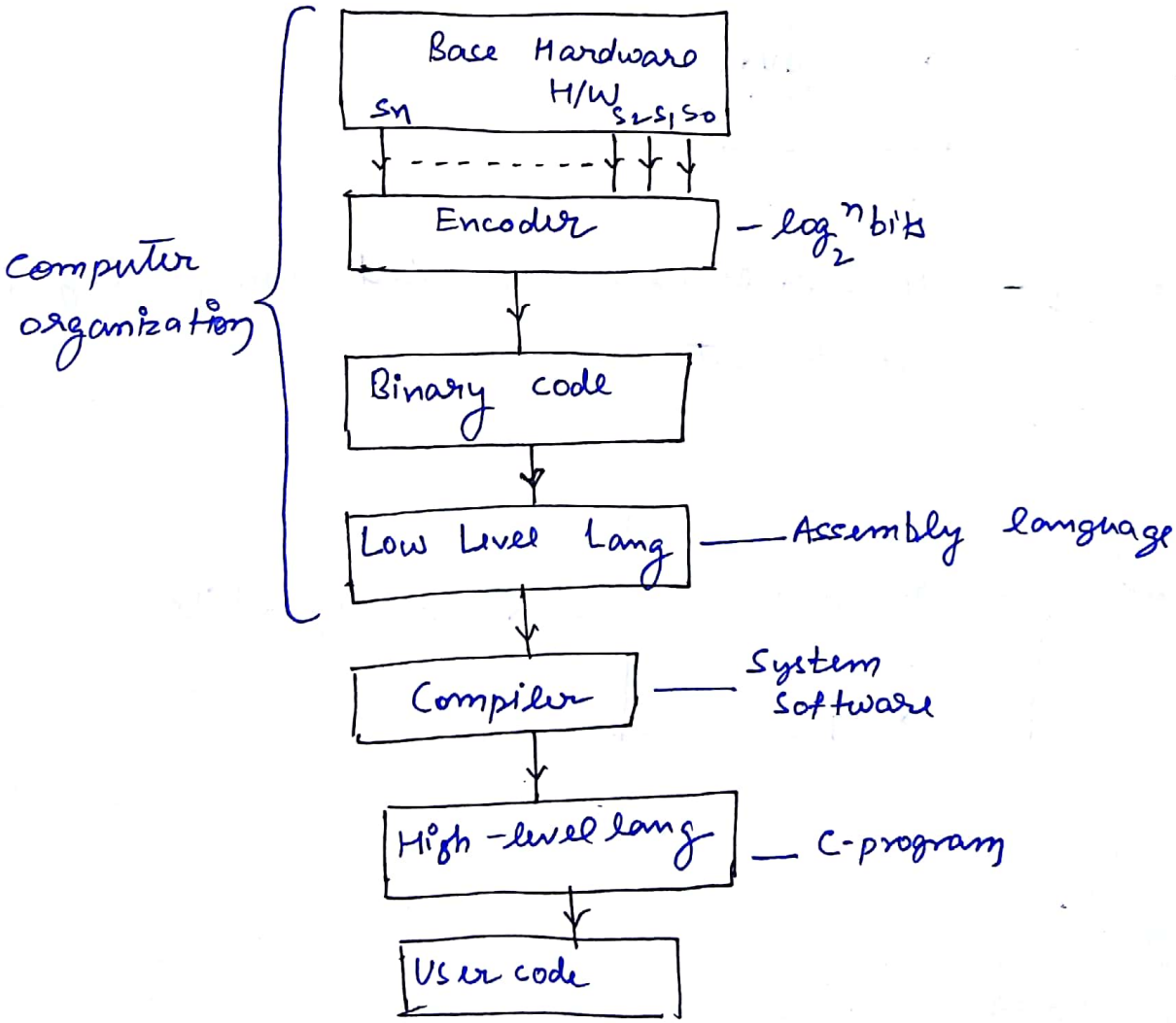
Binary code → bind with → operation

- If CPU -X supports 8 different operation then opcode

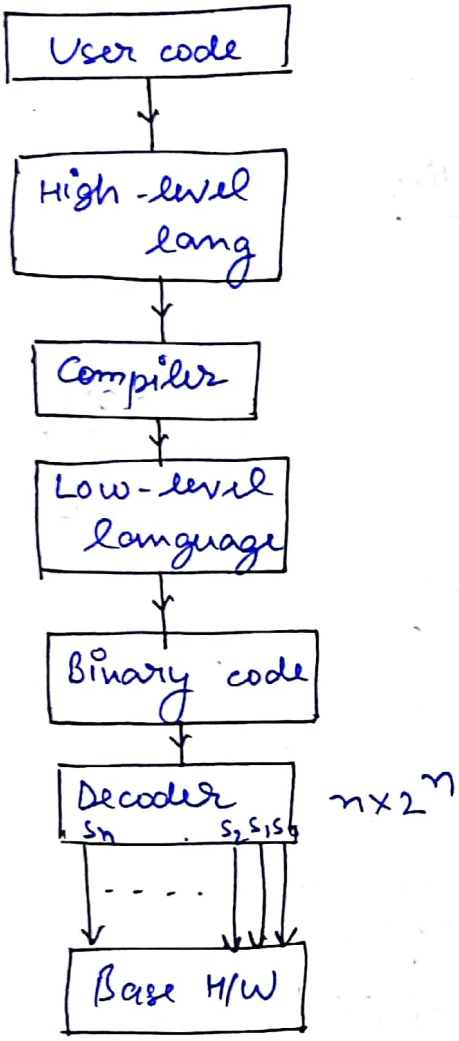
= $\log_2 8$ bits = 3 bits.



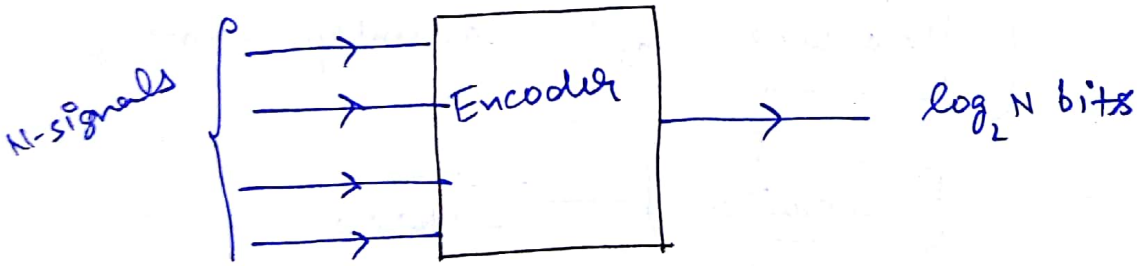
Designer View :-



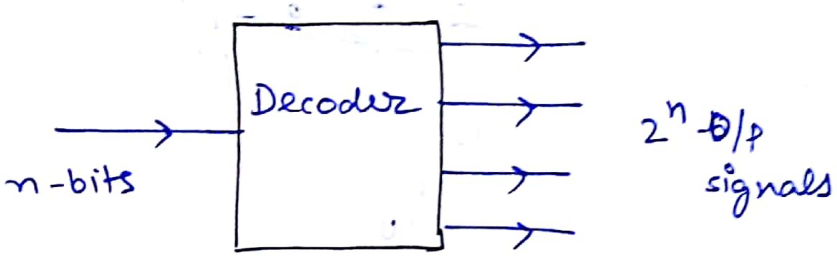
User View :-



Encoding - N signals are represented using $\log_2 N$ bit format.



Decoding - n-bit decoder produces 2^n output signals



(iv) Data :- It is a binary code which is associated with a value based on the data format.

Binary code → (binds with) → value

eg :- 0101 → 5
1101 → -5
} signed format

so, for representing data value specific format is required.

normal format

(+7) 0111
.
.
.
(+0) 0000
(-0) 1000
} data redundancy
.
.
.
(-7) 1111

FAILED

1's complement format

1000 (-7)
1's
0111

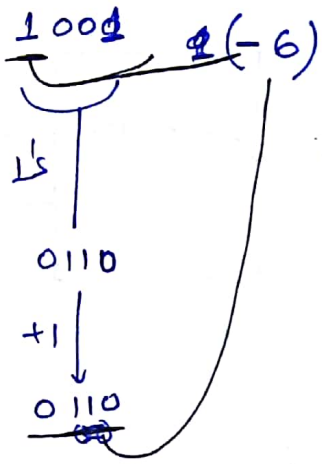
0000 (+0)
1111 (-0)
1's
0000

FAILED

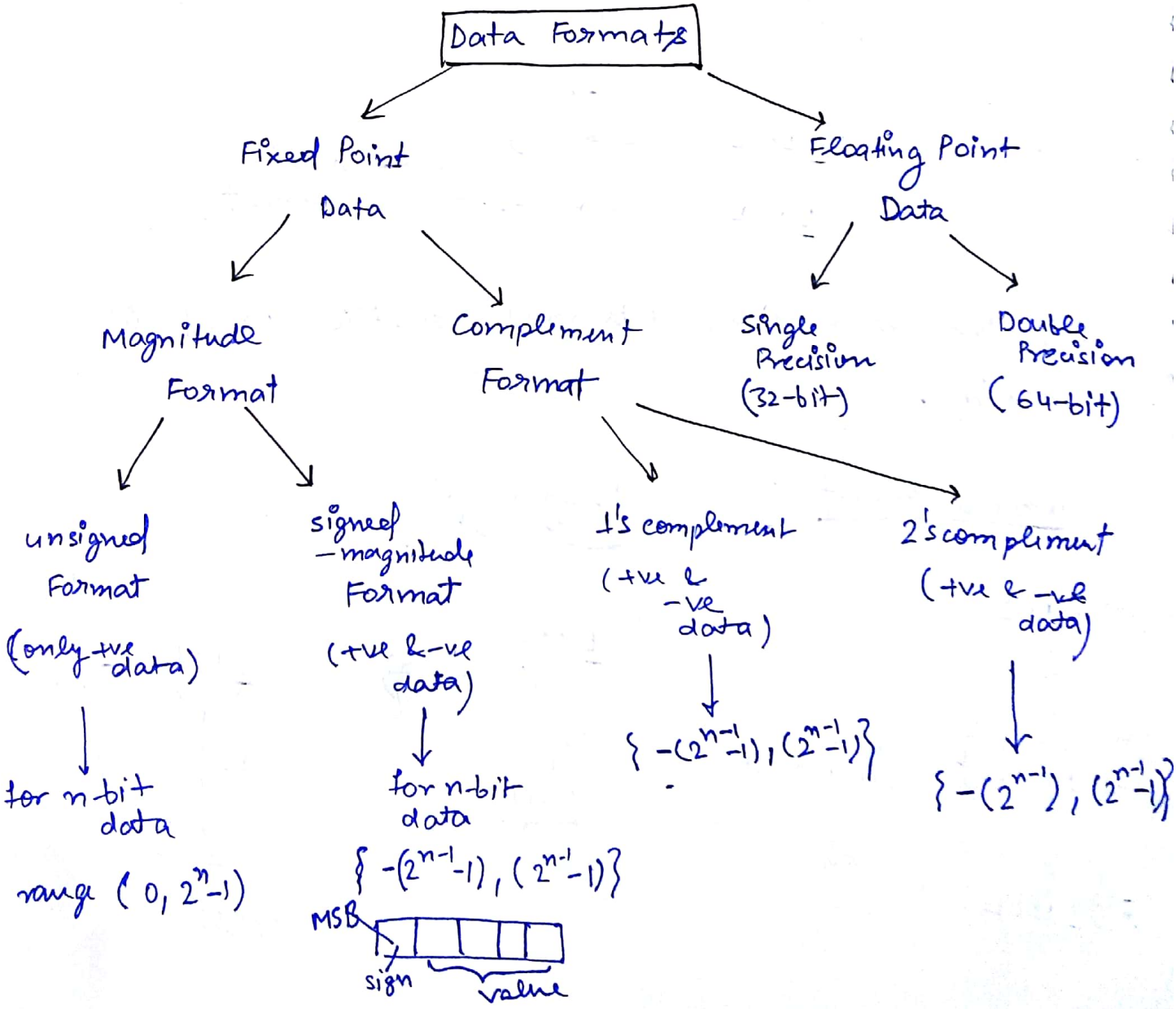
So, 2's complement is used –

0111 — +7

0101 — +5



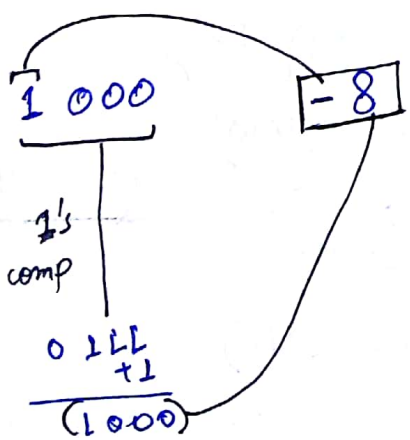
Data Representation :-



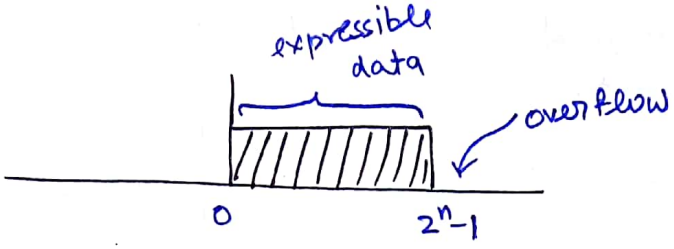
Fixed point

4-bit binary	unsigned data	signed data	1's complement	2's complement
0000	0	(+0)	(+0)	+0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	(-0)	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	(-0)	-1

eg- for 2's complement



Unsigned Data :-



4-Bit data = { 0 to 15 }

$$\begin{array}{r} 15 \\ +15 \\ \hline 30 \end{array} \qquad \begin{array}{r} 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ \hline (1)\,1\,1\,1\,0 \end{array}$$

carry (overflow) bit

So, if nbit + nbit = (n+1) bit

1-Bit storage space required

↓

1 flip-flop used

↓

Flag

↓

Carry Flag

condition - Is there extra bits (overflow) (Addition)

[True] [False]

set (C = 1) reset (C = 0)

or

Subtraction

Is borrow into the MSB

→ [True] set flag = 1 (C)

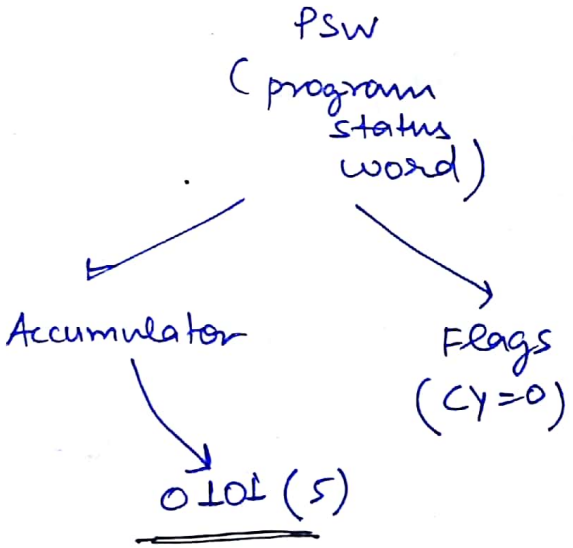
→ [False] reset flag = 0 (NC)

Eg:- 4-bit computation

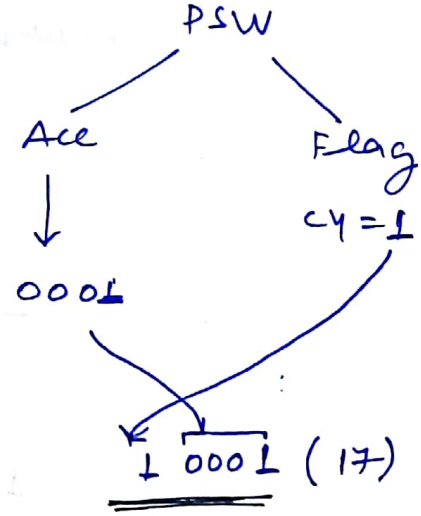
3	0011
+2	0010
5	0101
	CY = 0

8	1000
+9	1001
17	1 0001
	CY = 1

so, Justification



Justification



* Carry Flag is used to hold the range exceeding condition of a unsigned data.

Multiplication of unsigned numbers :-

- This operation is controlled by the multiplier
- Based on the multiplier bits, partial product is generated i.e when the multiplier bit is 1 then partial product is multiplicand otherwise partial product is 0.
- After the generation of a partial products, perform the summation of them to conclude the final result.

eg -

$(1111)_2 * (1111)_2$

multiplicand — multiplier

```
    1111
  * 1111
  ------
    1111
   1111
  1111
 1111
+1111000
--------
 11100001
--------
```

Result (8-bits)

so, for n-bits input result is 2n-bits

Note:

n-bits * n-bits = 2n bits

↓

n-bits extra storage space is required

↓

Register pair is used to represent the result.

Q. Consider the following multiplication.

$(1\,0\,W\,1\,Z)_2 * (15)_{10} = (Y\,0\,1\,0\,1\,1\,0\,0\,1)_2$

what are the value of W, Y, Z ?

```
    1  0  W  1  Z
  *    1  1  1  1
-------------------
    1  0  W  1  Z
 1  0  W  1  Z
 1  0  W  1  Z
-------------------
 Y  0  1  0  1  1  0  0  1
```

Z = 1

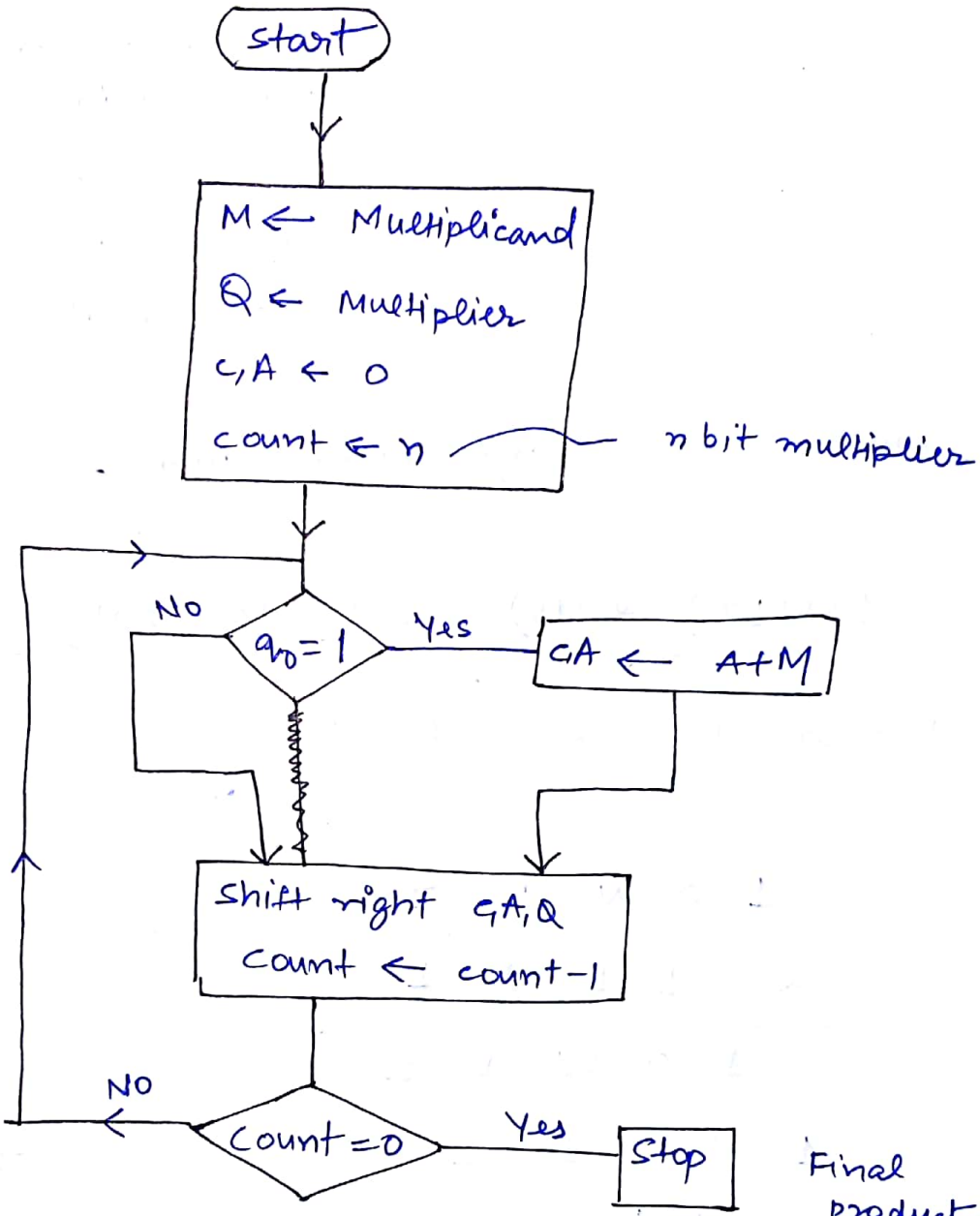
W = 1

Y = 1

* Limitation in the manual multiplication is -
  - ↳ Requires more register to store the partial product.
  - ↳ summation becomes complex.

So optimization is required i.e accumulated addition.

Unsigned Multiplication :-



4-bit multiplication (11 * 13)

M = 1011 Q = 1101

count = 4

C	A	Q	
0	0000	1101 (q_0)	initial
0	1011	1101	CA ← A+M
0	0101	1110 (q_0)	shift right
			count = 3
0	0010	1111 (q_0)	shift right
			count = 2
0	1101	1111	CA ← A+M
0	0110	1111 (q_0)	shift right
			count = 1
1	0001	1111	CA ← A+M
0	1000	1111	shift right
			count = 0

Result

STOP

So, Result = AQ

= 10001111

= 143

Division :-

- In this operation, dividend bits are scanned from MSB to LSB in a bit-wise sequence.
- After each bit scanning, partial dividend is compared with divisor.
- when the partial divident is greater than or equal to divisor, then put 1 in quotient and substract the divisor from it.
- Go to next bit scanning otherwise put 0 in the quotient.
- Continue this procss all the dividend bits are procssed.

$7 \div 2$ $\quad Q = 3$, $R = 1$ $\quad$ since $7 = 2 \times 3 + 1$

($2n$ bits / n-bit / n-bit)

so, for n-bits divisor $2n$-bits dividend is taken.

so, Divisor = 0010

then divident = 00000111

0010) 00000111 (00000011 (Q)

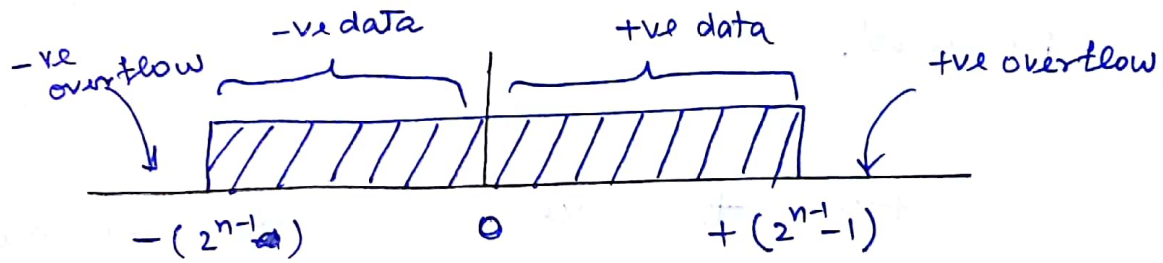
- 0010

00011

00010

... 00001 (R)

Signed Data :- (default 2's complement)



4-bit data = { -8 to +7 }

eg :-

	cy=0	
+7	0111	
+7	0111	
+14 (+ve overflow)	1110	
	OV = 1	

	cy=1	
-8	1000	
-8	1000	-ve overflow
-16	0000	
	OV=1	

	cy=1
+6	0110
-3	1101
+3	0011
	OV=0

- When two same signed (MSB) numbers are added (two positive or two negative) then there is a possibility of overflow.

+ve + +ve — may be overflow

or

-ve + -ve

whereas when two numbers of different sign are added then no overflow will be occured.

+ve + -ve — no - overflow

n-bit + n-bit = (n+1) bit

↓

1-bit storage space

↓

1-Flip-Flop reg

↓

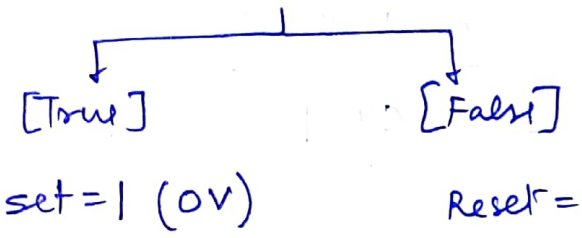
Flag needed

↓

Overflow flag

Used to hold the range exceeding condition

COND:- There is a carry into the MSB & no carry out of the MSB (or) vice versa



operation :- XOR op^n b/w in-carry & out carry w.r.t MSB

Expression :-

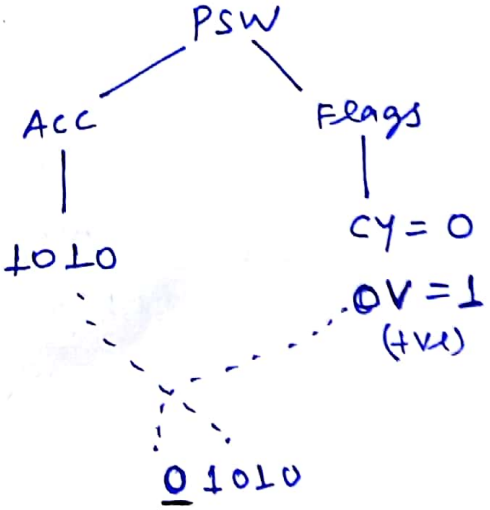
$$\left(OV(x,y,z) = \underbrace{\bar{x}\bar{y}z}_{\text{+ve OV}} + \underbrace{xy\bar{z}}_{\text{-ve OV}} \right)$$

$x \rightarrow$ MSB of operand 1
$y \rightarrow$ MSB of operand 2
$z \rightarrow$ MSB of result

Eg:-

+7	0 1 1 1
+3	0 0 1 1
+10	1 0 1 0
+ve OV	CY=0

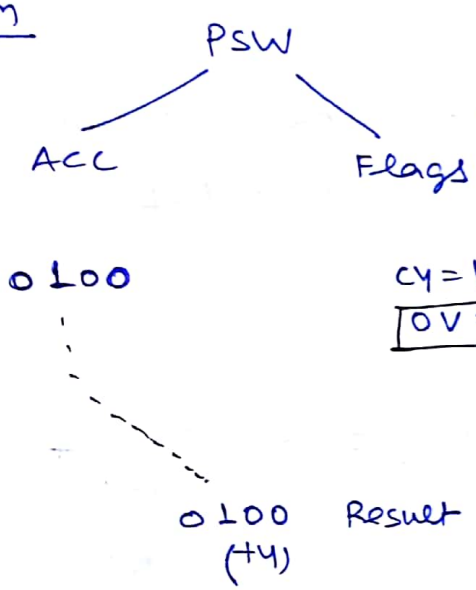
Justification



Eg:-

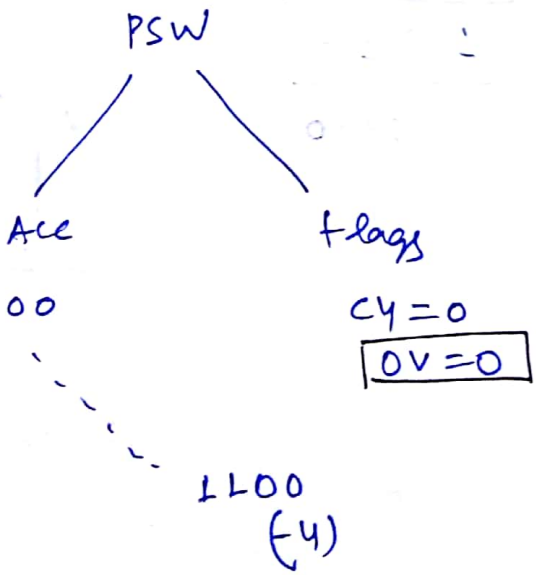
+7	0 1 1 1
−3	1 1 0 1
+4	0 1 0 0
OV=0	CY=1

Justification



Eg:-

−7	1 0 0 1
+3	0 0 1 1
−4	1 1 0 0
~~overflow~~	CY=0
OV=0	OV=0



Sign Extention (Replication of MSB)

-7 : 1001 (4-bit Register)

↓

stored in 8-bit Reg

↓

+9 : | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | padding 0

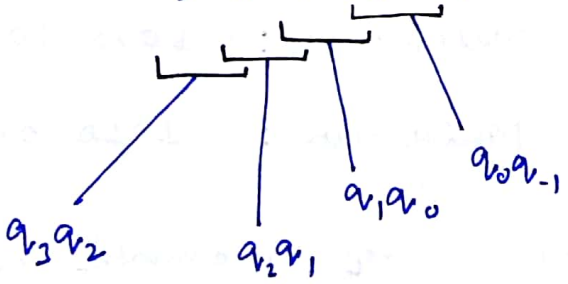
-7 : | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | Replication of MSB

sign Extention

Signed Multiplication (Booth's Multiplication)

- This multiplication is controlled based upon the multiplier bit-pairs.
- Bit-pair is formed based on multiplier. In this process one assumption is required ie ($q_{-1} = 0$).

eg: Multipliers : $q_3\ q_2\ q_1\ q_0\ q_{-1}$



Multiplier	Pair with
q_0	q_{-1}
q_1	q_0
q_2	q_1
q_3	q_2

- Following operations are defined w.r.t bit pairs, to control the multiplication.

Bit-pair	operation	Re-coding
0 0	ASR	0
0 1	+ & ASR	+1
1 0	− & ASR	−1
1 1	ASR	0

ASR: Arthumatic shift right

Q. Consider the following booth's multiplication:

Multiplicand: 1000 1010 1100

Multiplier : 1010 0101 1101

(a) How many arthmatic operations are required in multiplication.

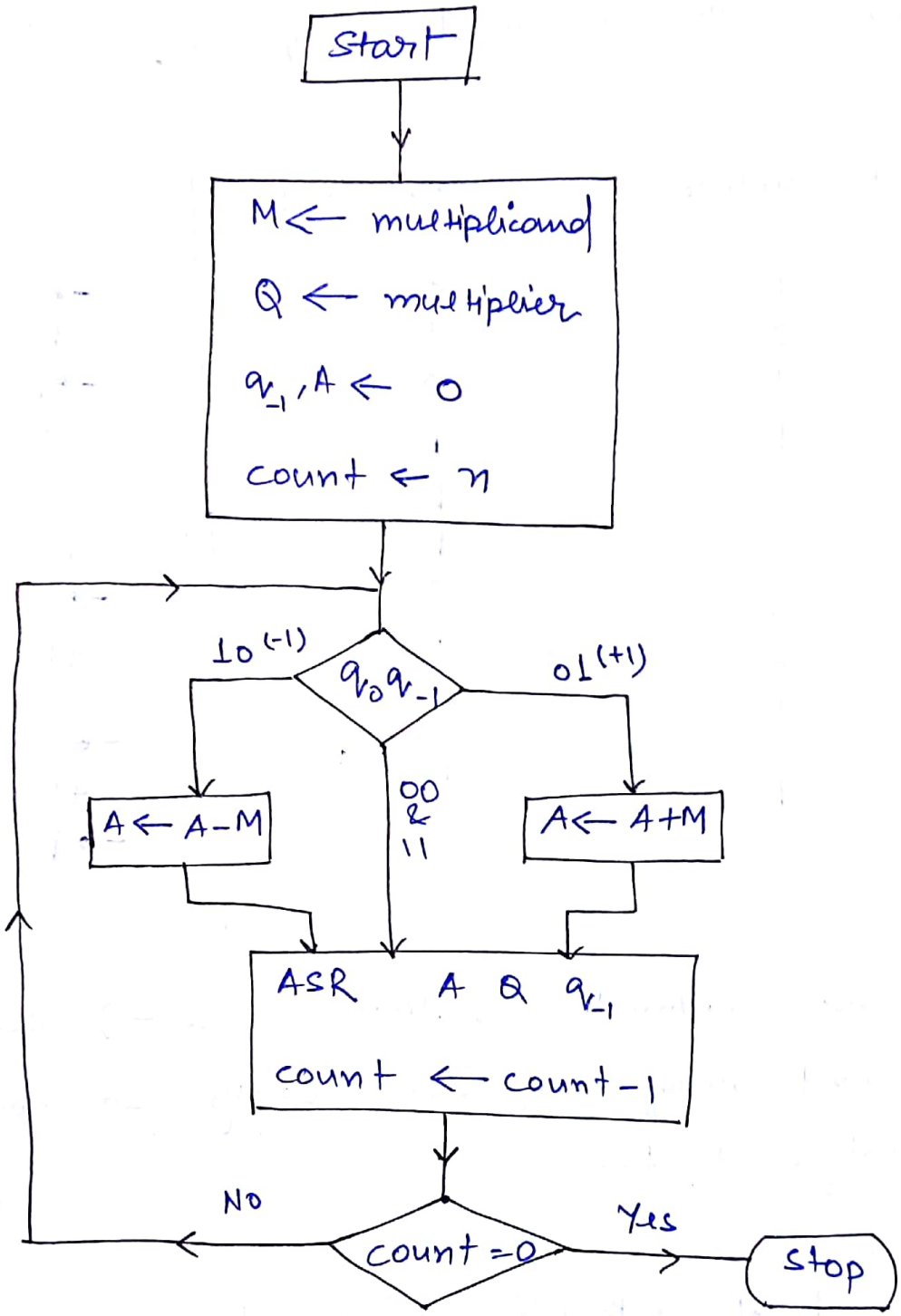
(b) what is the re-coded multipliers.

Multiplier	Pair with	Re-coding
1	0	-1
0	1	+1
1	0	-1
1	1	0
1	1	0
0	1	+1
1	0	-1
0	1	+1
0	0	0
1	0	-1
0	1	+1
1	0	-1

so, for arthmatic operation (+ & -) are occured during +1 and -1 recoding so count number of 1's.

(a) No of arthmatic op^{n} = 9

(b) Recoded multiplier : -1 +1 -1 0 +1 -1 +1 0 0 -1 +1 -1



Final product in AQ Reg Pair

Eg:- 4-bit multiplication

(-7) * (-3)

1001 1101

Bit-pairs:-

Multiplier		Pair with		Recoding
q_0	1	q_{-1}	0	-1
	0		1	+1
	1		0	-1
	1		1	0

Recoded multiplier = 0 -1 +1 -1

Now,

M = 1001 , count = 4

A	Q	q_{-1}	
0000	1101	0	initial
0111	1101	1	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	ASR
			count ← 1
0001	0101	1	ASR
			count ← 0

Result

STOP

Result = 0001 0101 = +21

Q. Consider the following binary code, processed on a addition hardware. What is the result in decimal when the data is in ~~unsigned~~

(a) unsigned format

(b) signed format

Data = 1010 1101

1101 1010

~~(a) For unsigned format~~

1 ← carry-in

1010 1101

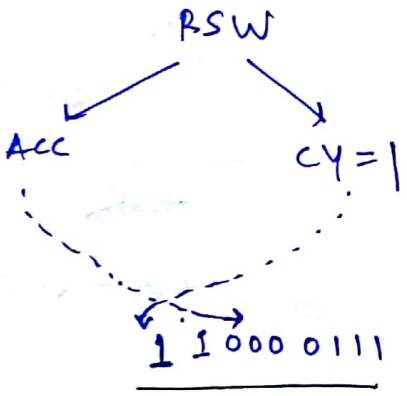
\+ 1101 1010

---

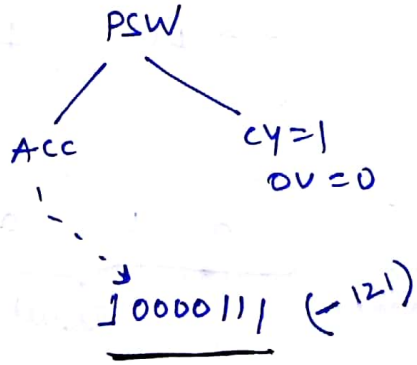
1 1000 0111 CY = 1

carry out OV = 0

for unsigned format



for signed format



Q. Consider the following 8-bit data computation. What is the status of overflow flag and its expression after the computation.

$(-120) + (-63)$

-120
-63
-183

since data is in 8-bits

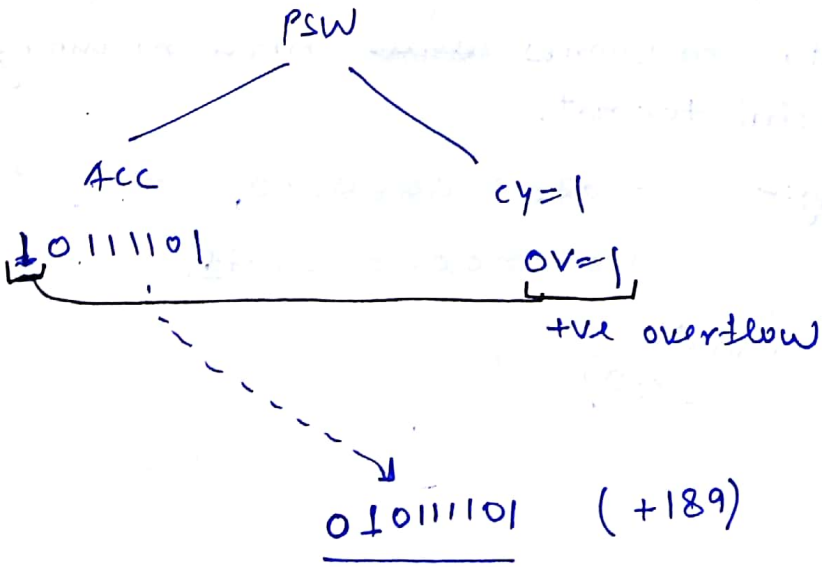
so, range is { -128 to $+127$ }

so, -ve overflow is there

expression = $xy\bar{z}$.

Q. Consider the Following status present in the hardware, after processing the signed data. What is the result in decimal.

ACC = 10111101 CY=1 OV=1



Q. Consider the following binary data stored in 16-bit CPU. What is the status of a CPU register when the data is in

(a) unsigned format

(b) signed format

Data : 1011011011011

Unsigned format : 000000 1011011011011

signed format : 111111 1011011011011

Floating Point Data Format :-

- Representation of a very large or very small fraction of data consumes ~~whole~~ more memory space using fixed point format.

eg:- +875000000000 — very large ($\approx \infty$)

+0.000000000875 — very small (≈ 0)

- To represent the large and small fraction of data within a less storage space, floating point format is used.

- General form of a floating point data is:

$$(\pm M \times B^{\pm e})$$

where M is mantissa / significand (Fraction)

B is base / Radix

e is exponent

- Normalization :-

$$\left(\pm \underset{\text{digit}}{\text{valid}} . \text{Fraction} \times B^{\pm e}\right)$$

- To represent data in floating point format, normalization concept is used to bring the data into a uniform structure.

- In the computer system, data is always represented in a form of binary so, $(B = 2)$ therefor valid digit becomes implicit i.e 1.

- General form of normalization is —

$$\pm 1.bbbb\ldots \times 2^{\pm e}$$

implicit — Default

- Mantissa Alignment process is used to adjust the decimal point. In this process right alignment increase the exponents and left alignment decreament the exponent.

eg:- $+ 1011.0111 \times 2^{+2}$

↓ normalization

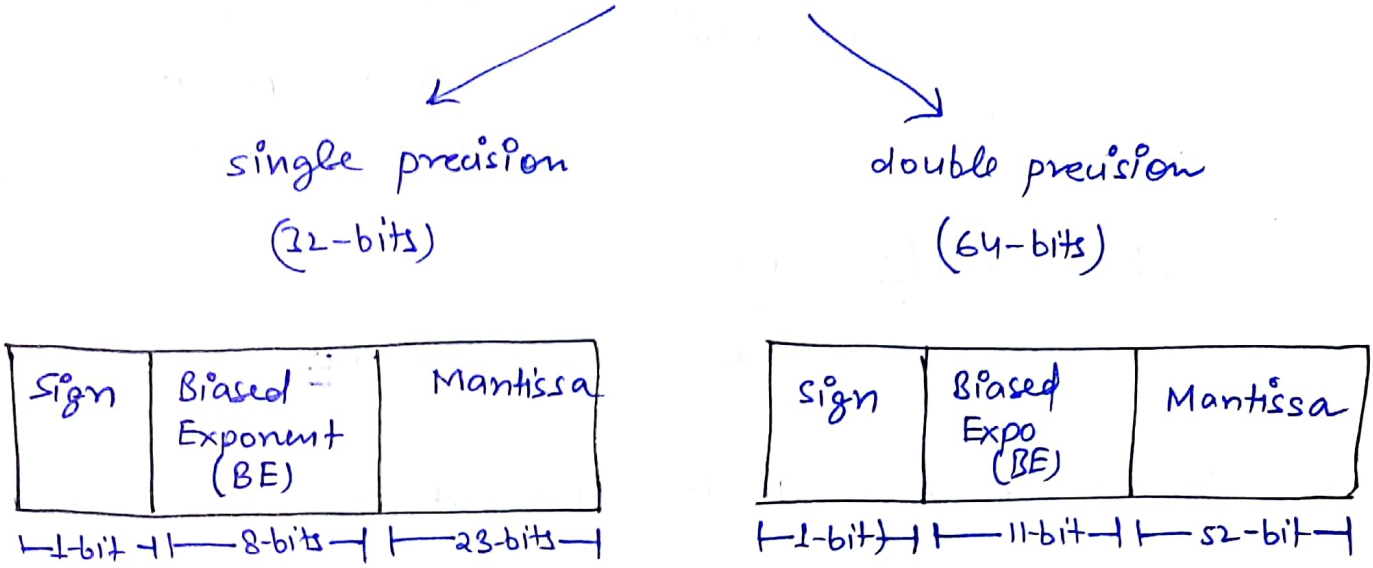
$1.0110111 \times 2^{+2+3}$

↓

$(1.0110111 \times 2^{+5})$

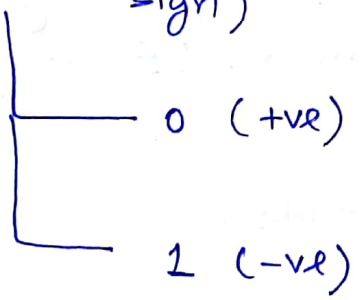
- when the data is in normal format then it is stored into the memory using the IEEE formats.

- According to IEEE standard 754 two kinds of format present :



- Floating point data is stored in memory with 3 fields of information.

(i) <u>Sign</u> : Used to represent ~~sign~~ sign of data (mantissa sign)



(ii) <u>Biased exponent</u> : Used to represent the exponent.

In the memory exponent is always stored in a biased format rather than 2's complement because MSB is already reserved for mantisa sign.

BE = AE + bias

AE: Actual Exponent

bias: maximum possible positive exponent, (depends on the format)

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

where n is size of BE

This concept is used to represent the sign of exponent.

- BE > Bias: then AE is +ve
- BE < Bias: AE is -ve

* To retrieve the data from the memory, we need to substract the bias from the BE to get AE.

ie AE = BE - bias.

eg: if BE = 158

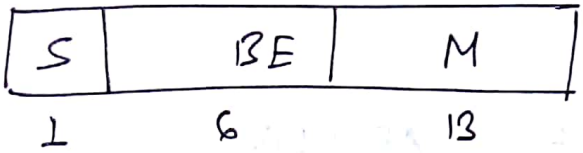
then AE = BE - bias

= 158 - 127

(AE = 31)

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

Eg:- custom Format :-



Data = $+1.1011 * 2^{+13}$

Store

$BE = AE + Bias$

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

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

$= +(2^5-1)$

$= +31$

Bias = +31 and AE = +13

Now,

AE : 0 0 1 1 0 1

\+ Bias : 0 1 1 1 1 1

BE : 1 0 1 1 0 0

Retrieve

$AE = BE - Bias$

BE : 1 0 1 1 0 0

\- Bias : 0 1 1 1 1 1

AE : 0 0 1 1 0 1

AE = +13

BE > Bias so AE is +ve

Data = $+1.1011 \times 2^{-13}$

13 : 001101

1's ↓

110010

+1

110011

Store

BE = AE + Bias

AE : 1 1 0 0 1 1

Bias : 0 1 1 1 1 1

BE : 0 1 0 0 1 0

Retrieve

AE = BE − Bias

BE : 0 1 0 0 1 0

− Bias : 0 1 1 1 1 1

AE : 1 1 0 0 1 1

(BE < Bias)

~~ex~~ AE is -ve

NOTE: when format is designed with an excess bias, then bias is a center of the exponent range

i.e $Bias = \frac{2^n}{2}$ n is no. of bits in BE field

In this format, we can take the complement of a MSB bit of BE to get AE rather than the subtraction

Format

S	BE	M
1	6	13

↳ Excess bias

Data : $1.10111 \times 10^{+13}$

STORE : $BE = AE + Bias$

$Bias = +2^{n-1} = 2^{6-1} = +32$

so,

$AE = +13$

AE : 001101

\+ Bias : 100000

BE : 101101

RETRV : $AE = BE - Bias$

BE : 101101

Bias : 100000

\- 001101

$AE = +13$

or

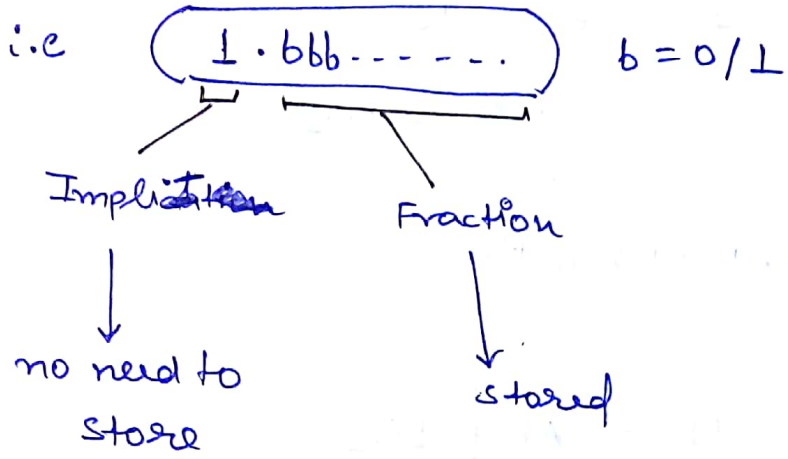
BE. 101101

1's ↓

AE 001101

(iii) Mantissa :- used to represent the fraction.

In memory mantissa is always stored in a normalized mantissa format.



Note:- To retrive the data from the memory denormalization procss is used to get the implit digit into a fraction i.e align the mantissa to right direction upto one time.

From the memory when we retrive -

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

↓

De-normalization

↓

Align-to-right upto 1 time

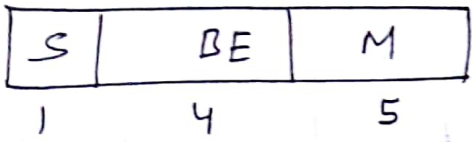
↓

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

↓

$$0.1bbb \ldots \times 2^{\pm e'}$$

Eg: Format



Data = − 6.75

STORE

Data (−6.75)

↓

Binary data (−110.11)

↓ normalization

-1.10112×2^{2}

Now sign bit = 1 (-ve)

BE = AE + bias (normal bias)

& bias = $+2^{n-1}-1$ = +7

AE = +2

AE = 0010

bias = 0111

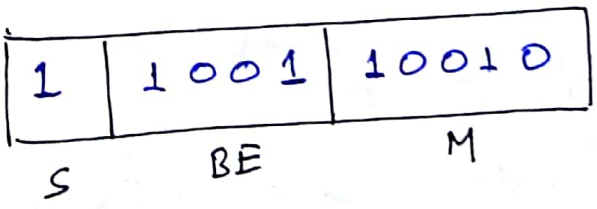
\+

BE = 1001

5-bit

10010

padding bit



READ:

mem data

1	1001	10110

Sign = 1 i.e -ve

BE = 1001

so AE = BE - bias

BE = 1001

- bias = 0111

AE = 0010

so, AE = 0010 i.e (+2)

Now Mantissa = 10110

i.e 0.10110

so, data = $-0.10110 \times 2^{+2}$

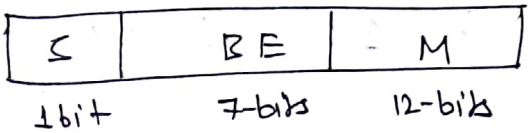
↓ de-normalization

$-0.110110 \times 2^{+2+1}$

implicit

i.e data = $-0.110110 \times 2^{+3}$

Q. Consider the following hypothetical format used to represent the format:



Data = $-13.25 \times 2^{+9}$

what is its hexadecimal equivalent when the data is stored in the memory -

(a) without normalization

(b) with normalization

$0.25 \times 2 = 0.5$
$0.5 \times 2 = 1.0$
0.0

(a)

$-13.25 \times 2^{+9}$

↓ binary

-1101.01×2^{9}

no normalization

sign = 1

BE = AE + bias

$\therefore$ bias = $+2^{7-1} - 1 = +2^{6} - 1 = +63$

AE = +9

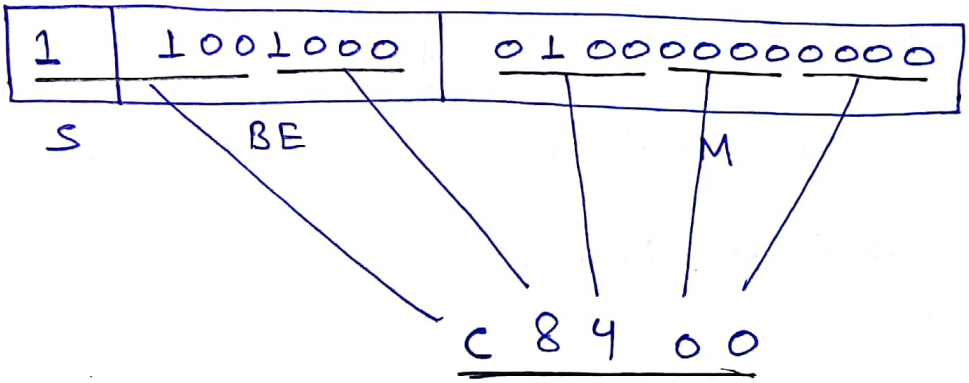
E AE : 0 0 0 1 0 0 1

bias : 0 1 1 1 1 1 1

BE : 1 0 0 1 0 0 0

Mantissa = 1 1 0 1 . (0 1)

store only



(b) we get -1101.01×2^{9}

apply normalization

-1.10101×2^{12}

so, sign = 1

mantissa = 10101 0000000

AE = 12 = 0001100

bias = +63 = 0111111

So, BE = AE + bias

AE : 0 0 0 1 1 0 0

\+ bias : 0 1 1 1 1 1 1

BE : 1 0 0 1 0 1 1