(1) Unsigned

→ Carry always indicate overflow.

(2) Signed

→ Two cases for overflow condn

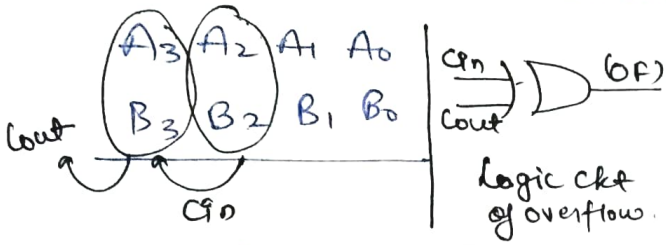
•) two (+ve) add to give (-ve) no.

•) two (-ve) add to give (+ve) no.

Note → In case of signed, carry doesn't indicate overflow.

Trick

→ Assume two 4 bit numbers

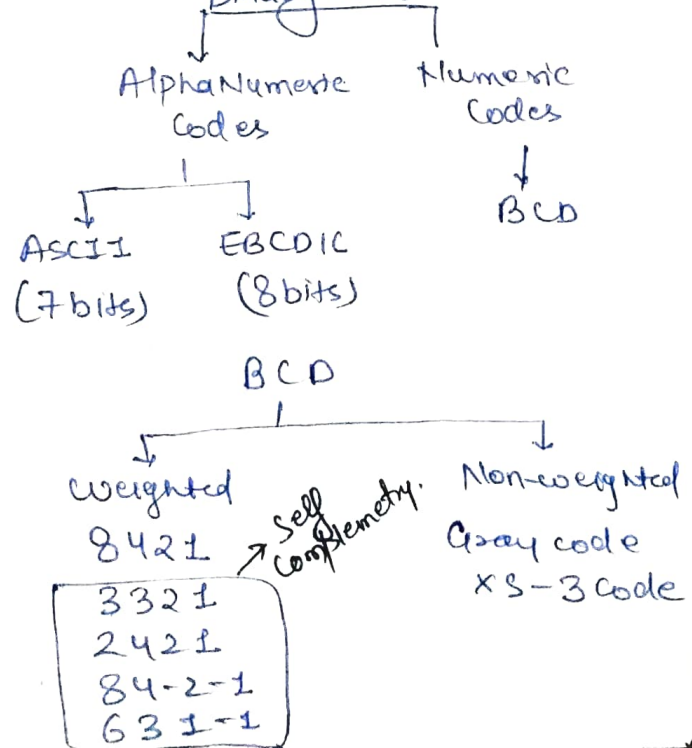


If Cin ≠ Cout then overflow

If Cin = Cout then no overflow

→ Verify all the previously solved examples by above Cin, Cout tricks.

Classification of binary Codes



Gray code called as unit distance codes

XS-3 code is sequential and self complementary.

Self Complementary Codes

In self complementary codes, 1's complement of patterns represents the 9's complement of digit.

Suppose a weighted code

6 3 1 −1

0 1 0 1	=	2
Pattern		Digit
↓ 1's comp.		↓ 9's comp
1 0 1 0	=	(9-2) = 7

and 1010 = 7 in (6 3 1, -1) code

So, 6 3 1 -1 is a self comp. code.

Ex

2 4 2 1

0 0 1 0	=	2
↓ 1's comp.		↓ 9's com
1 1 0 1	=	7

→ 2×1 + 4×1 + 1×1 = 7

8421, XS-3, 3321 Codes

Decimal no.	8421	XS-3	3321
0	0000	0011	0000
1	0001	0100	0001
2	0010	0101	0010
3	0011	0110	0011
4	0100	0111	0101

5	0101	1000	1010
6	0110	1001	1100
7	0111	1010	1101
8	1000	1011	1110
9	1001	1100	1111
	(10-15) are invalid here	Add 3 to 8421 code	

→ In 3321 you might get end up with different combⁿ for same example.

'3' can be represented as

BCD	3	3	2	1	
3 ←	0	0	1	1	→ 3 combⁿ
4 ←	0	1	0	0	
8 ←	1	0	0	0	

Go for min. BCD number

→ Choose the combination that gives min. BCD number upto Decimal number 4.

Prob

For no. greater than 4 take its 9's complement and then take 1's complement of pattern denoting obtained no.

Ex - for 5 do (9-5) = 4

4 represent by 0101 then take its 1's complement

So 5 = 1010

Similarly obtain 3321 code for 6, 7, 8, 9.

→ for a self complementary code sum of weights should be 9, for weighted code only.

Ex 3321 code is self comp. So 3+3+2+1 = 9

It is necessary condition not sufficient condition.

Suppose 'a' weighted code

	a	b	c	d	
b ←	0	1	0	0	→ Say (x) in BCD
1's	1	0	1	1	→ (9-x)

a + c + d

⇒ (9-x+x) = a+b+c+d

⇒ a+b+c+d = 9

→ 8421 is sequential code because we get another number by just adding 1. Similarly XS-3 is also sequential.

→ 3321 is not sequential code.

Examples on Code

Which of the following is invalid 8421 code.

i) 0010 ii) 1100 iii) 1001

iv) 0111

i) → 2 (valid) ii) 1100 → 12 (invalid)

bcoz 8421 is BCD ranges in (0-9).

Represent $(863)_{10}$ in a) BCD b) Binary c) 2241 d) 3321

(e) xs-3.

Soln ① (8 6 3)

1000 0110 0011

BCD → (1000 0110 0011)

---

② Binary

16 | 863	↑ 15 → F	
16 | 53		5
16 | 3		3
0		

$(35f)_{16} \rightarrow (0011\,0101\,1111)_2$

$(1101011111)_2$

---

③ 2 2 4 1

It is self complementary code.

8 6 3

9's com ↓ 9's com ↓ (0101)

1 3

(0001) (0101)

1's ↓ 1's Com ↓

1110 → 8 (1010) → 6

$(863)_{10} \rightarrow (1110\,1010\,0101)_{2241}$

Since 8, 6 are greater than 4 so we go for 9's comp. and then 1's comp.

④ 3321

(1110 1000 0011)

Approach same as 2241 for 3321

---

⑤ xs-3

Don't add 3 to carry digit

8 6 3 +3

+3 +3 6

11 9 (0110)

(1011) (1001)

$(863)_{10} = (1011\,1001\,0110)_{xs\text{-}3}$

BCD Addition

3 → 0011

+ 6 → 0110

9 1001 Ans

But if sum > 9 then we have to do some modification bcoz (10-15) form invalid binary pattern.

Ex 3 → 0011

7 → 0111

10 1010

(1010) is invalid in BCD

To represent 10 to decode 10 as (0001 0000) but it not valid in decimal

So, (10-15), 6 combinations are invalid. So for Sum > 9 add 6 to it and make it valid, as,

```
  3   0011
  7   0111
 ---  ----
 10 → 1010
+ 6  +0110
 ---  ----
 16   10000
 ---  -----
  1    0
```

```
  9
  9
 ---
 18  → max output we can get bcoz max. no. allow is 9
       but we can't go beyond 15

  1001
  1001
 -----
 10010 → 18
   ↓
 0001  0010
   1     2      (12) BCD.
```

Add 6 and get answer

```
 0001  0010
 0000  0110   (18) BCD.
 ----  ----
 0001  1000
   1     8
```

If sum > 9 then add 6 to the sum to get answer.

Ex

```
  1  2  3
  7  8  9
 ------
  9  1  2
 ------
```

In BCD

```
        carry
 0001 | 0010 | 0011
 0111 | 1000 | 1001
 ---- | ---- | ----
 1001 | 1010 | 1100  → 12 out of range
      | 0110 | 0110
      | ---- | ----
      | 0001 | 0010
```

```
 1001    0001    0010
  9       1       2
```

$(912)_{BCD}$

→ Correction used in BCD addition is +6.

XS-3 Addition

If carry occur add (+3) as correction.

If carry doesn't occur add (−3) as correction.

Input and output should have same no. of digits.

Ex ① 7 → 0111 + 0011 → 1010

2 → 0010 + 0011 → 0101

9 1111

↓ No carry

→ Subtract 3

```
  1111
  0011
 ----
  1100  → represent 9 in XS-3
```

② 07 → 0011 1010

03 → 0011 0110

10 → [illegible] → Cont

Bcoz of two digits in o/p we make to add '0' to make i/p of two digits

0011	1010
0011	0110
0111	(1) 0000
(−) 0011	(+) 0011
0100	0011
(1)	0
XS-3	(XS-3)

No carry (left); Carry generated (right)

Gray Code (Reflexive, Unit distance code, cyclic code, non-weighted)

for 1 bit — either 0 or 1 can fill it.

for 2 bit — Choose 1 bit
Put 0 in first half, Put 1 in 2nd half.

0	0	→ 0
0	1	→ 1
1	1	→ 2
1	0	→ 3

Mirror reflex

for 3 bits — Used here, Choose 2 bits

0	0	0	→ 0
0	0	1	→ 1
0	1	1	→ 2
0	1	0	→ 3
1	1	0	→ 4
1	1	1	→ 5
1	0	1	→ 6
1	0	0	→ 7

Mirror

for 4 bits Choose bits — Mirror for 4 bits

0	0	0	0	→ 0
0	0	0	1	→ 1
0	0	1	1	→ 2
0	0	1	0	→ 3
0	1	1	0	→ 4
0	1	1	1	→ 5
0	1	0	1	→ 6
0	1	0	0	→ 7
1	1	0	0	→ 8
1	1	0	1	→ 9
1	1	1	1	→ 10
1	1	1	0	→ 11
1	0	1	0	→ 12
1	0	1	1	→ 13
1	0	0	1	→ 14
1	0	0	0	→ 15

that is why it is called as reflexive.

Ex → Represent 12 in
1. BCD (00010010)
2. 3321 (0001, 0010)
3. Gray (1010)
4. Binary (1100)

→ Unit distance code because on going from one no. to another just one bit is changed only.

→ Cyclic bcoz distance between last number and first number is also one.

Ex 0 → 0000, 15 → 1000 } only 1 bit change

Binary to Gray and Vice Versa

12 → Binary (1100)

Binary to Gray

Binary 1 1 0 0
Gray 1 0 1 0

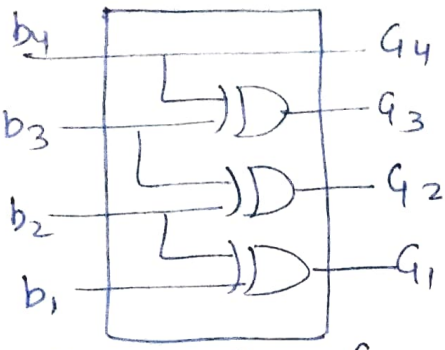
(1010) : 12 in Gray.

$b_4 \quad b_3 \quad b_2 \quad b_1$

$G_4 \quad G_3 \quad G_2 \quad G_1$

$G_4 = b_4, \quad G_3 = b_4 \oplus b_3$

$G_2 = b_2 \oplus b_3, \quad G_1 = b_1 \oplus b_2$



Binary to Gray Converter

Binary 15 → (1 1 1 1)

Gray 15 → (1 0 0 0)

Binary 200 to gray 200.

1 1 0 0 1 0 0 0 → Bino (200)

1 0 1 0 1 1 0 0 Gray(200)

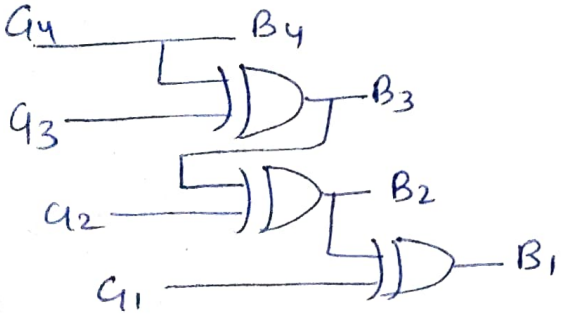
→ Gray to Binary

Gray 12 → 1 0 1 0

(12) Binary ← (1 1 0 0)

$G_4 = B_4$, $B_3 = B_4 \oplus G_3$

$B_2 = B_3 \oplus G_2$ $B_1 = G_1 \oplus B_2$



$b_3 = G_4 \oplus G_3$

$b_2 = G_4 \oplus G_3 \oplus G_2$

$b_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1$

Error Detection

We know that BCD have range (0-9). So if error occured as,

0 0 0 0] 1 bit gets corrupted
0 0 0 1]

but here both 0 and 1 lies in range 0-9. It means if both the pattern before and after error are valid pattern then it is impossible to detect the error.

0 0 0 0 (0) ← Here we can detect
↓ error bcoz 1100 doesn't
1 1 0 0 (12) ← lies in range (0-9).

So, (1100) is invalid code.

→ To detect 1 bit error the distance between 2 patterns should be atleast 2.

0 0 0 0] Error can be detect
0 1 0 1 0]

0 1 1 0] Error can't be detect
0 0 1 0]

→ To detect 'n' bit error in n bits, the distance should be atleast (n+1), also called hamming distance.

Error Correction

→ It includes detection and correction

→ for single bit error correction we need all valid combination with distance 3 and combination other than distance 3 is invalid.

→ To correct 2 bit error, distance between two valid patterns should be atleast 5.

→ for 3 bit error correction, distance between two valid pattern should be atleast 7.

→ for t-bit error correction, the min. hamming distance should be $(2t+1)$.

→ Bcoz of error correction, distance between two valid patterns can be very high and it reduces the no. of valid combinations. So, to get all the valid combination we have to increase the number of bits in pattern

→ Min. distance btw any two pattern called hamming distance.

Note → Just remember the imp. formula for hamming distance in error correction and detection. All the Qns will get solved but if very much needed only then watch the video again in downloaded videos.

Hamming Code :- (1-bit error correction Code)

Suppose message have m-bits and we add 'p' parity bits. So message finally contains

$m+p$ bits

Assume there is 1 bit error or no error in $(m+p)$ bits.

So, for $(m+p)$ bits total no of combination of error equals to

$(m+p)+1$ (No error)

for any bit of $(m+p)$ can get corrupted

So, condition is,

$$2^p \geq (m+p+1)$$

for adding no. of parity bits to check errors.

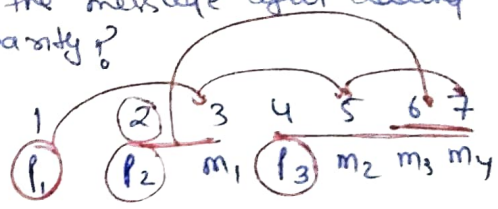
Suppose $m = 4$

$\Rightarrow 2^p \geq (p+5)$

So, $p = 3$ satisfy above cond^n means we send total $(m+p) = 7$ bits.

Ex: If $m = 4$ and $p = 3$, message (m) is 0101. What should be the message after adding parity?

Sol^n



Add parity bits at power of 2.

P1 P2 0 P3 1 0 1

(P3) C1	(P2) C2	(P1) C3	
0	0	0	— 0
0	0	1	— 1
0	1	0	— 2
0	1	1	— 3
1	0	0	— 4
1	0	1	— 5
1	1	0	— 6
1	1	1	— 7

P_1 → form pair of 1, P_2 form pair of 2, P_3 form pair of 4 and check at distance equals to no. of elements in pairs

C_3 is 1 at (1, 3, 5, 7)
C_2 is 1 at (2, 3, 6, 7)
C_1 is 1 at (4, 5, 6, 7)

So, every check bit will take care of respective position to correct errors.

→ Generally, by default we go for even parity.

→ So, $P_1 = 0$ bcoz, 5, 7 already have digit 1. so 2 (1's) means even parity.

Similarly $P_2 = 1$ so make even no. of pattern.

$P_3 = 0$. So message should be

0 1 0 0 1 0 1

P_1 P_2 P_3

Suppose corroped message is

0 1 0 0 0 0 1

C_1 C_2 C_3 [check bits]

1 0 1 (5).

Each check bit check according to respective position for even parity

→ Location 5 have error, So we can change it to

→ Check bit is 1 means error.

Imp:
→ We can have error in any (m+p) comb^n or 1 for no correction. So total comb^n possible is (m+p+1)

Floating point Conversion

Binary to decimal

Ex ① 110.101

2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3}

$6 + (0.5 + 0.25 + 0.125)$

$\times 1 \quad \times 0 \quad \times 1$

$\Rightarrow 6 + (0.5 + 0.125)$

$= 6.625$

② 1101.01

$13 + (0.25)$

$= 13.25$

Decimal to binary

① 0.625 to binary.

0.625
+ 0.625
1.250 → 1
0.250
0.500 → 0
0.500
1.000 → 1

$(0.101)_2$

→ Stop when you get all zeroes.

② 0.25 to binary

0.25
0.25
0.50 → 0
0.50
1.00 → 1

$(0.01)_2$

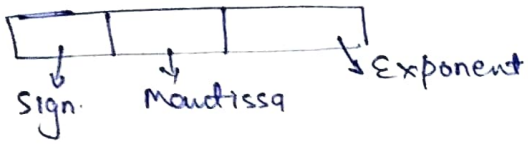
③ 0.623
0.623
1.246 → 1
0.246
0.452 → 0
0.452
0.904 → 0
0.904
1.808 → 1
0.808
1.616 → 1
0.616
1.232 → 1

So, for some no. it is not possible to give binary representation of fixed length.

→ If number can't represent completely in binary then we represent only some part of it.

Floating point representation - 1

$(4.5)_{10} = (100.1)_2$



$(100.1)_2$ can be written as

(0.1001×2^3)

Evaluate the value as

0.1001×2^3

(2^{-1} and 2^{-4})

$\Rightarrow (2^{-1} + 2^{-4}) \times 2^3$

$\Rightarrow 2^2 + 2^{-1}$

$\Rightarrow 4 + 0.5 = (4.5)$

So, (0.1001×2^3) is a correct representation, as

0	1001	11

So here is a problem that (0.01001×2^4) also represent (4.5) so we will get more than one combination for same number.

→ So, we will use mantissa, exponent method with normalization to avoid more combinations problem.

→ Normalization says use only one combination for a number.

→ Normalization says that move the decimal upto more significant '1'.

all zeros . → 1 followed by anything

do1

100.1

So, (0.1001×2^3) is only one valid combination

0	1001	11

Also called as 'explicit normalization.'

→ Mantissa and exponent can replace their position.

Suppose mantissa can hold 5 bits and exponent needs 4 bits then (0.1001×2^3) can be written as

0	10010	0011

Add zero after

Add 0's before

Ex: 0.00101

Above no. be represented

0.101×2^{-2}

We can represent (-2)(-ve) number using 1's or 2's complement but we use 2's complement

0	10100	1110

(-2)

2 = 0010

1's comp = 1101

2's comp = 1110

→ $(-1)^S \ 0.M \times 2^E$ use to get the original no. from representation.

Biasing → Converts every (-ve) number to (+ve).

→ It is difficult to check (-ve) exponent. So we generally prefer to convert it into (+ve) value and then write it.

Ex → 0.101×2^{-2}

→ -2 will represent as

$-2+16 = 14$

S	M(4)	E(5)
0	1010	01110

↳ biased exponent

Suppose you have exponent of n bits then in 2's comp. you can represent the range -2^{n-1} to $(2^{n-1}-1)$

then, bias used is (2^{n-1}).

means max. (-ve) no.

→ In 5 bit $2^{n-1}) = 16$

In case of biased exponent we can get the number as

$(-1)^S \times 0.M \times 2^{E-bias}$

$\Rightarrow (-1)^S \times 0.1010 \times 2^{14-16}$

$= 0.101 \times 2^{-2}$ Ans.

Ex: 4.875

(100.111)

→ Explicit normalization

0.100111×2^3

	M(5)	

To save 6 bit mantissa in 5 bit field we have to neglect a bit.

0	0	1	1	(1)	→ neglected

S	E(4)	M(5)
0	1011	10011

for 4 bit $-2^3 = -8$ (max (-ve) no.)

for exponent 3 in 2^3 we get the final exponent after adding 8 (biasing)

so, $E = 8+3 = (1011)_{11}$

$(-1)^S \times 0.M \times 2^{E-Bias}$

$\Rightarrow (-1)^0 \times 0.10011 \times 2^3$

$= 100.11$

$= (11.75) \neq 4.875$

So, we have two methods to avoid this,

① Increase no. of bits

② Implicit Normalization.

100.111 can be written in Implicit normalization as

1.00111×2^2.

It means move left upto 1 and don't cross the one, comes last.

S	E(4)	M(5)
0	10 10	00111

$2+8=10$

Biased exponent

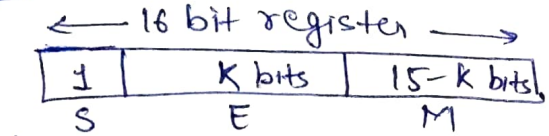
$(-1)^S (1.M) \times 2^{E-Bias}$

$(-1)^0 (1.00111) \times 2^2$

Example: Consider a 16-bit register of the following format is used to store a floating point number. Mantissa (M) is denoted as normalised signed magnitude fraction, Exponent(E) is expressed as excess-64 form. Base of system is 2.

(1) How many bits are allocated for fractional mantissa?

Soln



With k bit max. negative no. possible = -2^{k-1}

Bias $2^{k-1} = 64$

$\Rightarrow k = 7$

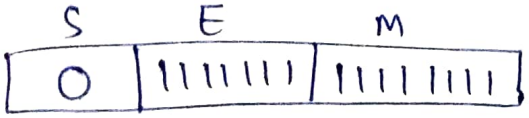
for fractional mantissa = 8 bits

(2) What is the expression for decimal value.

$[(-1)^S \times (1.M) \times 2^{E-64}]$

→ In general we use implicit norma.

(3) What is the largest no. represent in base 10.



$(-1)^0 \times 1.11111111 \times 2^{127-64}$

$= 1.(\frac{1}{2} + \frac{1}{4} + \cdots \frac{1}{2^8})$

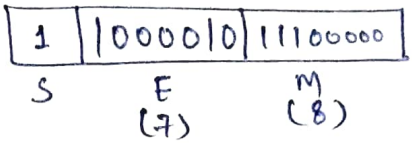
$= (1 + (0.11111111))2^{63}$

$= (1 + 1 - 2^{-8}) \times 2^{63}$

$= \underline{2^{64} - 255}$ Ans

(4) What is the 16 bit pattern for $(-7.5)_{10}$.

$= -(111.1)_2 = \underline{-(1.111 \times 2^2)}$

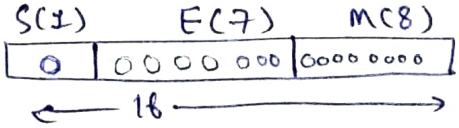


$E = 64 + 2 = 66$

$(-1)^1 \times 1.111 \times 2^{66-2}$

Example: (5) Question is same as previous examples

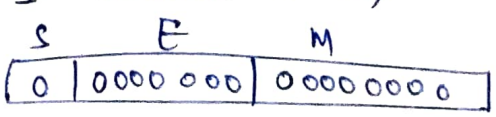
(5) What is the difference between 1st smallest (+ve) number and 2nd smallest (+ve) numbers



2^{16} possible comb^n

→ Numbers are densely distributed near '0' and sparsely distributed far from zero

1st smallest no. is,



$(1.0) \times (2)^{0-64}$

$= \underline{1.0 \times 2^{-64}}$

2nd smallest no.

S	E	M
0	000000	00000001

$(1.00000001) \times 2^{-64}$

Difference

$= (1.00000001 - 1.0) \times 2^{-64}$

$= \underline{2^{-72}}$

Extension

1st highest positive no:

$(1.1111111) \times 2^{63}$

2nd highest number

$(1.1111110) \times 2^{63}$

Difference = 255

So, difference is very small near 0 means densly packed and very far away from zero sparsly packed between two consecutive numbers.

(6) What is the pattern denotes the value $(-16.125)_{10}$

$-(10000.001)_2$

1	1000100	0000001

→ Pattern

Implicit normalization

$-(1.0000001 \times 2^4)$

M E

$4 + 64 = 68$ to store in exponent

Theory Floating point representation

Floating point numbers stored in exponent and mantissa form.

→ Generally we represent mantissa as normalized signed magnitude fraction.

→ Normalization can be explicit or implicit.

→ Exponent denoted in biased form.

→ Biased exponent is an unsigned number, which can represent in signed exponent of original number.

True Exp. = Biased Exp. − Bias

→ Representation

S	E	M

n

S = 1 bit E = k bit

M = (n − 1 − k)

Biased Exp. = $0 \leq E \leq 2^k - 1$

Bias = 2^{k-1} (2's complement)

IEEE Standards

1985 → Sigle, double precesion

2008 → half, Quadraple Precesion

1985 standards → More popular.

→ Base of system is 2.

→ Single Precesion → 32 bits

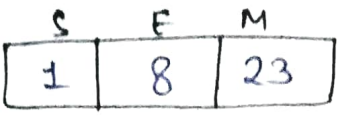
Double ——— → 64 bits

→ Floating point no. can be represent as −

-) fractional form
-) Implicit Normalized form.

→ Certain mantissa and exponent combinations doesn't represent any number (Not a number - NAN) it used to represent 0 and ∞.

Single Precesion



bcoz of 8 bits its (Excess-127)

Sign.	E(8)	M(23)	Value
0/1	00000000 E=0	000---0 M=0	± 0
0/1	11111111 E=255	000--0 M=0	± ∞
0/1	1 ≤ E ≤ 255	M = xx---x anything	Implicit Normalized
0/1	E = 0	M ≠ 0	fractional form
	E=255	M ≠ 0	NAN

Because we can't represent 0 and ∞ in as general so here we reserved some combinations for it, otherwise we willnt be able to represent it..

→ In Implicit Normalization value is

$$(-1)^S \times (1.M) \times 2^{E-127}$$

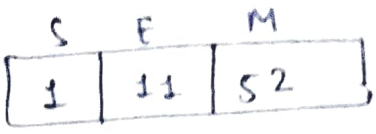
→ fractional form also known as denormalized form and value is

$$(-1)^S \times (0.M) \times [2^{-126}]$$

Instead of 2^{-127} here it is 2^{-126} and it is exception.

→ we can used E=255, M≠0 as a combination.

Double precision



Excess → 1023

Sign.	E(11)	M(52)	Value
0/1	E=0	M=0	± 0
0/1	E=2047	M=0	± ∞
0/1	E∈[1,2046]	M=xx---x	Implicit Normalized.
0/1	E=0	M≠0	fractional form
	E=2047	M≠0	NAN

M = xxx---x means, it can hold any combination.

→ In Implicit Normalization value is

$$(-1)^S \times (1.M) \times 2^{E-1023}$$

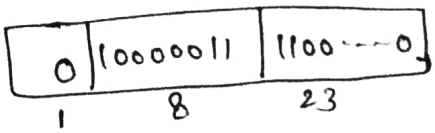
→ fractional form

$$(-1)^S (0.M) \times 2^{-1022}$$

Exception.

Ex→ Consider 32 bit register which stores floating point number in IEEE single precision format

1) what is the value of number, if 32 bits are given below



Sol^n

$(-1)^S (1.M) 2^{E-127}$

$(-1)^0 (1.11) 2^{131-127}$

$= (1.11) \times 2^4$

$= 11100. \quad (28)_{10}$

2) What is the value if 32 bits are given as.

S	E	M
0	11111111	11000 -- 01

→ $E = 255$ $M \neq 0$ means it is NAN.

Q If we have 64 bit register to stores floating point no. in IEEE double precesion format. What it the value of number given in 64 bits below

1	11	52
1	10000000011	1101100 ---- 0

Implicit normalized form

$(-1)^1 (1.11011) \times 2^{1027-1023}$

$-1 \times 1.11011 \times 2^4$

$-(11101.1)_2 = -(29.5)_{10}$

Ex: Represent $(117)_{10}$ in IEEE 754 single and double precesion formato

Sol^n

$(1110101)_2 = (117)_{10}$

(1.110101×2^6)

Mantissa going to be same in both formato

for single precesion Bias = 127

$E = 6 + 127 = 133$

for double precesion Bias = 1023

$E = 1023 + 6 = 1029$

Single precesion →

0	10000101	11010100 -- 0
(1)	(8)	(23)

double precesion

1	10000000101	11010100 -- 0
(1)	(11)	(52)

Gate 2016: Let X be the no. of distinct. ---, then X-Y is ——

Sol^n

Using 16 bits we get 2^{16} comb^{n}o

In 2's comp. we denoted each comb^n uniquely so, 2^{16} combination are possible

In signed mag. representation we waste two comb^n for 0. So we consider $2^{16}-1$ numbers, distinctly.

So, $X - Y = 2^{16} - (2^{16}-1) = 1$ Ans

Binary Multiplication (partial sum method)

$$\begin{array}{r} 0101 \rightarrow 5 \\ \times\ 0110 \rightarrow 6 \\ \hline 0000 \\ 0101\times \\ 0101\times\times \\ 0000\times\times\times \\ \hline 00111110 \end{array} \qquad 5*6 = 30$$

$= (11110)_2 = (30)_{10}$

To multiply 2, n bits numbers, in worst case it will take 2n bits to store Answer.

Ex → To multiply 2, 4 bits no. we need register of size 8 bits

0 1 0 1
× 0 1 1 0
0 0 0 0
0 1 0 1 ×
0 1 0 1 × ×
0 0 0 0 × × ×
0 0 1 1 1 1 0

[0|0|0|0|0|0|0|0]
0 0 0 0
0 1 0 1
[0|0|1|0|1|0|0|0]
shift >>1
[0|0|1|1|0|1|1|0]
0 1 0 1
[0|0|1|1|1|1|0|1|0]
shift >>1
[0|0|1|1|1|1|1|0|1|0]
+ 0 0 0 0
[0|0|1|1|1|1|1|1|0|1|0]
shift >>1
[0|0|0|1|1|1|1|1|1|0]

Ans

→ After adding into Most significant bits shift the value by 1 digit.

No of Adds require = No. of 1's in multiplier

No of shift require = No. of bits in multiplier

Idea Behind Booth's Algo.

$$0\,1\,1\,1\,1\,0 = (2^{4+1} - 2^1)$$
$$2^5\ \underline{2^4}\ 2^3\ 2^2\ \underline{2^1}\ 2^0$$

$$0\,1\,1\,0 = (2^{2+1} - 2^1)$$
$$2^3\ 2^2\ 2^1\ 2^0$$

General

$$0\,1\,1\,1 \cdots 1 \cdots 0 = (2^{j+1} - 2^i)$$
(2^j ↑ , 2^i ↑)

2^j represent MSB and 2^i represent LSB in sequence of 1's in binary no.

M × Q
↓ Multiplicant
Q → Multiplier

If Q = 011110 means it will take 6 shift and 4 addition will be perform.

$\Rightarrow M \times (2^5 - 2^1)$

$= (2^5)M + 2^1(-M)$

Multiply any no with 2^n means n times left shifts.

So in above computation
total shift = $2^5 + 2^1$
= 6 shift

But instead of 4 addition we will perform only 1 additions.

Example

$$\begin{array}{r} 0\,1\,0\,1\,1 \quad \rightarrow M \\ \times\ 0\,1\,1\,1\,0 : (2^4 - 2^1) \rightarrow Q \end{array}$$

$(2^4 - 2^1)M$

$= 2^4 M + 2^1(-M)$ — ①

Here $(-M) = 10101$

$(2^1) \times M = \underline{10\,1\,0\,1\,0}$ (1 left shift by 1)

$(2^4) \times M = \underline{0\,1\,0\,1\,1\,0\,0\,0\,0}$

111 1 0 1 0 1 0 — we fill it with 1's to maintain the significance of (-ve) sign.
× 0 1 0 1 1 0 0 0 0
① 0 1 0 0 1 1 0 1 0 → $(154)_{10}$

discard it bcoz we are using 2's comp. arithmatic

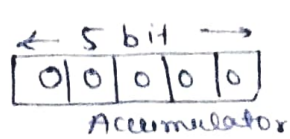
If don't understand here watch other YT video

Booth's Algorithm

We are explaining the concept by same example discussed above.

M = 01011

−M = 10101



A	Q	Q-1	
00000	01110	0	
00000	00111	0	Right shift
10101	00111	0	A − M
11010	10011	1	Right shift
11101	01001	1	>> shift
11110	10100	1	>> shift

↳ obtained pattern

Now add M to to obtained pattern

```
 11110 10100
 01011 00000   ] A+M
(1) 01001 10100
 ↓          ↳ (154)10
discard it
```

Note: Do atleast 5 Question on Booth's Algorithm.

Operations perform

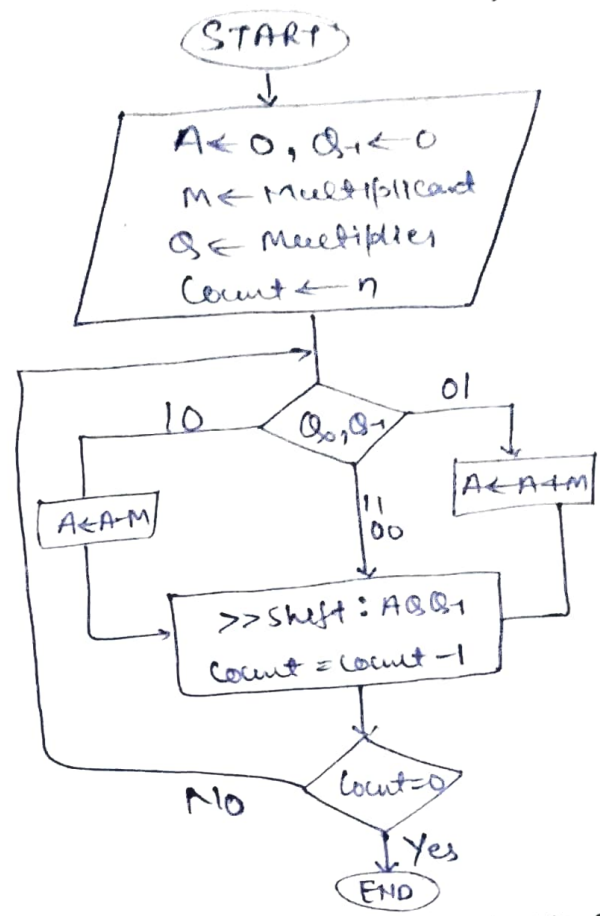
LSB(Q)	Q-1	
0	0	>> shift
1	0	A−M / >> shift
1	1	>> shift
0	1	A+M / >> shift

→ No. of steps = no. of bits in M

Booth Algorithm flow Diagram

Scan Qn to p

Cont −



Advantages and disadvantages of Booth's

→ In general the number of operations required will be reduced.

→ Ex 01110 → 1 ADD, 1 SOB and 6 shifts.

for each sequence of consecutive 1's we perform 1 ADD, 1 SOB

Ex

01110011110110 → 3 ADD, 3 SOB

→ It can do signed multiplications

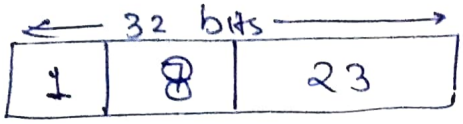
→ If negative no. are involved, it give better performance.

Disadv.: If block contains only single 1.

Ex: 0101010101

Example on fractional form

What is the range of floating point number represented using fractional form of IEEE 754 single precision?



for fractional form $E=0$, $M \neq 0$.

$$(-1)^S (0.M) \times 2^{-126}$$

for (+ve) $S=0$, (-ve) $S=1$, so other than M everything fix.

$M = \underline{000\cdots1}$ (To represent smallest no. > 0)

$0.M = 0.000\cdots1$

$$(-1)^0 (0.M)\, 2^{-126} = 2^{-23} \times 2^{-126}$$

$$= \underline{2^{-149}}$$

Smallest (+ve) fraction.

Largest possible fraction

$$(-1)^S \times (0.111\cdots1) \times 2^{-126}$$

$$= (2^{-1} + 2^{-2} + 2^{-3} - + 2^{-23})\, 2^{-126}$$

$$= \frac{1/2\,(1-(1/2)^{23}}{(1-1/2)}$$

$$= \underline{(1-2^{-23}) \times 2^{-126}}$$

→ Done

Imp. Points regarding the Number System