Parity → no. of 1's are present in the given code.

There are 2 types of Parity

Odd Parity → If the no. of ones present in the code is odd then this is called odd Parity.

For eg. In a 5 bit code

11010 → eg of odd Parity (3 ones are present)

To make this code as odd parity code, we need to add $P_{od} = 0$

eg. 11011 → To make this code as odd Parity Code, we need to add $P_{od} = 1$ [P_{od} → odd Parity bit]

Even Parity → If the no. of ones present in the code is even then this is called even Parity.

eg. → 1100101 → eg. of even Parity (4 ones)

To make this code as even parity, we need to add even parity bit

$P_{ev} = 0$

eg 11010 → To make this code as even parity code, we need to add $P_{ev} = 1$

Use of Parity bit generator & Parity Checker

When digital data is transmitted from transmitter to receiver, it is necessary to know at the receiving end whether the received data is error free.

A simple form of error detection is achieved by adding an extra bit to the transmitted word. This additional bit is known as

Parity bit and it decides whether the data transmitted is free of error or not.

⇒ to detect the error in the digital data at the receiver end we use Parity bit.

Even Parity generator →

Even Parity generator will maintain the data in even no. of ones.

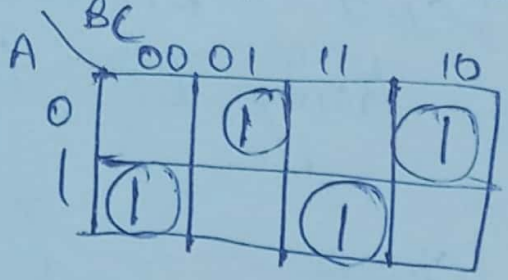
Let we are taken 3 bit message A, B, C

we can draw truth table for 3 bit data

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

In this truth table, Pev generate even Parity for given 8 Possible Combinations

Parity bit, we draw 8 Cell K-Map.



4 singles grouping

$$P_{ev} = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

$$= \bar{A}(\bar{B}C + B\bar{C}) + A(BC + \bar{B}\bar{C})$$

$$= \bar{A}(B \oplus C) + A(\overline{B \oplus C})$$

$$= \bar{A}x + A\bar{x}$$

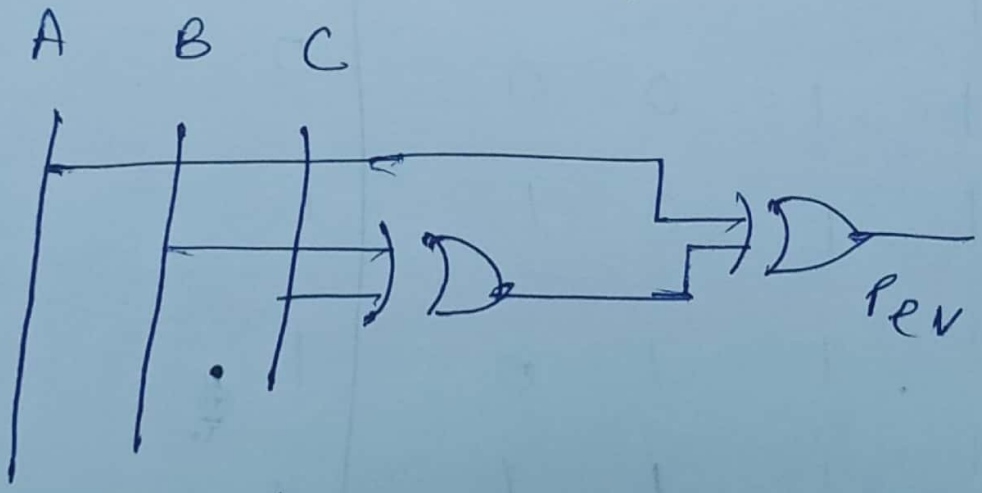
$$= A \oplus x$$

$x = B \oplus C$

$$\boxed{P_{ev} = A \oplus (B \oplus C)}$$

This is the expression for even parity

Logic dig. for even parity generator



X-OR Gate used to generate even Parity.

Even

Parity checker

Here we will send 3 bit data along with even parity bit to the receiver end

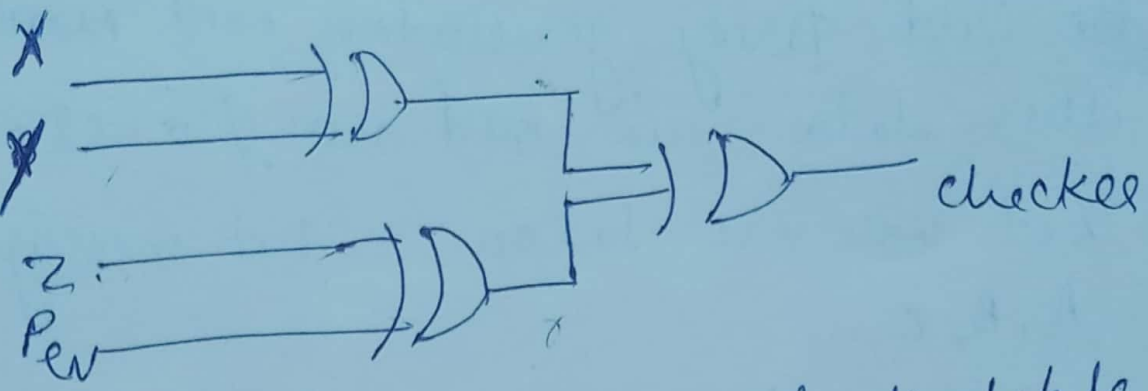
x	y	z	Pev	Checker
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

x, y, z → Received data

if checker = 0
No error

if checker = 1
error

Even
Parity checker ckt →



At is clear from the truth table as even no. of one's are present checker shows 0
& if odd no. of one's are present checker shows 1. This is nothing but the function of X-OR gate.

Odd Parity generator →

Odd Parity generator will maintain the data in odd no. of ones.

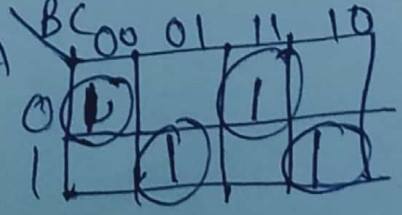
Let we are taken 3 bit message A, B, C

we can draw truth table for 3 bit data

A	B	C	P_{od}
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Here, we ~~will generate~~ P_{od} will generate odd Parity for given possible Combinations.

Now we will draw k-map for the odd Parity bit



Scanned with CamScanner

4 single grouping

$$P_{od} = \bar{A}\bar{B}\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$$

$$= \bar{A}(BC + \bar{B}\bar{C}) + A(\bar{B}C + B\bar{C})$$

$$= \bar{A}(\overline{B \oplus C}) + A(B \oplus C)$$

$$= \bar{A}\bar{x} + Ax \qquad x = B \oplus C$$

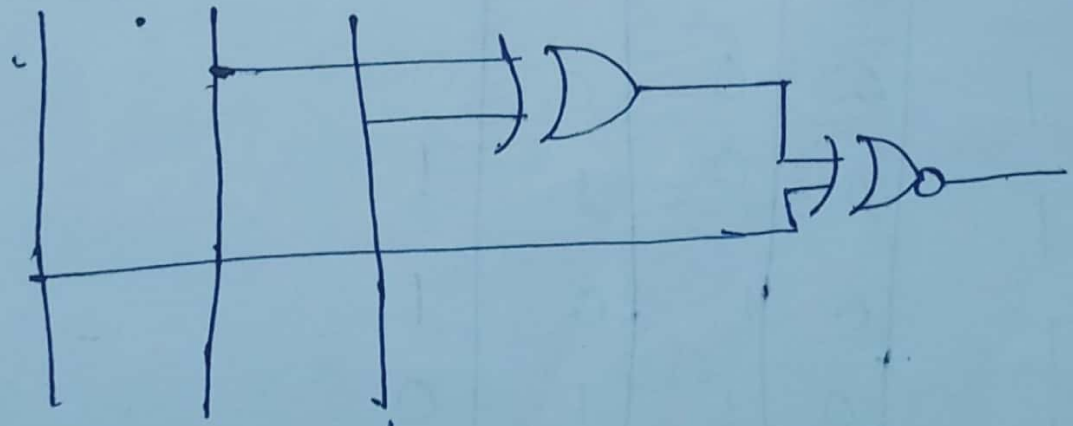
$$= \overline{A \oplus x} \qquad \overline{A \oplus x} = A \odot x$$

$$= A \odot x$$

A X Nor x

$$\boxed{P_{od} = A \odot (B \oplus C)}$$

Logic dig → A B C



Odd Parity generator

Odd Parity Checker

Now we will send 3bit data along with odd parity bit to the receiver end (total 4 bit data)

x	y	z	Pod.	Checker
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

if checker = 0
No error

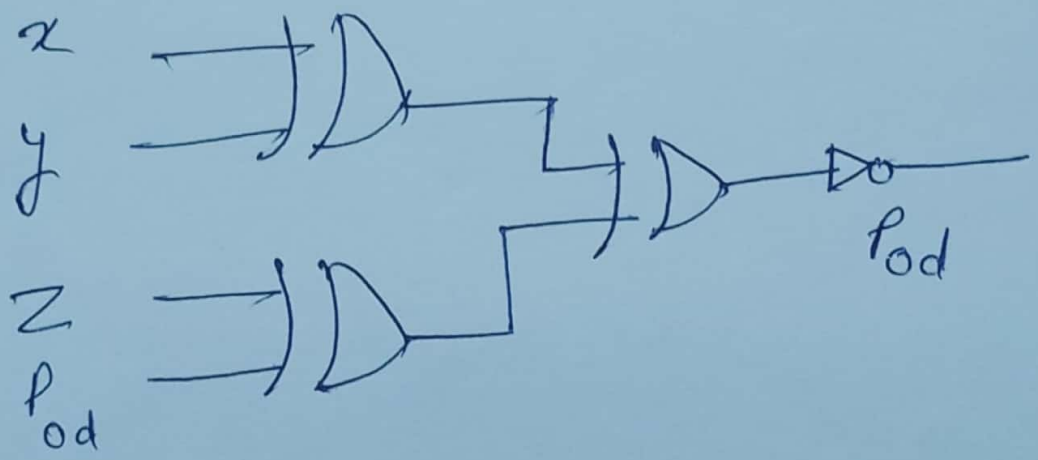
if checker = 1
Error

Odd Parity Checker ckt →

It is clear from the truth table as Odd no. of ones are present checker shows 0 as even no. of ones are Present checker shows 1

This is nothing but the truth table of 0 X-NOR Gate

odd Parity checker ckt →



Complements →

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation.

There are 2 types of Complements for each base r system

r Compliment

- r's Compliment (Radix Comp.)
- (r-1)'s compliment (diminished Compliment)

For binary, r = 2

Therefore 1's, 2's Compliments are possible in binary.

In decimal no. system r = 10

∴ 9's, 10's Compliments are possible.

~~For 2's & 10's~~

For 2's Compliment & 10's Compliment (r's Compliment)

Formula is r's (Compliment)

$$r^n - N$$

N is the number given
r // " base
n - no. of digit in the number given.

Q. Compliment of $(7)_{10}$

$n = 1$, $N = 7$, $r = 10$

$10^1 - 7 = 3$

Q. $N = 5690$

$r^n - N$
$10^4 - 5690$
$10000 - 5690 = 4310$

Q. $N = 1101$ find 2's compliment

$2^4 - 1101$
$16 - 1101$
$10000 - 1101$

Date __/__/__

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

10000 − 1101

= 11 (16 − 13 = 3)

H.W. N = 76895 10th Compliment

N = 11011 2's Compliment

For (r−1)'s Compliment

Formula

$(r^n - 1) - N$

N - number given

n - no. of digits in the number

r - base

4's Compliment of 0011 is 1100 ✓

by formula ⟶

$(2^4 - 1) - 0011$

15 − 0011 ⇒ 1111 − 0011

= 1100 ✓

M T W T F S S
Date __/__/__

(4)

9's Complement of 1234

I st Method →

substract the given no. from 9999

```
 9999
 1234
------
 8765
------
```

II Method →

$$(r^n - 1) - N$$

$$(10^4 - 1) - 1234$$

$$9999 - 1234$$

$$= 8765$$

ex → substract 1101 − 1001

by 2's Complement Method

1's Complement of 1001

is 0110

Add 1

```
------
 0111
------
```

Date __/__/__

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

If ~~there is~~ left most digit is 1 ⇒ Answer is +ve and in that case the left most digit dropped and the answer is read from the remaining digit

⇒ Answer is 0 1 0 0

[1101 → 13
  1001 → 9
  100 → 4]

ex 0 1 0 0
− 0 1 1 1

1's Compliment of 0 1 1 1 is 1 0 0 0

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

If there is 0 in the left most digit then answer will be negative (-ve). Zero is dropped

and remainder is converted to its 2's Compliment.

is Compliment of 1100 is 0011
2's Compliment 0011
+ 1
1100

eg. 0100
-0111

1's Comp. of 0111 is 1000
2's Comp. → Add 1
1001

0100
+1001
1101

if there is no extra digit ⇒ 0 in the left most digit ⇒ Answer is -ve. In that case, zero is dropped. ~~and Retu~~ to Convert 1101 into +ive no. we take 2's Comp of 1101

1101
1's Comp. 0010
2's Comp. + 1
0011

Again there is no extra digit (Carry) : so we take the answer as -0011.

example → substract 72532 - 13250
First find 10's compliment of 13250

99999
13250
86749
+1
86750

Now 72532
+86750
Sum 159282

there is carry of 1 ⇒ Ans is +ve and we can ignore it.

So 72532 - 13250 = 59282

Now substract 13250 − 72532

First find 10's Compliment of 72532

```
  99999
- 72532
-------
  27467
     +1
-------
  27468
```

Now 13250
27468

Sum 40718

there is no end carry
⇒ answer is -ve and then take 10's Compliment of 40718

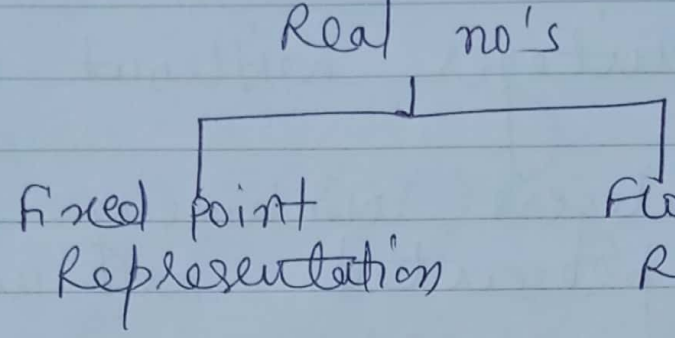
```
  99999
  40718
-------
  59281
     +1
-------
  59282
```

⇒ there is no end carry ⇒ Ans is −59282

M T W T F S S
Date __/__/__

Fixed Point Representation

In order to represent real no's. (Integer part + fractional part) in modern computers, there are 2 approaches



Fixed point representation → It means there is a fixed no. of digits after a decimal point. This implies binary point should be fixed.

Fixed point representation is of 2 types

(1) unsigned fixed point number.

(2) signed fixed point number.

signed fixed point number can be represented in 3 ways.

a) Signed - magnitude representation
b) signed 1's Compliment "
c) " 2's " "

Unsigned Fixed Point numbers →

It is an integer without a sign.

Positive integers including zero, can be represented as unsigned numbers.

Its range is between 0 & +ve infinity

Most significant bit of an unsigned no. is always zero. For eg. for 8 bit integer it should follow the range 0 to 255.

eg. : store 7 in an 8 bit memory location?

First step → Convert 7 into binary

111

Add five 0's to make total of N(8) bits 00000111.

zero means +ve integer

M T W T F S S
Date 1 / / /

eg. 2 store 258 into a 16 bit memory location.

⇒ change 258 into binary

2	258	
2	129	0
2	64	1
2	32	0
2	16	0
2	8	0
2	4	0
2	2	0
	1	0

1.000 0001.0

Add seven 0's to make total N(16) bits

0000000 1000000 10

So 0000000 1000000 10 will be stored in 16 bit memory location.

eg. of storing unsigned integer in 2 diff locations

Decimal	8 bit location	16 bit location
7	00000 111	0000000000000111
234	1.1101.010	00000000 11101010
258	overflow	0000000 1000000 10
24, 760	overflow	0.1.10 0000 1011 1000
1, 24, 5878	overflow	overflow

eg. Represent fixed point representation of unsigned binary no. 0110110 using four integer bits & 3 fractional bits.

no. can be divided into 3 parts [sign part | Integer part | fraction part]

0 110 110

↓

sign bit

Integer part is always included with the sign bit.

0110 . 110

Now Convert $(0110.110)_2$ into Decimal

$$0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0 + 1\times 2^{-1} + 1\times 2^{-2} + 0\times 2^{-3}$$

$$0 + 4 + 2 + 0 + 0.5 + 0.25 + 0$$

$$= (6.75)_{10}$$

Signed fixed point no. ⟶

When an integer binary no. is +ve the sign is 0 & if it is -ve, the sign will be 1 and the rest of the number may be represented in any of the 3 ways :-

a) Signed - Magnitude representation →
The signed - magnitude representation of a negative number consists of magnitude and a negative sign.
eg.
+14 to be stored in 8 bit register

(1) Change 14 to binary 1110
Now to make it of total N(8) bit add four 0's
00001110

Now
-14 to be stored in 8 bit register

10001110
↓
First Most significant bit will be 1 that represent number is -ve.

M T W T F S S
Date __/__/__

15

The signed magnitude system is used in ordinary arthimetic but it is awkward when employed in computer arthimetic.

(1½)

b) Signed - 1's Compliment Representation

In this method 1's Compliment of the positive value has to be taken.

eg (-14) → Convert +14 into binary

00001110

Now 1's Compliment of 00001110 is

11110001

so 11110001 will be stored in 8 bit register

The 1's Compliment imposes difficulties & seldomly used for arthimetic operations.

c) Signed - 2's Compliment Representation

→ In this method 2's Compliment of the +ive value has to be taken.

eg (-14) → 00001110

No with 1's Compliment 11110001

2's Compliment +1

11110010

1111.0010 will be stored in 8 bit Memory location.

eg Represent $(-7.5)_{10}$ using 8-bit binary representation with 4 integer digit & 4 fraction bit

$(7.5)_{10} = (111.1)_2$

0.5 x 2 = 1.0 1
0.0

$(0111.1000)_2$

Now find 2's Comp.

1's Comp 1000.0111
2's Comp 1
1000.1000

$(-7.5)_{10} = (1000.1000)_2$

~~1000.1000~~

eg. Compute 0.75 + (-0.625) using fixed point numbers

Step 1 0.75 Convert into binary

0000.1100

0.75 x 2 = 1.5 1
0.5 x 2 = 1.0 1
0.0 x 2 = 0 0

Now step 2

0.625 Convert into binary

0000.1010

Now 2's Compliment for (-0.625)

1111.0110

Step-3

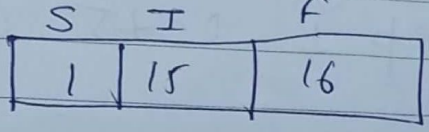
Now 0.75 + (-0.625)

```
  0000.1100
+ 1111.0110
-----------
| 0000.0010
-----------
```

~~exto~~ there is Carry in the MSB. left side → no. is +ve & it will be discarded.

ex. Assume no. using 32 bit format which reserve 1 bit for sign 15 bits for integer & 16 bits for fractional Part

no. is -43.625



Now step 1 43.625 in binary form

Date __/__/__

101011.1010

2	43	
2	21	1
2	10	1
2	5	0
2	2	1
	1	0

00000000 00101011.10100000000 0000

Just a representation

Representation

1 0000000 00 101011.10100000000 0000

0.625 x 2 = 1.25	1
0.25 x 2 = 0.5	0
0.5 x 2 = 1.0	1
0	0

exact value of -43.625 can be obtained by find 2's comp.

00000000 00.101011.10100000000.0000

[11111111 11010100.01100000000.0000]

eg. Convert decimal no. 7.75 to fixed point binary no.

7.75

step 1 Convert 7.75 into binary form

111.110

e.g. The following binary no are stored using 2's Compliment in a 12 bit register with 4 bits after binary point. Convert it into decimal fraction

0 1111 11 1·1 111

sign & Integer | Fraction

− 128	64	32	16	8	4	2	1	0·5	0·25	0·125	0·0625
0	1	1	1	1	1	1	1	1	1	1	1

→ last no. represents sign bit
– put at MSB

0 + 64 + 32 + 16 + 8 + 4 + 2 + 1 + 0·5 + 0·25 + 0·125 + 0·0625

⇒ 127·9375

M T W T F S S
Date __/__/__

2) Convert into 1111 1111.111

sign & integer | fraction

-128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625
1	1	1	1	1	1	1	1	1	1	1	1

$-128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 + 0.5 + 0.25 + 0.125 + 0.0625$

$\Rightarrow -0.0625$

eg 0000 0111.0010

sign & integer | fraction

-128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625
0	0	0	0	0	1	1	1	0	0	1	0

$4 + 2 + 1 + 0.125 \Rightarrow 7.125$

Q using 2's Compliment, Convert the following decimal no. into fixed Point binary no. to be stored in 16 bit register with 4 bits after binary point.

ex: 7.5

111.1000.

0000000000111.1000

ex.

-45.75

45.75 ⇒ 101101.1100

(0000000101101.1100)

Now

-(45.75) → 2's Comp. of 0000000101101.1100

$-(45.75)_{10} = (1111110100.10.0100)_2$

M T W T F S S
Date __/__/__

Floating Point ~~expres~~ Representation →

The floating point representation of a number has 3 parts.

a) Mantissa (M)
b) Base (B)
c) Exponent (E)

In floating point representation binary point floats to the right of the MSB and an exponent is used.

The fixed point mantissa may be a fraction or an integer.
The value of the exponent indicates that the actual position of the decimal point.

The scientific representation of a floating point no. is of the form

$$\pm M \times B^{E}$$

only the mantissa & exponent are physically represented in the register (including their signs).

(21)

the radix B and radix point position of the mantissa are always assumed.

Floating point representation is used to represent very small as well as very large no.

eg 0.0000657

6.57×10^{-5}

M B E

67000

6.7×10^{5}

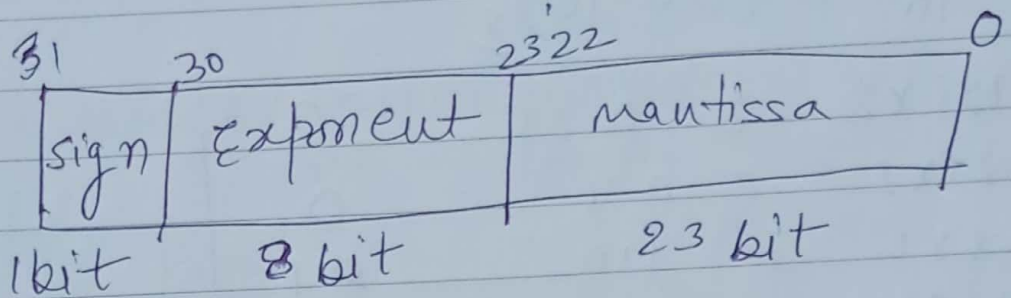
M B E

eg.

Number	Mantissa	Base	Exponent
9×10^{8}	9	10	8
110×2^{7}	110	2	7
4364.784	4364784	10	-3

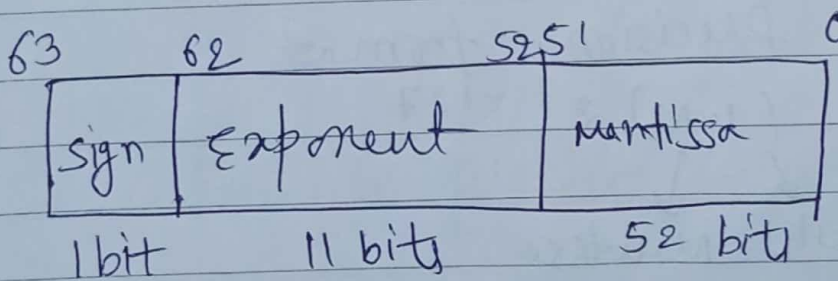
IEEE 754 floating point no representation →

It can be represented in 2 ways
a) Single precision formation →
It can represents no's upto 32 bit



if sign bit is zero → +ve no.
" " " is one → -ve no.

b) Double precision formation →
It can represents no's upto 64 bits



eg:→ Represent $(1259.125)_{10}$ in single & double precision format.

step-1 Convert $(1259.125)_{10}$ to $(??)_2$

$(1259)_{10} \rightarrow (10011101011)_2$

$(0.125)_{10} = (0010)_2$

$0.125 \times 2 = 0.25 \quad 0$

$0.25 \times 2 = 0.5 \quad 0$

$0.5 \times 2 = 1.0 \quad 1$

$0.0 \times 2 = 0 \quad 0$

$(1259.125)_{10} = (10011101011.0010)_2$

step-2 → Normalize the number

For single precision format

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

sign bit — Mantissa

127 – Bias no. for 32 bit

For double precision format

$(1.N) 2^{E-1023}$

Now $(1001.1101011.0010)_2$

$1.001111010111 \times 2^{10}$

step-3 single precision format

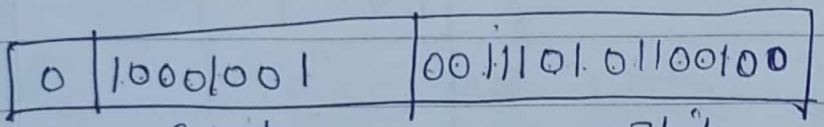
$[1.N]\ 2^{E-127}$

$1.00111010110010 \times 2^{10}$

$\Rightarrow E-127 = 10$

$\Rightarrow E = 137 = (10001001)_2$

2	137	
2	68	1
2	34	0
2	17	0
2	8	1
2	4	0
2	2	0
	1	0



1bit 8bits 23bits

step-4 Double precision format

$[1.N]\ 2^{E-1023}$

$1.00111010110010 \times 2^{10}$

$E - 1023 = 10$

$E = 1033$

$(1033)_{10} = (1000000\,1001.)_2$

0	1000 000 100 1	00111010110010...

Q. Represent IEEE standard for given no

$1.00010100 \times 2^{-10}$

31	30	23 22	0
0	0111010	00010100...000	

23 bit

$E - 127 = -10$

$E = 117$

$= (1110101)_2$

$(1.N) \times 2^E$

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

Date __/__/__

Represent in floating pt representation

1.110111.10010

1.110111 10010 $\times 2^6$

0	110	1101111.10010

Normalize floating point no →

In Computer, Real no's are represented in the form of Normalize floating point no.

A no. is said to be normalised if

① $0.1 \leq$ mantissa < 1 (The integer part should be zero.)

② In a given decimal no. before decimal point, it should always be zero. Immediately after decimal Point no. should be non-zero.

12.56 0.1256×10^2

123.46×10^{-6} 0.12346×10^{-3}

0.00246×10^{-3}

0.246×10^{-5}

00011010 8bit binary no. is not normalized

To make it normalize

0.11010×2^{-3}

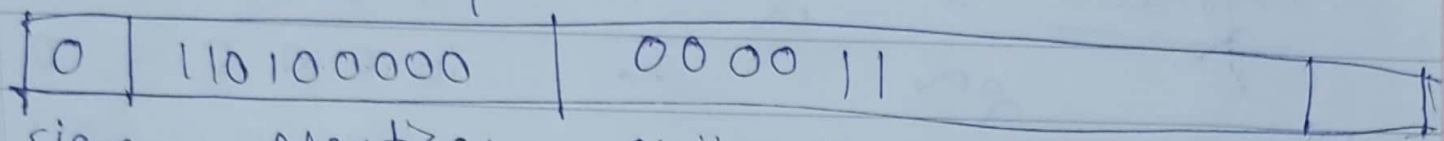
Mantissa — 11010000

Normalized numbers provide the maximum possible precision for the floating-point no.

eg. Convert the following floating point binary number into decimal in a single precision format.

① 0 1101.0000 0000 0000 11

Assume 9 bits as Mantissa & 6 bits exponent



Exponent →

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

$= 3$

Exponent $= 2^3$

~~0 1101.0000 0000 0000~~

~~0.1101.000 00 × 2^3~~

$0.1101000\,00 \times 2^3$

0 110. 100000

-8	4	2	1	0.5	0.25	0.125	0.0625
0	1	1	0	1	0	0	0

$4 + 2 + 0.5 = 6.5$

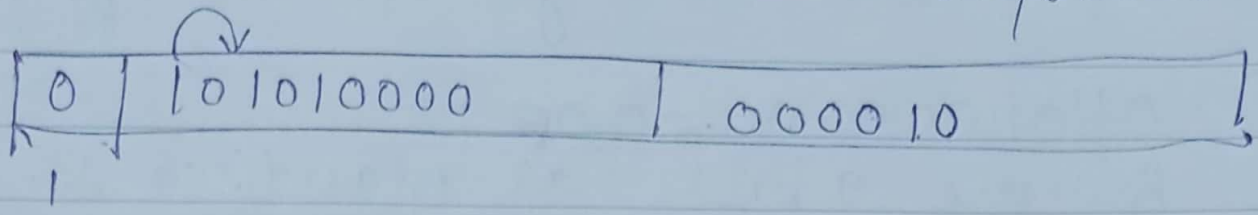
$(0110100000000011)_2 = (6.5)_{10}$

Date __/__/__

34

eg: 0 1.0.1 0 1000 00000 10

9 bits mantissa
6 bits exponent



exponent → 2^2

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
0 0 0 0 1 0

0 10 . 10 10000 $\times 2^2$

4 2 1 0.5 0.25 0.125

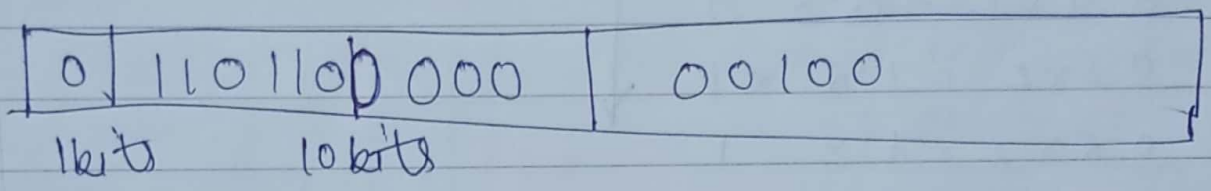
0 + 2 + 0 | + 0.5 + 0 + 0.125 + 0

~~2.75~~ $(2.625)_{10}$

M T W T F S S
Date __/__/__

eg: Convert the following floating point binary no. into decimal. Assume 10 bits for mantissa & 5 bit for exponent both in 2's compliment

0 1101 1 0000000100



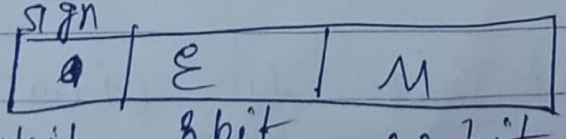
2^4

0 1101 | 1.000000 × 2^4

1 + 4 + 8 + 0.5 = $(13.5)_{10}$

eg. How to represent a number in IEEE 754 32 bit floating pt no.

263.3



Date __/__/__

(34)

$\Rightarrow (263)_{10} = (\quad)_2 = 100000111$

$\Rightarrow (0.3)_{10} = (\quad)_2 = 0.0100110011\ldots$

$0.3 \times 2 = 0.6 \quad 0$
$0.6 \times 2 = 1.2 \quad 1$
$0.2 \times 2 = 0.4 \quad 0$
$0.4 \times 2 = 0.8 \quad 0$
$0.8 \times 2 = 1.6 \quad 1$
$0.6 \times 2 = 1.2 \quad 1$
$0.2 \times 2 = 0.4 \quad 0$
$0.4 \times 2 = 0.8 \quad 0$
$0.8 \times 2 = 1.6 \quad 1$
$0.6 \times 2 = 1.2 \quad 1$
$0.2 \times 2 = 0.4 \quad 0$

Now step -1

$(263.3)_{10} = (100000111.0100110011)$

step -2 $1.00000111\,0100110011 \times 2^8$

step-3 $[1.N]\,2^{E-127}$ standard form

Exponent is in binary form.

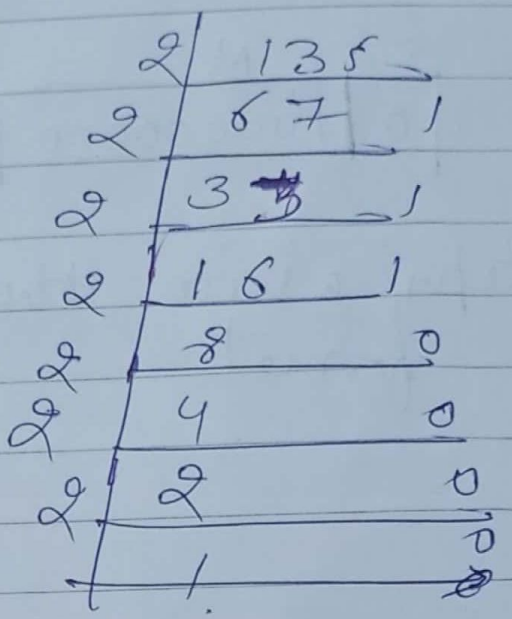
(5)

Compare $(1.N)2^{E-127} = 2^8$

$E - 127 = 8$

$E = 135$

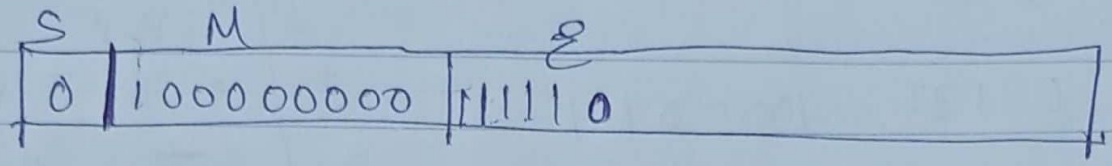
$E = 135 = 10000111$



Now

0	10000111	00001110011001100110

eg. Convert 0 1000000000 111 110
1 as sign bit, 9 bit as integer &
6 bit as Exponent



Step-1 write the binary no. in a proper format

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

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

$= -2$

$\Rightarrow$ Exponent is -2

So 2^{-2}

$\Rightarrow 0.1000000000 \times 2^{-2}$

Step-2

Now Put the decimal point according to the exponent.

so actual no. is

0.00 100000000

2	1	0.5	0.25	0.125
0	0	0	0	1

$= 0.125$

~~$\Rightarrow 0.100$~~

$\Rightarrow (0\,1000\,000\,00\,111110)_2 = (0.125)_{10}$

Gray Code → The table below gives gray code with their equivalent decimal & binary code.

Decimal no.	Gray Code	Binary
0	0000	0000
1	0001	0001
2	0011	0010
3	0010	0011
4	0110	0100
5	0111	0101
6	0101	0110
7	0100	0111
8	1100	1000
9	1101	1001
10	1111	1010
11	1110	1011
12	1010	1100
13	1011	1101
14	1001	1110
15	1000	1111

Comparision between Gray Code & Binary Code →

The Comparision between Gray Codes and binary code is as given

in the above table. If we take any 2 Consecutive decimal no's in gray Code then we see that there is a change in only one bit position. Now we Consider the binary Corresponding decimal no. 3 & 4. We notice that there is a change in 3 bit position. This greately enhance the chances of error whereas in the gray Code, since only one bit position changes between decimal no's 3 & 4 the chances of error reduced. Again the another advantage is it takes a finite time for changes in bit position. Therefore Gray Code circuitry Can operate at higher speed.

Disadvantages → The disadvantage with the Gray Code is that it Cannot be used in arthimatic operation. For arthimetic operation Gray Code must be Change to binary form.

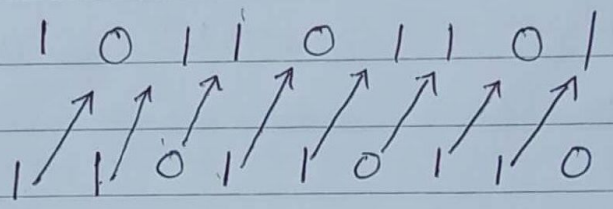
Date __/__/__

Gray Code to Binary Conversion →

eg. Convert 1011011101 into binary Conversion

step - 1 Gray Code 1011011101

First digit remain as it is and next bits will be obtained by cross X-OR ⊕ operation.



So Binary no. is 1101101110

Binary to Gray Code Conversion →

eg. Convert 11001110 into Gray code.

Binary no. 11001110

step - 1 First bit remain as it is and for second bit onwards we have to do

Date __/__/__

40

x-OR operations.

1 1 0 0 1 1 1 0

1 0 1 0 1 0 0 1

So Gray Code is 1 0 1 0 1 0 0 1

0 0 1 1

↓

0 0 1

Excess - 3 Code →

The excess - 3 code is a non-weighted code used to express ~~code~~ decimal numbers. It is a self complimentary binary coded decimal (BCD) code and of numerical system which has biased representation. It is particularly significant for arthimetic operations as it overcomes short coming encountered while using 8421 BCD code to add 2 decimal digits whose sums exceeds 9.

Representation →
It can be obtained by adding 3 to each decimal digit

'Binary to Excess-3 Code Conversion →

Step 1 - Find the decimal equivalent of the given binary no.

Step 2 - Add 3 to each digit of decimal number

Step 3 - Convert the newly obtained decimal no. back to binary no.

to get required excess-3 equivalent.

eg. A Find the excell 3 of 0100

step-1 Decimal equivalent of 0100 is 4.

step-2 Add 3 in that

$$\begin{array}{r} 4 \\ +\ 3 \\ \hline 7 \end{array}$$

step-3 Binary Conversion of 7 is 0111

Table-1 shows the excess-3 codes for decimal digits

Decimal	BCD	excess-3 code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

eg. Convert 23 to X-3 Code

Step - 1 Add 3 in each of the decimal no.

```
  23
+ 33
----
  56
```

step - Convert 56 into ~~binary~~ BCD

01010110

eg. Convert 15.46 into excess 3

step-1 Add 3 into both digit in the decimal no.

```
  15.  46
  33.  33
-------------
  48.  79
-------------
```

step - 2 Convert 48.79 into BCD

0100 1000 . 0111 1001

6. Excess-3 to BCD and decimal number. →

excess-3

eg. 1001001 Convert it into BCD & decimal no.

1001 001

Make a grouping of 4 bit for each group

i.e.

0100 1001

Now subtract 3(0011) from each four bit group that will be BCD

0100 1001
0011 0011
0001 0110

Now Convert this 0001 0110 into decimal

$(0001\ 0110)_{BCD} = (16)_{10}$

Excess - 3 Addition →

1. Take excess-3 code for decimal numbers
2. Add excess-3 codes
3. If Carry is there then add 3 otherwise subtract 3.

eg. Add 4 & 8

step-1 take excess 3 for 4

$$\begin{array}{r} 4 \\ +3 \\ \hline 7 \end{array} \quad = \quad 0111$$

for 8 excess-3 code is

$$\begin{array}{r} 8 \\ 3 \\ \hline 11 \end{array}$$

1011

Now

$$\begin{array}{r} 4 \\ +8 \\ \hline \end{array} \qquad \begin{array}{r} 0111 \\ 1011 \\ \hline 10010 \end{array}$$

there is a Carry ⇒ Add 3

$$\begin{array}{r} 0001 \quad 0010 \\ 0011 \quad 0011 \\ \hline 0100 \quad 0101 \end{array}$$

cross check

0100 0101

↓4 ↓5 → in excess 3

1 2 in decimal

$$\begin{array}{r} 4 \\ +8 \\ \hline 12 \end{array}$$

Advantages of Excess-3 Code →

a) These are unweighted BCD.

b) These are self complimentary codes.

c) It is particularly significant for arithmetic operations as it overcomes short coming encountered while using 8421 BCD Code to add 2 decimal digits whose sums exceeds 9.

M T W T F S S
Date __/__/__

ASCII CODE →

ASCII Stands for American standard Code for information interchange. It is the standard Code Commonly used for the transmission of binary information. Each character is represented by a 7-bit Code and usually an eighth bit is inserted for Parity. The Code Consists of 128 characters. Ninety-five characters represent graphic symbols that include upper & lower case letters, numerals 0 to 9, punctuation marks, and special symbols. 23 characters represent format effectors, which are functional characters for Controlling the layout of printing or display devices such as Carriage return, line feed, horizontal tabulation, and back space. The other 10 characters are used to direct the data Communication flow and report its Status.

Some important ASCII Codes are given below →

Table-1

32 - space	42 - *	61 - =
33 - !	43 - +	62 - >
34 - "	44 - ,	63 - ?
35 - #	45 - -	64 - @
36 - $	46 - .	65-90 - (A-Z) (Cap)
37 - %	47 - /	91 - [
38 - &	48-57 - (0-9)	92 - \
39 - '	58 - :	93 -]
40 - (	59 - ;	94 - ^
41 -)	60 - <	95 - _

96 - ` (Grave accent)
97 - 122 (a - z) (small)
123 - {
124 - | (vertical line or stripe line)
125 - }
126 - ~

Date __/__/__

eg. $(100\,1001\,100000\,1100101)_2$ Convert into ASCII

step-1 Convert Binary into decimal then decimal to ASCII

step-2 Make a group of 7 bit

64 32 16 8 4 2 1

1001001 1000001 1001101

step-3 Convert into decimal

73 65 77

Convert these decimal into ASCII with the help of Table 1.

I AM

I AM = ~~1000~~ $(1001001\,1000001\,1001101)_{10}$