Digital Logic Design

UNIT- I

1. Digital Systems:-

A Digital system is a system in which signals have a finite number of discrete values.
Examples of discrete sets are 10 decimal digits, 26 letters etc.,

Signals:- Discrete elements of information are represented in a digital system by physical quantities called signals.

The signals in most present day electronic digital systems use just two discrete values and they are said to be binary.

A binary digit called a bit has two values either "0 or 1".

Discrete elements of information are represented with groups of bits called "Binary codes".

Thus a digital system is a system that manipulates discrete elements of information represented internally in binary form.

For ex; the decimal digit 0 through 9 are represented in a digital system with a code of four bits. The number 7 is represented by 0111.

How a pattern of bits is interpreted as a number depends on the code system in which it resides.

↳ $(0111)_2$ to indicate the pattern is to be in binary system.

↳ $(0111)_{10}$ to indicate the pattern is to be in decimal system.

The users are allowed to specify and change the program or the data according to the specific need. Because of this flexibility,

General purpose digital computers can perform a variety of information processing tasks that range over a wide spectrum of appln's.

Ex:- (1) A payroll schedule is an discrete process, which has discrete data values such as letters.

On the other hand,

(2) A Research scholar may observe a continuous process, but record only specific quantities in tabular form.

In many cases, the quantization of a process can be performed automatically by an Analog-to-digital converter.

Analog-to-Digital Converter:-

A Device that forms a digital (discrete) representation of a Analog (continuous) quantity.

The general purpose digital computer is the best known ex., of digital system. The major parts of computer are memory unit, central processing unit, Input & Output.

A digital computer is a powerful instrument that can perform not only arithmetic computations, but also logical operations.

↳ **Advantages of Digital Systems:-**

(i) Ease of programmability.
(ii) Reduction in cost of hardware
(iii) High speed
(iv) High Reliability
(v) Design is easy
(vi) Results can be reproduced easily.

1.1 NUMBER SYSTEM:-

It defines a set of values used to rep., quantity. It is a language of digital system consisting of set of symbols called digits with rules defined for their addition, multiplⁿ and other mathematical Operations. There a 2 types

(i) Positional Numbers system
(ii) Non positional Number system.

↳ **Positional Number System:-**

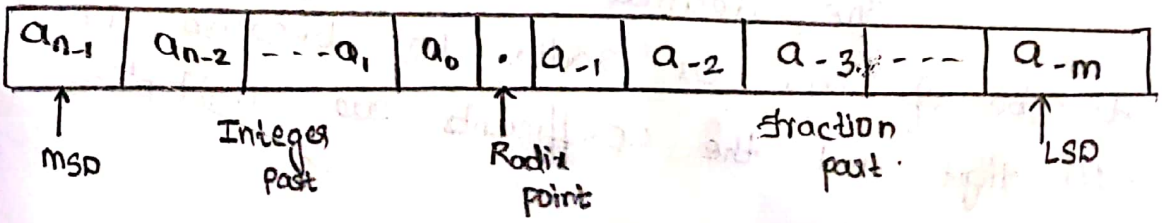
The position of each digit of a number has some positional weight. It is most widely used. Ex:- Decimal numbers.

↳ **Non-Positional:-** In this a digit of a number does not indicate any significance in position and weight. Ex:- Roman numericals. It is very difficult to use because zero is not present.

A number is constructed by a collection of digits and it has Integer and Fraction, both are seperated by a Radix point.

General format of a Number:-

$$N_r = a_{n-1} r^{n-1} + a_{n-2} r^{n-2} + \cdots + a_1 r^1 + a_0 r^0 + a_{-1} r^{-1} + \cdots + a_{-m} r^{-m}$$



N_r → Number with base r

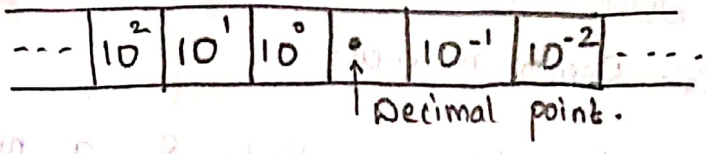
r → radix or base

a → integer in the range $0 \leq a_i \leq (r-1)$.

Digital Logic Design.

1.2 Decimal Number System:-

Decimal number system has 10 symbols The symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. So the radix or base of this number system is "10". The position of each digit in a decimal number indicates with the magnitude of the quantity represented and can be assigned a weight. Hence it is called Positional Weight Numbes System.



In the decimal number system we can express any decimal number in units, tens, etc.,

Decimal Number:-

7392, it can be written as

$$7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

The Thousands, hundreds etc, are powees of 10 implied by position of the co-efficients.

In general, A number with a decimal point is rep, by a series of coefficients as follows

$$a_5\ a_4\ a_3\ a_2\ a_1\ a_0 \cdot a_{-1}\ a_{-2}\ a_{-3}$$

The decimal number system is said to be of base or radix 10 because it uses 10 digit and the co-efficients are multiplied by powees of 10.

Ex:- $10^5 a_5 + 10^4 a_4 + 10^3 a_3 + 10^2 a_2 + 10^1 a_1 + 10^0 a_0 + 10^{-1} a_{-1} + 10^{-2} a_{-2} + 10^{-3} a_{-3}$.

2. BINARY NUMBERS :-

The coefficients of binary number system have only two values "0 & 1".

bit (0 or 1)
↓
4 bits (Nibble)
↓
8 bits (Byte)
↓
16 bits (word)
↓
32 bit (Double word)

0+0 = 0
0+1 = 1
1+1 = 10

1 1 1 1
1
100

For ex., the decimal equivalent of the binary number 11010.11 is 26.75

Multiplication of the co-eff by powers of 2.

$1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0 + 1\times2^{-1} + 1\times2^{-2} = 26.75$

In general, a number expressed in a base - r system has coeff multiplied by powers of r.

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \cdots + a_2 \cdot r^2 + a_1 \cdot r + a_0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \cdots + a_{-m} \cdot r^{-m}$$

To distinguish between numbers of different bases we enclose the coefficients in parenthesis and write a subscript equal to the base used.

Ex:- Base - 5 number $(4021.2)_5$

$$= 4\times5^3 + 0\times5^2 + 2\times5^1 + 1\times5^0 + 2\times5^{-1} = (511.4)_{10}$$

The coeff values for base '5' can be only 0, 1, 2, 3 & 4.

3. OCTAL NUMBER:-

The Octal number system is a base-8 system that has eight digits i.e., 0, 1, 2, 3, 4, 5, 6, 7.

Ex:- $(127.4)_8 = 1\times8^2 + 2\times8^1 + 7\times8^0 + 4\times8^{-1} = (87.5)_{10}$

Note:- The digits 8 & 9 cannot appear in an Octal number. By adding each digit of an octal number in a power of 8 we can find the decimal equivalent of octal number.

4. HEXA DECIMAL NUMBER:-

It is a base 16 number system. The first ten digits are borrowed from the decimal system. i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. It is another number system that is particularly useful for human communications with a computer.

Ex:- $(B65F)_{16} = 11\times16^3 + 6\times16^2 + 5\times16^1 + 15\times16^0 = (46,687)_{10}$

The digits in a binary number are called bits. When a bit is equal to 0, it does not contribute to the sum during conversion.

Therefore the conversion from binary to decimal can be obtained by adding only the numbers with powers of two corresponding to the bits that are equal to 1.

Ex:- $(110101)_2 = 32+16+4+1 = (53)_{10}$

↳ 2^{10} is referred as Kilo (K)
↳ 2^{20} is referred as Mega (M)
↳ 2^{30} is referred as Giga (G)
↳ 2^{40} is referred as tera (T)

1KB = 1024 byte
1MB = 1024 KB
1GB = 1024 MB
1TB = 1024 GB

∴ So $4K = 2^{12} = 4096$
$16M = 2^{24} = 16,777,216.$

↳ A computer hard disk with four gigabytes of storage has a capacity of $4G = 2^{32}$ bytes (approx., 4 billion bytes)
↳ A terabyte is 1024 gigabytes (approx, 1 trillion bytes)

Extra

Examples of Binary numbers:-

(i) Addition

augend: 101101
addend: + 100111
sum: 1010100

(ii) Subtraction

minuend: 101101
subtrahend - 100111
difference: 000110

(iii) multiplication.

multiplicand: 1011
multiplier × 101
1011
partial product → 0000
10111
product: 110111

Number System	Base	First digit	Last digit	All digits / characters.
Binary	2	0	1	0, 1
Octal	8	0	7	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0	9	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadecimal	16	0	F	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Ex:- Relat^n b/w Binary decimal etc.,

Decimal	Binary	Octal	Hexadecimal.
0	0000	0	0
4	0100	4	4
8	1000	10	8
11	1011	13	B
15	1111	17	F

NUMBER BASE CONVERSIONS:-

The human beings use decimal number system while computer uses binary number system. Therefore it is necessary to convert decimal number into its equivalent binary number while feeding to computer and again convert to decimal ~~which~~ while displaying result.

- Binary to octal
- Octal to Binary
- Binary to Hexadecimal
- Hexadecimal to Binary
- Octal to Hexadecimal
- Hexadecimal to octal.
- Decimal to Binary
- Decimal to octal.

(i) convert decimal 41 to binary.

To convert decimal number to a binary, divide decimal number by 2 successively & its remindes form binary number & the final remainder becomes the most significant digit.

Ex:- $(41)_{10}$ =

2	41	
2	20	- 1
2	10	- 0
2	5	- 0
2	2	- 1
2	1	- 0
	0	- 1

$\therefore (41)_{10} = (101001)_2$
$a_5\ a_4\ a_3\ a_2\ a_1\ a_0$

If decimal number contains Fraction part, then it requires successive multiplication of the fractional part by 2, with the integer part after each multiplication becomes the next lower digit in the binary fraction.

For ex; $(41.375)_{10}$

	Integer part
$0.375 \times 2 = 0.75$	0
$0.75 \times 2 = 1.5$	1
$0.5 \times 2 = 1$	1

$\therefore (41.375)_{10} = (101001.011)_2$

(ii) Convert binary to decimal.

To convert binary to decimal, coefficients of binary number are multiplied by powers of 2.

Ex:- $(1001)_2 = 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 = (9)_{10}$

(iii) Convert Binary to Octal

The base for octal number is 8 & the base for binary number is 2. The base for octal number is the third power of the base of binary numbers. So, by grouping 3 digits of binary numbers & then converting each group digit to its octal equivalent we can convert binary number to octal equivalent.

Ex:- $(10101101.0111)_2$ to octal.

(i) Make group of 3 bits starting from LSB for Integer part & MSB for fractional part by adding 0's at the end, if required.

0	1	0	1	0	1	1	0	1	.	0	1	1	1	0	0	Binary
2			5			5			.	3			4			Octal

$\therefore (10101101.0111)_2 = (255.34)_8$

Adding 0's at MSB to make group of 3-bits

Adding 0's at LSB to make group of 3 bits.

(iv) Convert Octal to Binary.

Each digit of the octal number is individually converted to its binary equivalent to get octal to binary conversion of the number.

Ex:- $(125.62)_8$

↳ Remove any leading or trailing zeros.

1	2	5	.	6	2
0 0 1	0 1 0	1 0 1	.	1 1 0	0 1 0

1 0 1 0 1 0 1 . 1 1 0 0 1

$(125.62)_8 = (1010101.11001)_2$.

(v) Convert Binary to Hexadecimal

The base for hexadecimal is the fourth power of the base for binary numbers. Therefore by grouping 4 digits of binary numbers & then converting each group digit to its hexadecimal equivalent we can convert binary number to its hexadecimal equivalent.

Ex:- $(110110110.1001101)_2$

0 0 1 1	0 1 1 0	1 1 1 0	.	1 0 0 1	1 0 1 0
3	6	E	.	9	A

Adding 0's to make group of 4 bits.

$\therefore (110110110.1001101)_2$

$= (36E.9A)_{16}$

(vi) Hexadecimal to Binary Conversion.

Ex:- $(8A9.B4)_{16}$

8	A	9	.	B	4
1 0 0 0	1 0 1 0	1 0 0 1	.	1 0 1 1	0 1 0 0

$\therefore (8A9.B4)_{16} = (100010101001.101101)_2$

(vii) Convert decimal to octal.

Eg:- $(0.513)_{10}$

$0.513 \times 8 = 4.104$

$0.104 \times 8 = 0.832$

$0.832 \times 8 = 6.656$

$0.656 \times 8 = 5.248$

$0.248 \times 8 = 1.984$

$0.984 \times 8 = 7.872$

$\therefore (0.513)_{10} = (0.406517...)_8$

(viii) Convert octal to Hexadecimal.

Step-1:- Convert octal to ~~hexadecimal~~ binary equivalent.

Step-2:- Binary to its hexadecimal equivalent.

Eg:- $(615.25)_8$ to Hexadecimal.

6	1	5	.	2	5
110	001	101	.	010	101
0001	1000	1101	.	0101	0100
1	8	D	.	5	4

$(615.25)_8 = (18D.54)_{16}$

(ix) Convert Hexadecimal to octal.

Convert Hexadecimal to Binary

Convert Binary to octal.

Eg:- $(BC66.AF)_{16}$ to octal

B	C	6	6	.	A	F
1011	1100	0110	0110	.	1010	1111
001	011	110	001	100	110	.	101	011	110
1	3	6	1	4	6	.	5	3	6

** Solve for x

(i) $(367)_8 = (x)_2$ (ii) $(378.93)_{10} = (x)_8$

(iii) $(B9F.AE)_{16} = (x)_8$ (iv) $(16)_{10} = (100)_x$. [∵ Dec-16 6Mks]

(i) $(367)_8 = (x)_2$

3 6 7 octal

011 110 111

$\therefore (367)_8 = (111110111)_2$

(ii) $(378.93)_{10} = (x)_8$

8 | 378 -

8 | 47 - 2

5 - 7 ↑

$(572.734)_8$

$0.93 \times 8 = 7.44 \rightarrow 7$

$0.44 \times 8 = 3.52 \rightarrow 3$

$0.52 \times 8 = 4.16 \rightarrow 4$

$0.16 \times 8 = 1.28 \rightarrow 1$

(iii) B 9 F . A E

1011 1001 1111 . 1010 1100

$(5\ 6\ 3\ 7\ .\ 5\ 3\ 4)_8$

(iv) $(16)_{10} = (100)_x$

$16 = 1 \times x^2 + 0 \times x^1 + 0 \times x^0$

$16 = x^2$

$\rightarrow x = 4$

$\therefore (16)_{10} = (100)_4$

** Convert $(163.875)_{10}$ to Binary, octal Hexadecimal. [Dec-16 4mks]

16 | 163

10 (A) - 3

$\therefore (163)_{16} = (A3)_{16}$

$0.875 \times 16 = 14.0$

$\therefore (A3.E)_{16}$

A 3 . E

1010 0011 . 1110

2 4 3 . 7

$\therefore (163.875)_{10} = (A3.E)_{16} = (10100011.1110)_2$

$= (243.7)_8$

*** Determine the value of 'b' for the following.

(i) $(292)_{10} = (1204)_b$

$\therefore (292)_{10} = 1 \times b^3 + 2 \times b^2 + 0 \times b^1 + 4 \times b^0$

$= b^3 + 2b^2 + 4$

$\therefore b = 6$

(ii) $(16)_{10} = (100)_x$

$1 \times x^2 + 0 \times x^1 + 0 \times x^0$

$\therefore x^2 = 16$

$x = 4$

6. COMPLEMENTS:-

Complements are used in digital computers to simplify the Subtraction Operation and for logical manipulation.

There are 2 types of complements for base number system.

(i) r's compliment (or) Radix complement

(ii) (r-1)'s compliment (or) Diminished Radix complement.

↳ For decimal no's, the complements are termed as
- (i) 10's complement
- (ii) 9's complement

↳ For binary number system
- (i) 2's complement
- (ii) 1's complement.

↳ For octal number system
- (i) 8's complement
- (ii) 7's complement.

↳ For Hexadecimal number system
- (i) 16's complement
- (ii) 15's complement.

	r's comp	(r-1)'s comp
r=10	10's comp	9's comp
r=2	2's comp	1's comp
r=8	8's comp	7's comp
r=16	16's comp	F's comp

∴ r's comp = $r^n - N$

(r-1)'s comp = $r^n - N - 1$

(r-1)'s comp = r's comp - 1

(r-1)'s comp + 1 = r's comp (∵ By adding 1 on both sides)

No borrow — borrow

$r^n - N$

6.1 * r's complement [Radix Complement]

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$.

↳ r's complement is obtained by adding 1 to the (r-1)'s complement.

Ex:- Complement of $(7)_{10}$ [9-N = 9-7 = 2 → 9's comp

$+\frac{2}{3}$ → 9's comp

$+1$ → 10's comp

Sol:- $N = 7$

$n = 1$

$r = 10$

$\therefore r^n - N = 10^1 - 7 = 3$

Ex.2:- $(5690)_{10}$

9999 − 5690 = 4309 + 1 = 4310

$N = 5690$

$r = 10$

$n = 4$

$\therefore 10^4 - 5690 = 10000 - 5690$

$= 4310$

Ex-3:- Find 2's complement of 1101

1's → 0010

2's → 0011 (3)

$r = 2$

$N = 1101$

$n = 4$

$\therefore 2^4 - 1101 = 16 - 1101$

↳ $(16)_{10} - 1101$

↳ 10000 − 1101 (∵ 16 − 13 = 3)

↳ 11

10000 − 1101 = 0011

0010

0011

6.2 (r-1)'s Complement:-

If N is a positive numbers in base 'r' with 'n' integer digits & 'm' fraction digits, the (r-1)'s complement is defined as $[(r^n-1) - N]$ or $[r^n - r^{-m} - N]$

where r = base of the given number system
n = no. of digits [integers]
N = given positive number
m = no. of fraction digits.

(1) Find 9's complement for the given decimal numbers.

(a) 7543

n = 4
r = 10

$= (10^4 - 1) - 7543$
$= 9999 - 7543$
$= 2456$

(b) 0.7863

n = 0
r = 10
m = 4

$= 10^0 - 10^{-4} - 0.7863$
$= 1 - 0.0001 - 0.7863$
$= 0.2136.$

(c) 632.456.

n = 3
r = 10
m = 3

$= 10^3 - 10^{-3} - 632.456$

For integer
n = 3
r = 10
$(10^3 - 1) - 632$
$= (1000-1) - 632$
$= 999 - 632$
$= 367$

For fraction
n = 3
r = 10
m = 3
$= 10^0 - 10^{-3} - 0.456$
$= 1 - 0.001 - 0.456$
$= 0.999 - 0.456$
$= 0.543$

Verification ⟹ 367.543

[999-632].[999-456]

Subtraction with Complements

The subtraction of two n-digit unused numbers M-N in base r can be done as follows.

↳ Add the minuend M to the r's comp of the subtrahend N. i.e., $M + (r^n - N) = M - N + r^n$

↳ If $M \geq N$, the sum will produce an end carry r^n, which can be discarded, what is left is the result M-N

↳ If $M < N$, the sum does not produce an end carry r^n, and is equal to $r^n - (N-M)$, which is the r's complement of $(N-M)$. To obtain the ans, take r's complement of the sum & place a negative sign in front.

Ex:-

Using 10's complement subtract 72532 - 3250

M = 72532

10's com N = 96750

sum = 169282

carry generated

$M \geq N$

Discard end carry = 69282.

Using 10's complement subtract 3250 - 72532

M = 03250

10's comp N = +27468

sum 30718

carry Not Generated

$M < N$

7. SIGNED BINARY NUMBERS :-

Positive numbers including zero can be represented as Unsigned numbers & Negative numbers can be represented as Signed Numbers.

Because of hardware limitations in computers both positive & negative numbers are represented only with binary digits.

The leftmost bit [sign bit] in the numbers represents sign of the numbers.

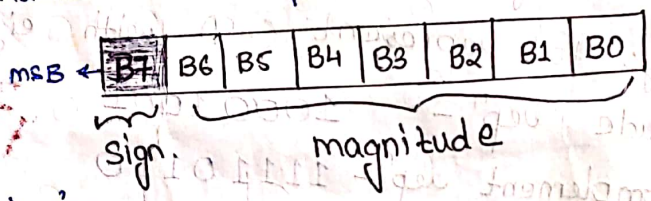
↳ Sign bit is '0' for positive numbers

↳ Signbit is '1' for Negative numbers.

↳ max '+ve' no = +127 [0 111 1111]

↳ max '-ve' no = -128 [1111 1111]

These numbers are represented by the sign Magnitude format.



If MSB=0 then '+ve'

If MSB=1 then '-ve'

+6 = 0000 0110

-14 = 1000 1110

+24 = 0001 1000

-64 = 1100 0000

Examples:-

↳ If string of bits 01001 can be rep^n as 9 [unsigned binary] because left most bit is '0'.

↳ string of bits 11001 can be rep^n as binary equivalent of '25' when considered as unsigned number & binary equivalent of '-9' when considered as '-ve' or signed number.

Usually there is no confusion in interpreting the bits if the type of rep^n for the number is known in advance.

The rep^n of signed numbers above is refered as the "signed-magnitude Convention".

In this notation, the ~~mag~~ number consists of magnitude & a symbol (+ or -) or (0 or 1) indicating sign

This is the repⁿ of signed numbers used in ordinary arithmetic.

Another system mostly used is "signed complement system" for representing '-ve numbers.

In this system negative number is indicated by its complement.

↳ where as signed magnitude system negates a number by changing its sign.

↳ the signed complement system negates a number by taking its complement.

In this system we can use both 1's or 2's complement but 2's complement is most common.

There is only 1 way to represent (+9) → 00001001

But three ways to represent (-9) with eight bits.

signed magnitude repⁿ - 10001001
signed 1's complement repⁿ - 11110110
signed 2's complement repⁿ - 11110111

Ex:- (+6) → 00000110
signed magnitude repⁿ - 10000110
signed 1's comp repⁿ - 11111001
signed 2's comp repⁿ - 11111010

The signed magnitude system is used in ordinary arithmetic, but awkward when employed in computer arithmetic because of seperate handling of sign & magnitude.

So, signed complement system is widely used.

1's complement is used for logical operations because its equivalent to a logical complement operⁿ.

Decimal	signed 2's complement	signed 1's complement	signed magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	-	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	-	-

↳ Positive numbers in all three repn's are identical & have '0' in the left most position.

↳ All -ve numbers have 1 in left most bit position.

↳ SIGN EXTENSION :-

+20 in 6 bits: 0 1 0 1 0 0 (sign bit)

+20 in 7 bits: 0 0 1 0 1 0 0

+20 in 8 bits: 0 0 0 1 0 1 0 0 (sign extn)

-20 in 6 bits using 2's comp: 1 0 1 1 0 0 (sign bit)

7 bits: 1 1 0 1 1 0 0

8 bits: 1 1 1 0 1 1 0 0 (sign extn.)

ARITHMETIC ADDITION:-

PROCEDURE:-

The Addition of two signed binary numbers with negative numbers represented in signed 2's complement form is obtained from the addition of the two numbers including their sign bits. A carry out of the sign-bit position is discarded.

Numerical Example:-

+6	0000 0110	-6	1 1111010
+13	0000 1101	+13	0000 1101
+19	0001 0011	+7	0000 0111

(carry is discarded)

+6	0000 0110	-6	1111 1010
-13	1111 0011	-13	1111 0011
-7	1111 1001	-19	11101101

NOTE:

The negative numbers must be initially in 2's complement form & that if the sum obtained after the addition is negative, it is in 2's complement form.

For ex; '-7' is rep^n as 11111001, which is 2's comp of +7.

↳ If we start with two n-bit numbers & the sum occupies n+1 bits, we say overflow occurs.

★ Overflow is a problem in computers because the number of bits that hold a number is finite, & a result that exceeds the finite value by 1 cannot be accomodated.

Half Addition:- Ex:- 1 + 1 = 10 [The higher significant bit of result is called carry & lower is sum, such oper^n is half A...

Full Addition:- The oper^n which performs addition of three bits [2 significant bits & a previous carry]

ARITHMETIC SUBTRACTION:-

PROCEDURE:-

↳ Subtract bits column-wise starting from LSB with borrow if any.

↳ Put the difference at the bottom of same column.

↳ Take the borrow, if required from next column.

$0 - 0 = 0$

$0 - 1 = 1$ with borrow

$1 - 0 = 1$

$1 - 1 = 0$

Ex:- $(11101100)_2 - (00110010)_2$

Sol:-

$$\begin{array}{r} 1\ \overset{1}{\not{1}}\ \overset{1}{\not{1}}\ \overset{1}{\not{0}}\ 1\ \overset{1}{\not{1}}\ \overset{1}{\not{0}}\ 0 \\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\ \hline 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0 \end{array}$$

$(\because (10)_2 - (1)_2 = (1)_2)$

BINARY SUBTRACTION USING 1's COMPLEMENT:-

In 1's complement subtraction, negative number is represented in the 1's complement form & actual addition is performed to get the desired result.

For example A-B

(i) Take 1's complement of B

(ii) Result ← A + 1's complement of B

(iii) If carry is generated then the result is positive & in true form. Add carry to the result to get final result.

(iv) If carry is not generated then the result is '-ve' & in 1's complement form.

Ex:- Perform $(28)_{10} - (15)_{10}$ using 6-bit in 1's complement.

Sol:- $(28)_{10} - (011100)_2$

$(15)_{10} - (001111)_2$

1's comp of $(15)_{10}$ | 1 | 1 | 0 | 0 | 0 | 0 |

$$\begin{array}{r} 0\ 1\ 1\ 1\ 0\ 0 \\ 1\ 1\ 0\ 0\ 0\ 0 \\ \hline 1\ 0\ 0\ 1\ 1\ 0\ 0 \\ \rightarrow\ 1 \end{array}$$

Binary equivalent of $(13)_{10}$ = | 0 | 0 | 1 | 1 | 0 | 1 |

BINARY SUBTRACTION USING 2's COMPLEMENT.

Procedure:-

↳ Take 2's comp of B

↳ Result ← A + 2's comp of B

↳ If carry is generated then the result is positive & in the true form. In this case carry is ignored.

↳ If carry is not generated then the result is negative & in the 2's complement form.

Ex:- Perform $(15)_{10} - (28)_{10}$ using 6-bit 2's comp :-

$(15)_{10} = (001111)_2$

$(28)_{10} = (011100)_2$

2's comp of 28 = 1 0 0 1 0 0

```
  0 0 1 1 1 1
  1 0 0 1 0 0
-------------
  1 1 0 0 1 1   ↳ Binary equivalent of (-13)10.
```

Verification:-

```
  0 0 1 1 0 0   → 1's comp of Result
      (Add) 1
-------------
  0 0 1 1 0 1   Binary equivalent of (13)10
```

Ending for General

BINARY Multiplication

0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1

BINARY DIVISION

0 ÷ 1 = 0

1 ÷ 1 = 1

With this fractional numbers system we can represent the fractional numbers in the following range:

$$2^{-(n-1)} \leq F \leq 1 - 2^{-(n-1)}$$

Eg: If $n = 32$, then value range is approximately

$$2^{(-31)} \leq F \leq 1 - 2^{-(31)} \Rightarrow (1 - 2.3283 \times 10^{-10})$$

But this range is not sufficient to represent fractional numbers. To accomodate very large integers & very small integers fractions, a computer must be able to represent numbers & operate on them in such a way that the position of the binary point is variable & is automatically adjusted as computation proceeds. In this case the binary point is said to float, & the numbers are called floating point numbers.

Floating Point Repn:-

The floating point numbers must be represented in floating point representation.

The floating point representation has three fields.

↳ Sign
↳ Significant digits
↳ exponent.

Let us consider the number 111101.100010 to be represented in floating point format.

(i) first binary point is shifted to the right of the first bit & the number is multiplied by the correct scaling factor to get same value. The number is said to be in normalized form & given as: $111101.100010 \rightarrow 1.11101100010 \times 2^{5}$

(5 → Exponent; 1.11101100010 → Significant digits; 2^{5} → Scaling factor; whole → Normalized form.)

The base in the scaling factor is fixed '2' & therefore does not need to appear explicitly in the machine repⁿ of a floating point numbers.

So, finally, three fields sign, significant digit & Exponent repⁿ floating point numbers system. The string of the significant digit is known as "Mantissa".

So, Sign = 0
Mantissa = 11101100010
Exponent = 5

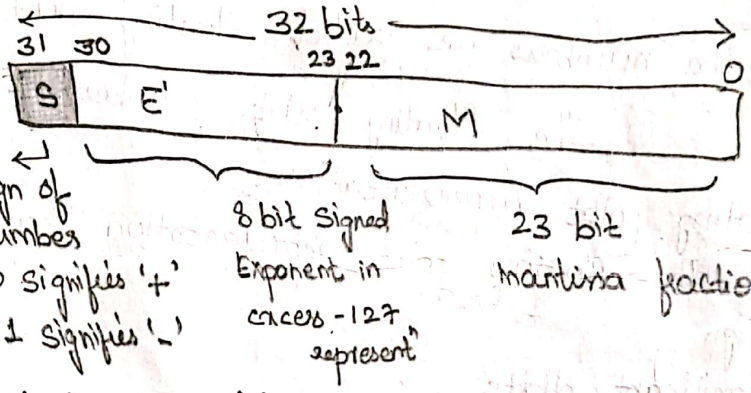
In floating point numbers, bias value is added to the true exponent. This solves the problem of representation of negative exponent. Due to this the magnitude of two numbers can be compared by doing arithmetic on the exponent first.

IEEE Standard for Floating point Numbers:-

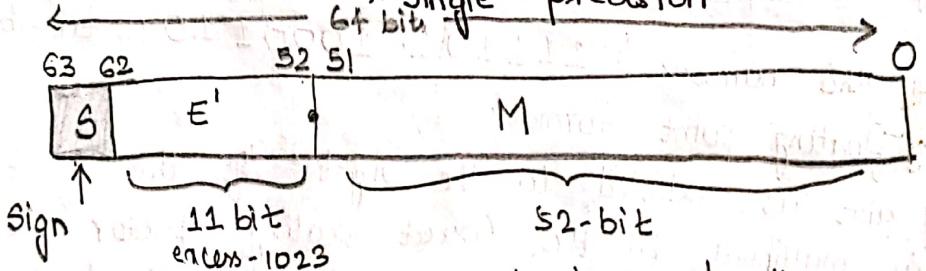
The standards for representing floating numbers in 32-bits & 64-bits have to be developed by IEEE (Institute of Electrical & Electronics Engineers (IEEE).

If it is 32-bit, then the standard represent" is called "Single-precision representation", because it occupies a single 32-bit word. The 32 bits are divided into three fields

Sign ← 1 bit
Exponent ← 8 bits
Mantissa ← 23 bits



(a) Single-precision.



(b) double-precision.

Single precision value represented as $= \pm 1.M \times 2^{E'-127}$

Double precision value represented as $= \pm 1.M \times 2^{E'-1023}$

Instead of signed Exponent 'E', the value actually stored in the exponent field is $E' = E + 127 \Rightarrow E' = E + bias$ (↳ Bias)

This representation is also called as excess-127 format.

The end values of e', namely $0 \& 255$ are use to indicate the floating point values of exact zero & Infinity, in a single precision.

The range of e' for normal single precision is $0 < e' < 255$.

This means that for ~~data~~ 32 bit rep^n the actual exponent E is in the range $-126 \leq E \leq 127$.

The 64 bit standard rep^n is called "double-precision" or "double precision representation, because it occupies two 32 bit words. The 64 bits are divided into

↳ Sign ← 1-bit
↳ Exponent ← 11-bits
↳ Mantissa ← 52-bits

$E' = E + 1023$
↳ Bias value or excess-1023 format.

The end values of E', namely 0 & 2047, are used to indicate the floating point exact values of exact zero & infinity exactly. Thus E' for normal values in double precision is $0 < E' < 2047$. This means that for 64-bit rep^n the actual exponent E is in $-1022 \leq E \leq 1023$.

PROBLEMS:-

1) Represent 1259.125_{10} in the single precision & double precision format.

Step-1 Convert decimal numbers into Binary format.

Sol: Integer part:- 1259

$$\begin{array}{r|l} 16 & 1259 \\ \hline 16 & 78 \\ \hline & 4 \end{array} \quad \begin{array}{l} -11 \to B \\ -14 \to E \\ 4 \to 4 \end{array}$$

(Q R)

$$\begin{array}{r} 78 \\ 16\overline{)1259} \\ 112 \\ \hline 0139 \\ 128 \\ \hline 011 \end{array} \qquad \begin{array}{r} 4 \\ 16\overline{)78} \\ 64 \\ \hline 14 \to E \end{array}$$

$\hookrightarrow B$

$\therefore (1259)_{10} = (4EB)_{16}$

$\Rightarrow 4EB = \underbrace{0100}_{4}\ \underbrace{1110}_{E}\ \underbrace{1011}_{B}$ → Binary numbers → Hex numbers

Fractional Part:-

$0.125 \times 2 = 0.25$
↓ ↓ ↓
Fraction, Base Product

$0.125 \times 2 = 0.25 \rightarrow 0$

$0.25 \times 2 = 0.50 \rightarrow 0$ so it is 0.001

$0.50 \times 2 = 1.00 \rightarrow 1$

∴ Binary number = $1001.101011 + 0.001$ (combine & write)

$= 1001.101011001$.

Step-2 [Normalize the number].

$1001.101011001 = 1.001101011001 \times 2^{10}$

Step-3 Single precision Rep^n :-

For given number $S=0$, $E=10$ & $M = \underbrace{00111010110011}_{\text{Mantissa}}$

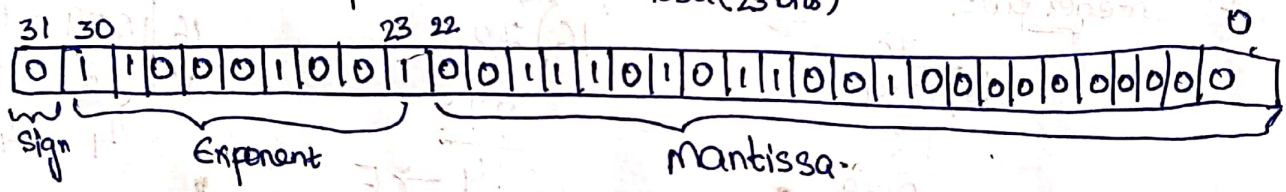
for single precision format is 127

$E' = E + 127 = 10 + 127 = 137_{10}$

$= 10001001_2$

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

0 (Sign) | 10001001 (Exponent) | 00111010110011 (Mantissa (23 bits))



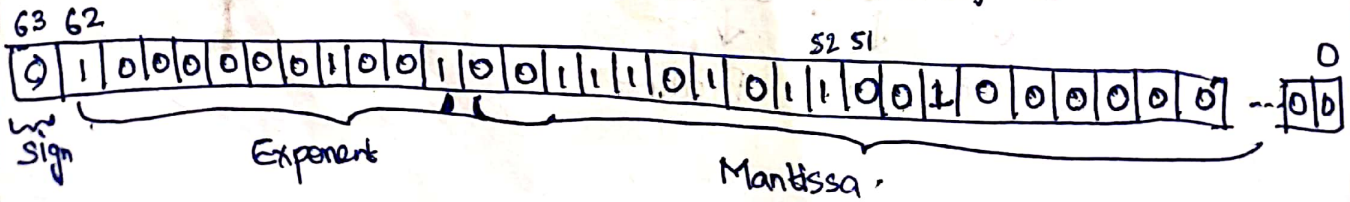
Step-4 :- Double precision Rep^n :-

$S=0$, $E=10$, $M = 0011101011001$

Bias for DPR = 1023 $\Rightarrow E' = E + 1023 = 1033_{10}$

$= 10000001001_2$

0 (Sign) | 10000001001 (Exponent) | 0011101011001 ---- 0 (M (52 bits))



2) Represent -307.1875_{10} in Single precision & double precision formats.

Step-1: Convert decimal pa number in binary format integer part.

16) 307 (19 — 304 — 3

16) 19 (1 — 16 — 3

$\therefore (307)_{10} = (133)_{16}$

1 → 0001 | 3 → 0011 | 3 → 0011 | Hex numbers / Binary numbers

So, $(307)_{10} = (100110011)_2$ ~~0001~~

Fractional part:-

$0.1875 \times 2 = 0.375 \rightarrow 0$

$0.375 \times 2 = 0.75 \rightarrow 0$

$0.75 \times 2 = 1.50 \rightarrow 1$

$0.50 \times 2 = 1.00 \rightarrow 1$

$\therefore (0.1875)_{10} = (0.0011)_2$

Binary number = $-100110011 + 0.0011 \Rightarrow -100110011.0011$

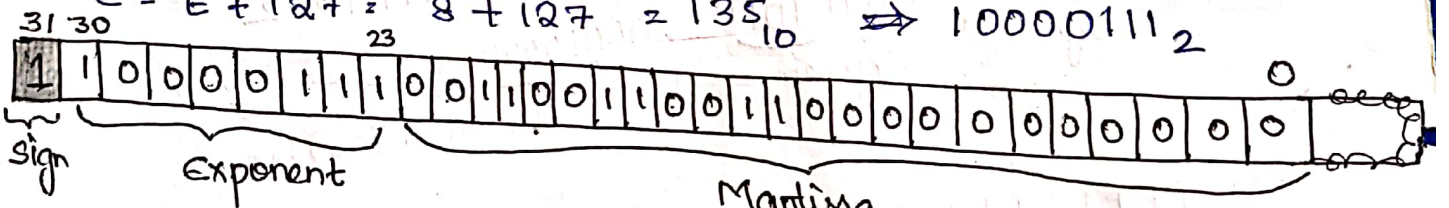
Step-2:- Normalize the number

$-100110011.0011 = -1.00110011 0011 \times 2^8$

Step-3:- Single precision format:-

For a given number $S=1$, $E=8$ & $M = 00110110011$

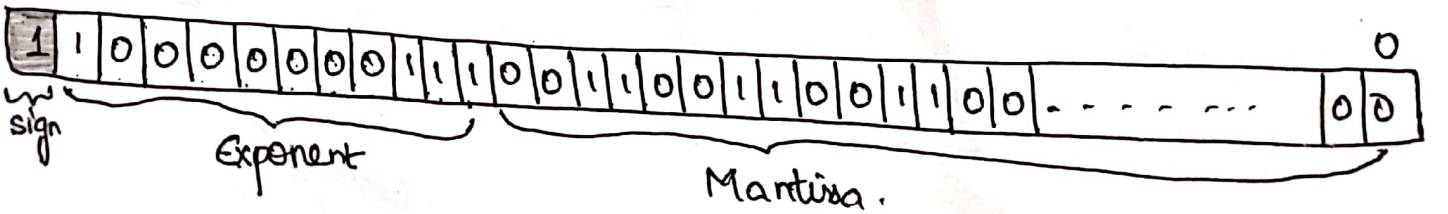
$E' = E + 127 = 8 + 127 = 135_{10} \Rightarrow 10000111_2$



Double precision rep^n:- (Step-4)

$S=1$, $E=8$ & $M = 00110110011$ $\quad E' = E + 1023 = 8 + 1023 \Rightarrow 1031 = 10000000111_2$

1 (Sign) | 10000000111 (Exponent) | 00110011001100...0 (Mantissa (52-bits))



3. Convert 0.0625_{10} in single & Double precision formats.

Step-1 Convert decimal number in binary format.

Integer part = 0

Fraction part:-

$0.0625 \times 2 = 0.125 \rightarrow 0$

$0.125 \times 2 = 0.25 \rightarrow 0$

$0.25 \times 2 = 0.50 \rightarrow 0$

$0.50 \times 2 = 1.00 \rightarrow 1$

$\therefore (0.0625)_{10} = (0001)_2$

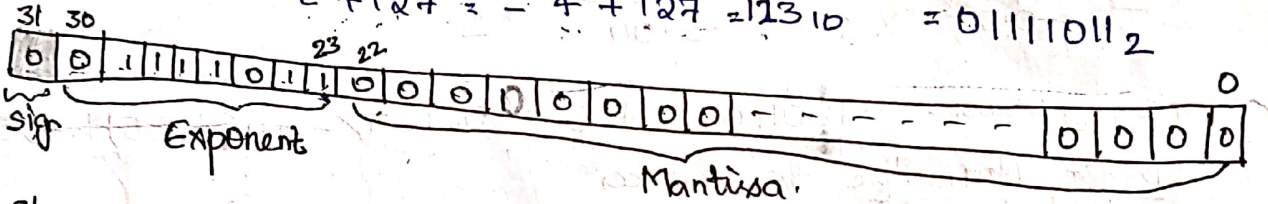
Step-2:- Normalize the number

Step-3:- $0.0001 = 1.0 \times 2^{-4}$

Single precision representation

For given number $S = 0$, $E = -4$, $M = 0001$

$E' = E + 127 = -4 + 127 = 123_{10} = 01111011_2$



Step-4:-

$S = 0$, $E = -4$, & $M = 0001$ $\quad E' = E + 1023 \Rightarrow -4 + 1023 = 1019$

$= 01111111011_2$

0 | 011 1111 1011 | 0001000000000---0

sign | Exponent | Mantissa (52) bits.