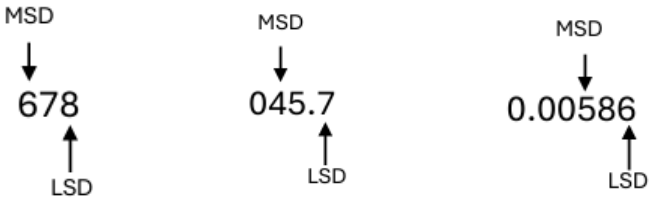
UNIT 03 SHORTNOTE

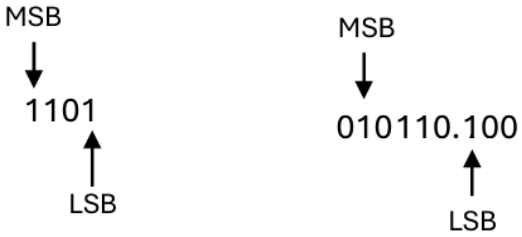
✦ Most Significant Digit and Least Significant Digit



You can use this method to find the most and least significant digits of decimal, octal, and hexadecimal numbers.

✦ Most Significant Bit and Least Significant Bit

Only binary number system is used to identify the MSB and the LSB.



rightmost bit is the LSB, and the leftmost bit that is not zero is the MSB.

✦ Decimal Number System

- These are the numbers that we use in day-to-day life.
- The base is 10.
- Digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Positive values are represented by (+) and negative values are represented by (-)
- Contains integer values and fractional values. Fractional part is separated by a dot (.)

Eg - 345.632

✦ Binary Number System

This system has two digits which can represent two states.
- These two states are represented by digits 1 and 0.
- In binary numbers, place values are multiples of 2.

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
16	8	4	2	1	.5	.25	.125

→ Binary to Decimal

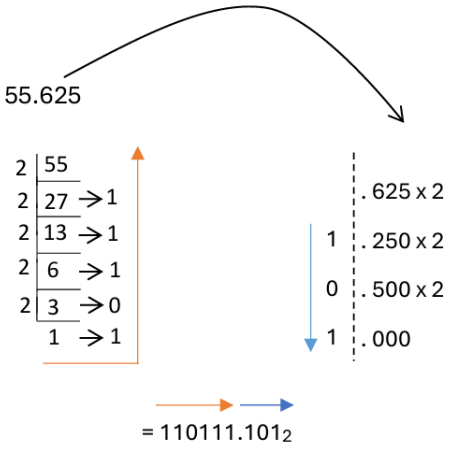
101010.11_2

1	0	1	0	1	0	.	1	1
2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}

32 + ~~16~~ + 8 + ~~4~~ + 2 + ~~1~~ + 1/2 + 1/4

=42.75

→ Decimal to Binary



✦ Octal Number System

- Base value is 8.
- Digits used in this representation are 0,1,2,3,4,5,6 & 7.
- In octal numbers, place values are multiples of 8.

E.g :

8^2	8^1	8^0	8^{-1}	8^{-2}
64	8	1	1/8	1/64

→ Octal to Decimal

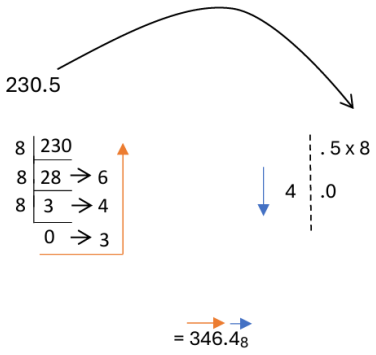
166.1_8

1	6	6	.	1
8^2	8^1	8^0		8^{-1}

32 + 16 + 6 + 1/8

=54.125

→ Decimal to Octal



✦ Hexadecimal Number System

- Base value is 16.
- Contains 16 digits starting from 0 to last digit F.
- The values of the digits are represented as follows. usernames, passwords, and different access levels.

Hexadecimal	0	1	2	3	4	5
Decimal	0	1	2	3	4	5

6	7	8	9	A	B	C	D	E	F
6	7	8	9	10	11	12	13	14	15

➔ Hexadecimal to Decimal

$2A1.2_{16}$

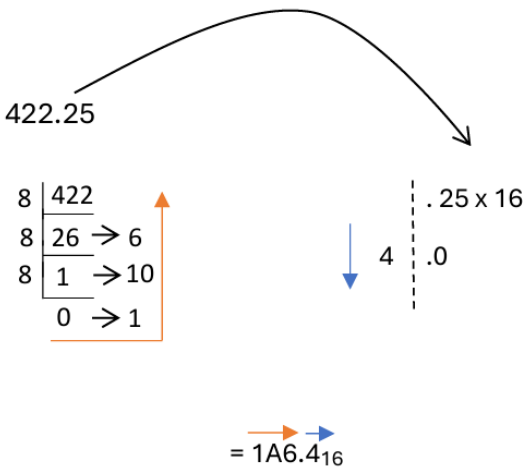
2 A 1 . 2

16^2 16^1 16^0 16^{-1}

512 + 160 + 1 + 0.125

=673.125

➔ Decimal to Hexadecimal



✦ Conversions between Number systems

➔ Binary to Octal

* Group 3 binary bits each, starting from the right.
* Find the total of each group and use the corresponding octal digit.

Eg – 11001101_2

0 1 1 | 0 0 1 | 1 0 1_2

3 1 5 $_8$

➔ Octal to Binary

* Represent each octal digit as 3 binary bits.

Eg - 607_8

6 0 7 $_8$

1 1 0 | 0 0 0 | 1 1 1_2

➔ Binary to Hexadecimal

* Group 4 binary bits each, starting from the right.
* Find the total of each group and use the corresponding octal digit.

Eg – 101010011101_2

1 0 1 0 | 1 0 0 1 | 1 1 0 1_2

A 9 D $_{16}$

➔ Hexadecimal to Binary

* Represent each octal digit as 4 binary bits.

Eg – $6B3_{16}$

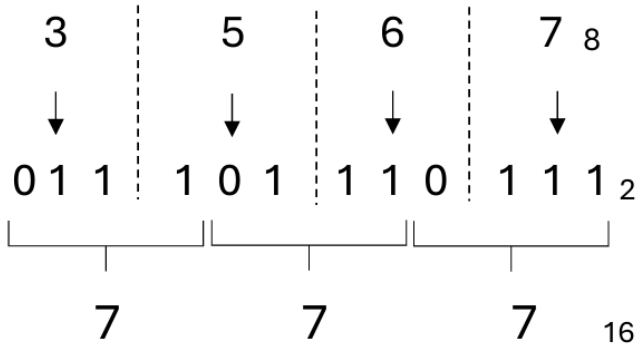
6 B 3 $_{16}$

0 1 1 0 | 1 0 1 0 | 0 0 1 1_2

➔ Octal to Hexadecimal Conversion

* First, convert to binary number and then Convert to Hexadecimal number.

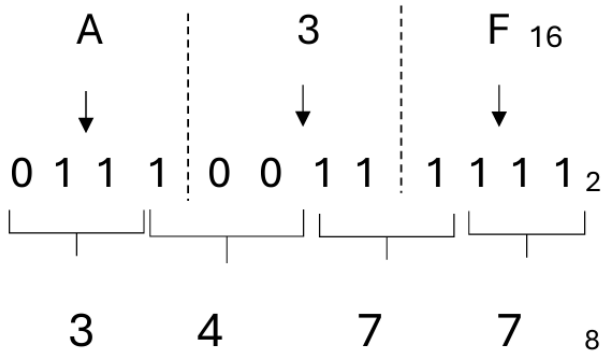
Eg – $6B3_{16}$



➔ Hexadecimal to Octal Conversion

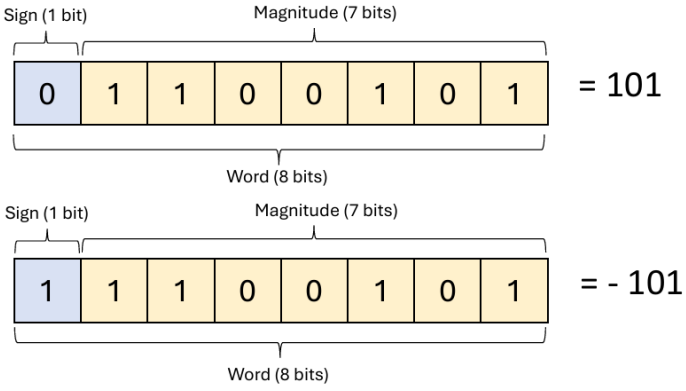
* First, convert to binary number and then Convert to Hexadecimal number.

Eg – $6B3_{16}$



✦ Number Representation

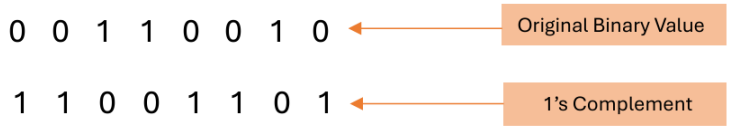
➔ Sign Magnitude



➔ One's Complement

* positive numbers are represented in standard binary form.
* negative numbers are formed by flipping all bits- changing zeros to ones and ones to zeros.

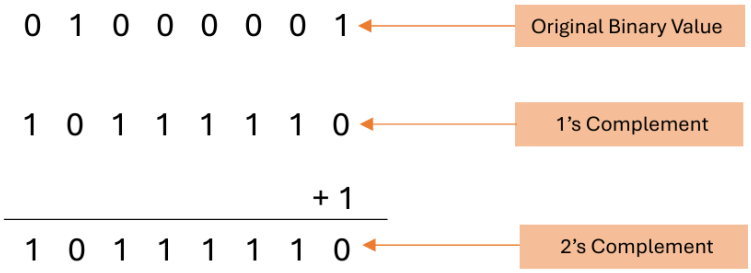
Eg : -50 in One's complement



➔ Two's Complement

* positive numbers are represented in standard binary form.
* to obtain the negative value of a number, invert all the bits of its positive value and then add one to the result.

Eg : - 65

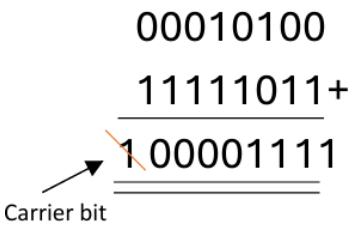


Addition using 1's complement and 2's complement

- Computation 20+(-5) is done in 8-bit two's complement form.

20 ➔ 00010100 **5** ➔ 00000101

- 5 ➔ 11111010 + 00000001 = 11111011



* **In one's Complement additions**, carrier bit is added again to the answer, **But in Two's complement**, carrier bit is ignored

➔ Differences between 1's Complement and 2's Complement

1's Complement	2's Complement
1. Zero has two representations, +0 and -0	Zero has only one representation
2. Negative values obtained by flipping bits	To obtain a negative value, should flip bits and add 1
3. More complex for arithmetic operations due to considering carrier bit	Much easier in arithmetic operations
4. Less used in modern computing systems	Widely used in modern computer systems

✦ Fixed point numbers

* The decimal point is unchanged.

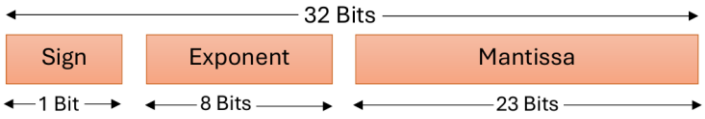
Eg :- 850.753
652.365 +
1503.118

✦ Floating point numbers

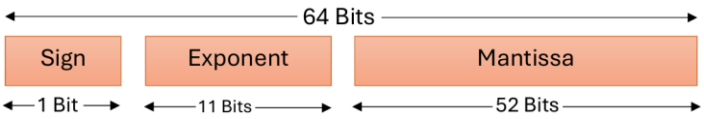
Used to represent:
- Numbers with fractions, such as 3.1416
- Very small numbers, like 0.000000001
- Very large numbers, for example, 3.15576 × 10^9

IEEE 754 standard has three main components:
1. Sign Bit: This indicates the sign of the number.
2. Exponent: A bias is added to the actual exponent
3. The Normalized Mantissa

1. Single Precision



2. Double Precision



Advantages and disadvantages of Fixed-point and Floating-point representations

	Advantage	Disadvantage
Fixed Point Representation	The performance is good, not requiring extra hardware or software logic	Limited range of values can represent
Floating point representation	A wider range of numbers can be represented, with different levels of precision.	Requires more storage space, slower processing times, and reduced precision.

✦ Coding systems used in Computers

➔Binary Coded Decimal (BCD)

* each decimal digit is represented by 4 bits.

Decimal value	BCD value
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

➔ ASCII – American Standard Code for Information Interchange

* 7-bit code used to represent text in computers and communication devices.
* primary encoding system used in IBM PCs.

Extended ASCII expands this by using 8 bits

Eg:

Character	ASCII code	Binary code
null	o	0000000
a	97	1100001
b	98	1100010
0	48	0110000

➔ EBCDIC (Extended Binary Coded Decimal Interchange Code)

* 8-bit character encoding is primarily used on IBM mainframe and midrange systems.

Differences between ASCII and EBCDIC

ASCII	EBCDIC
Used for electronic communication	Used on IBM mainframe & IBM midrange computers
Uses 7 bits	Uses 8 bits
Only 128 characters can be represented	256 characters can be represented
Compatible with Unicode	Not compatible with Unicode
More efficient	Less efficient

→UNICODE

* Widely adopted by major companies like Apple, HP, IBM, Microsoft, Oracle, SAP, and many others.
* It can store a vast number of characters, including those from almost every language in the world.

There are different types of Unicode:
- UTF-8: Uses 1 byte (8 bits).
- UTF-16: Uses 2 bytes (16 bits).
- UTF-32: Uses 4 bytes (32 bits).

Differences between ASCII and Unicode

Property	ASCII	Unicode
Space Requirement	Needs less space	Needs more space than ASCII
Bit Usage	Uses 7 bits	Uses 8/16 or 32 bits
Relationship	ASCII is a subset of Unicode	Unicode is the superset of ASCII
Conversion	ASCII →Unicode possible	Most Unicode → ASCII impossible
Language Support	Limited to English	Supports most written languages

✦ **Bitwise operations**

1. NOT operation

E.g:- NOT 01112 (7_{10})
= 10002 (8_{10})

A	NOT A
0	1
1	0

2. Bitwise AND operation

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

E.g. :- 01012 (5_{10})
AND 00112 (3_{10})

$0\,1\,0\,1_2$
$0\,0\,1\,1_2$
$0\,0\,0\,1_2\,(1_{10})$

3. Bitwise OR operation

E.g. :- 01012 (5_{10}) OR 00112 (3_{10})

$0\,1\,0\,1_2$
$0\,0\,1\,1_2$
$0\,1\,1\,1_2\,(7_{10})$

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

4. Bitwise XOR operation

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

E.g :- 00102 (210) XOR 10102 (1010)

$1\,0\,1\,0_2$
$0\,0\,1\,0_2$
$1\,0\,0\,0_2\,(8_{10})$

	Advantage	Disadvantage
BCD	• Easy to encode and decode decimals into BCD and vice versa. • Simple to implement a hardware algorithm for the BCD converter. • It is very useful in digital systems whenever decimal information is given either as inputs or displayed as outputs. • Digital voltmeters, frequency converters and digital clocks all use BCD as they display output information in decimal.	• Not space efficient. • Difficult to represent the BCD form in high speed digital computers in arithmetic operations, especially when the size and capacity of their internal registers are restricted or limited. •Require a complex design of Arithmetic and logic Unit (ALU) than the straight Binary number system. •The speed of the arithmetic operations slow due to the complete hardware circuitry involved.
ASCII	• Uses a linear ordering of letters. • Different versions are mostly compatible. • compatible with modern encodings	• Not Standardized. • Not represent world languages.
EBCDIC	• uses 8 bits while ASCII uses 7 before it was extended. • Contained more characters than ASCII.	• Does not use a linear ordering of letters. • Different versions are mostly not compatible. • Not compatible with modern encodings
UNICODE	• Standardized. • Represents most written languages in the world. • ASCII has its equivalent within Unicode.	• Need twice as memory to store ASCII characters.