

**Program :** Diploma in Computer Engineering  
**Course Code :** 3134  
**Course Title:** Digital Computer Fundamentals  
**Semester :**3

## **Module 1: Number Systems:**

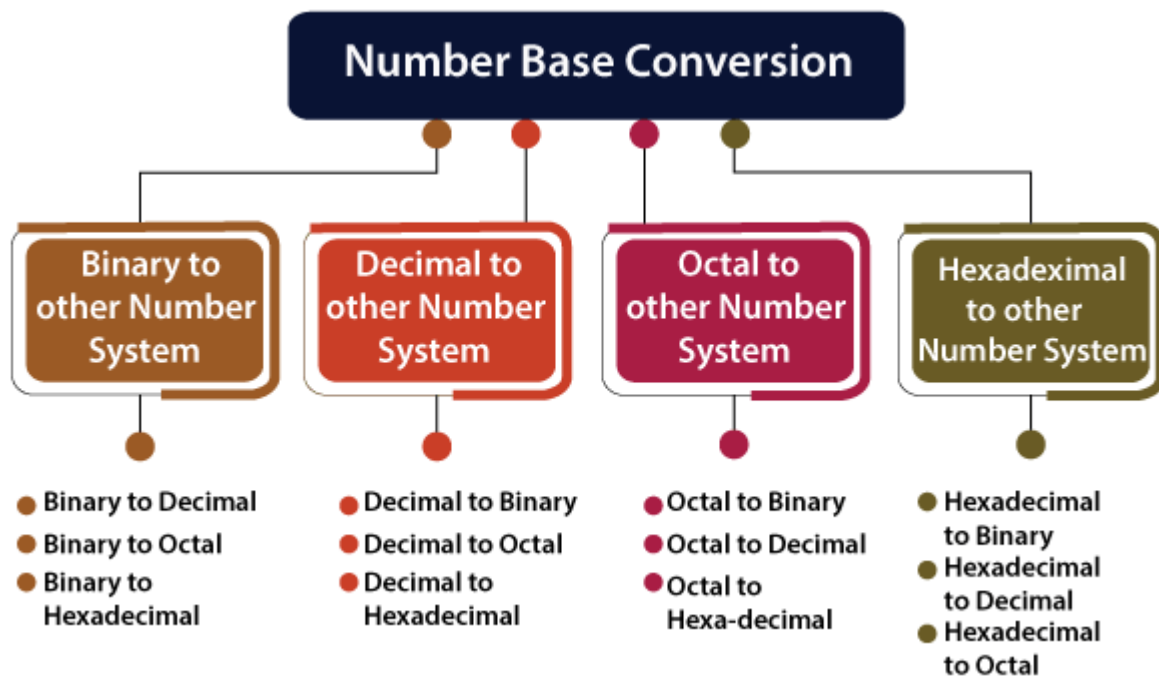
### **Important topics**

- 1.Decimal to binary,hex,octal conversions**
- 2.Binary to Decimal,Hex,Octal conversions**
- 3.2's compliment subtraction**
- 4.BCD addition**
- 5.Gray Code**
- 6.Parity**
- 7.Hamming code**

### **Introduction to different Number Systems – Decimal, Binary, Octal**

In mathematics, a “base” or a “radix” is the number of different digits or combination of digits and letters that a system of counting uses to represent numbers.

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14



## 1.DECIMAL TO BINARY

Convert Decimal (98.46) to Binary

Binary equivalent of 98 is  $1100010_2$

Fractional part:

$$\begin{aligned}
 0.46 \times 2 &= 0.92 = 0 \\
 0.92 \times 2 &= 1.84 = 1 \\
 0.84 \times 2 &= 1.68 = 1 \\
 0.68 \times 2 &= 1.36 = 1 \\
 0.36 \times 2 &= 0.72 = 0
 \end{aligned}$$

2	98	
2	49	- 0
2	24	- 1
2	12	- 0
2	6	- 0
2	3	- 0
1		- 1

**Ans: 1100010.01110 ...**

## 2.BINARY TO DECIMAL

Convert Binary number 11011.101 to decimal **Ans:27.625**

$$\begin{array}{cccccccc}
 2^4 & 2^3 & 2^2 & 2^1 & 2^0 & 2^{-1} & 2^{-2} & 2^{-3} \\
 1 & 1 & 0 & 1 & 1 & . & 1 & 0 & 1
 \end{array}$$

Integral Part  $(11011)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$$\begin{aligned}
 &= 16 + 8 + 0 + 2 + 1 \\
 &= (27)_{10}
 \end{aligned}$$

Fractional Part  $(0.101)_2 = 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

$$\begin{aligned}
 &= 0.5 + 0 + 0.125 \\
 &= (0.625)_{10}
 \end{aligned}$$

### 3.DECIMAL TO OCTAL


► Convert the decimal number  $(225.225)_{10}$  into octal number.

Solution:

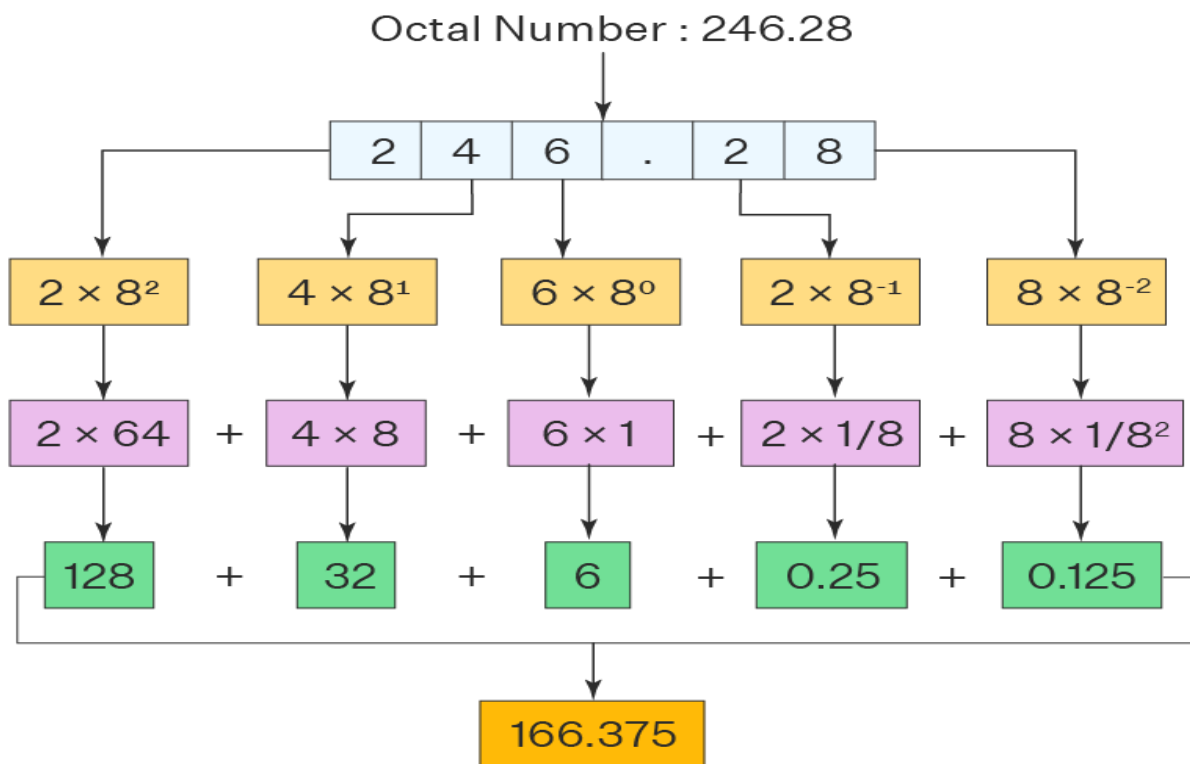
$$\begin{array}{r} 8 \overline{) 225} \\ 8 \overline{) 28} \quad 1 \\ 8 \overline{) 3} \quad 4 \\ \quad 0 \quad 3 \end{array} \quad \uparrow$$

Fractional part:

$$\begin{array}{rcl} 0.225 \times 8 = 1.800 & 1 & \\ 0.800 \times 8 = 6.400 & 6 & \\ 0.400 \times 8 = 3.200 & 3 & \\ 0.200 \times 8 = 1.600 & 1 & \\ 0.600 \times 8 = 4.800 & 4 & \end{array} \quad \downarrow$$


$$(225.225)_{10} = (341.16314)_8$$

### 4.OCTAL TO DECIMAL



Decimal Number = 166.375

$$(246.28)_8 = (166.375)_{10}$$

## 5.DECIMAL TO HEXADECIMAL

- Convert the decimal number  $(225.225)_{10}$  into hexadecimal number.

Solution:

$  \begin{array}{r}  16 \overline{) 225} \\  \underline{16 \overline{) 14}} \phantom{0} \\  0 \phantom{00} 14 \\  \phantom{00} \phantom{00} \text{(or E)}  \end{array}  $	$\uparrow$	<p>Fractional part:</p> $  \begin{array}{rcl}  0.225 \times 16 & = & 3.600 \quad 3 \\  0.600 \times 16 & = & 9.600 \quad 9  \end{array}  $	$\downarrow$
---	------------	--	--------------

$$(225.225)_{10} = (E1.39)_{16}$$

## 6.HEXADECIMAL TO DECIMAL

Convert Hexadecimal number 54.D2 to Decimal

Digit	5	4	.	D	2
Place value	$16^1$	$16^0$		$16^{-1}$	$16^{-2}$

$$54.D2_{16}$$

$$= 5 \cdot 16^1 + 4 \cdot 16^0 + D \cdot 16^{-1} + 2 \cdot 16^{-2}$$

$$= 5 \cdot 16^1 + 4 \cdot 16^0 + 13 \cdot 16^{-1} + 2 \cdot 16^{-2}$$

$$= 80 + 4 + 0.8125 + 0.0078125$$

$$= 84.8203125$$

## 7.BINARY TO OCTAL

Binary Number: 101010011.110100

Group of three digits:  $\overline{101} \ \overline{010} \ \overline{011} . \overline{110} \ \overline{100}$

↓ ↓ ↓ ↓ ↓

Octal Equivalent: 5 2 3 6 4

= (523.64)<sub>8</sub>

## 8.OCTAL TO BINARY

### Octal to Binary

**1 2 . 5**

001 010 101

$(12.5)_8 = (001010.101)_2$

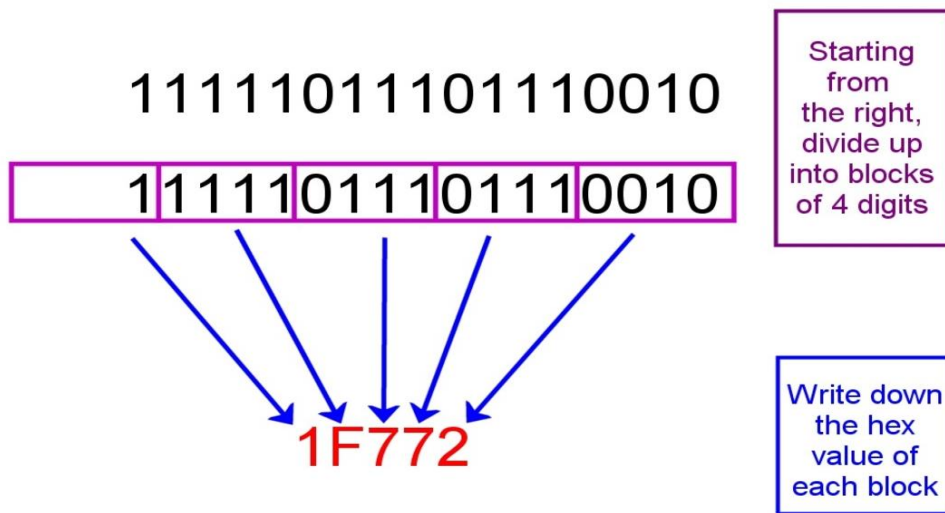
**7 2 1**

111 010 001

$(721)_8 = (111010001)_2$

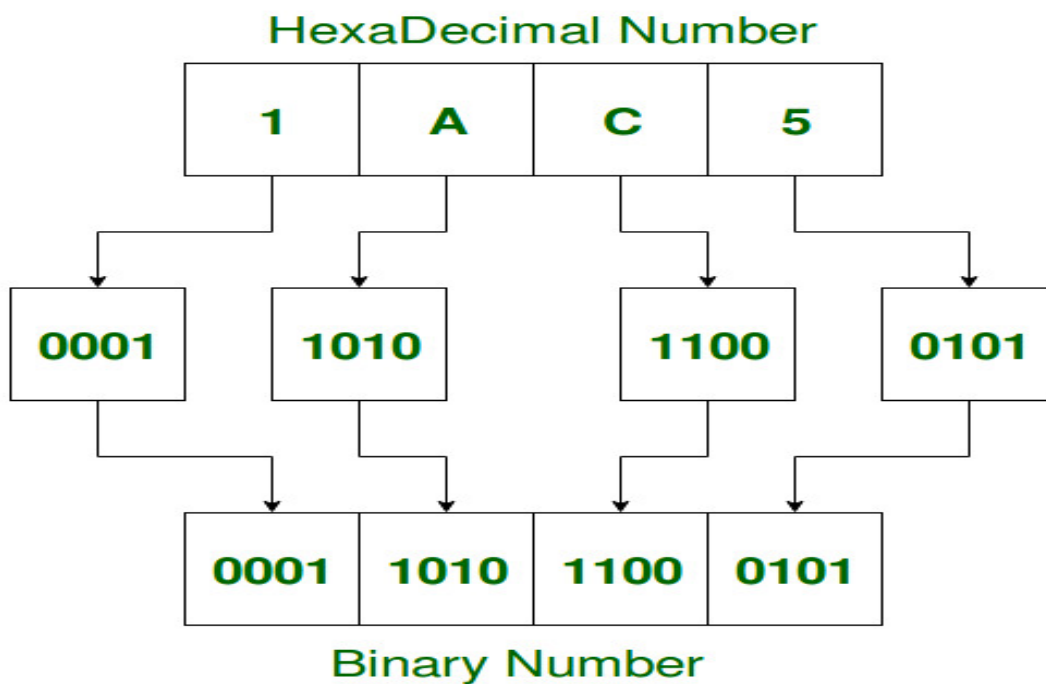
Binary	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

## 9.BINARY TO HEXADECIMAL

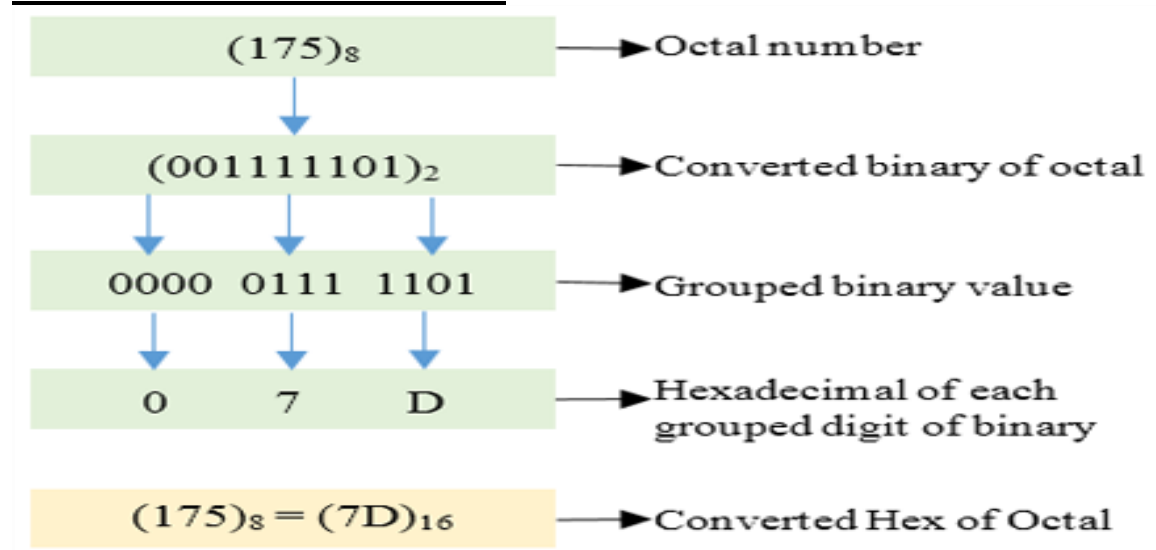


$$11111011101110010_2 = 1F772_{16}$$

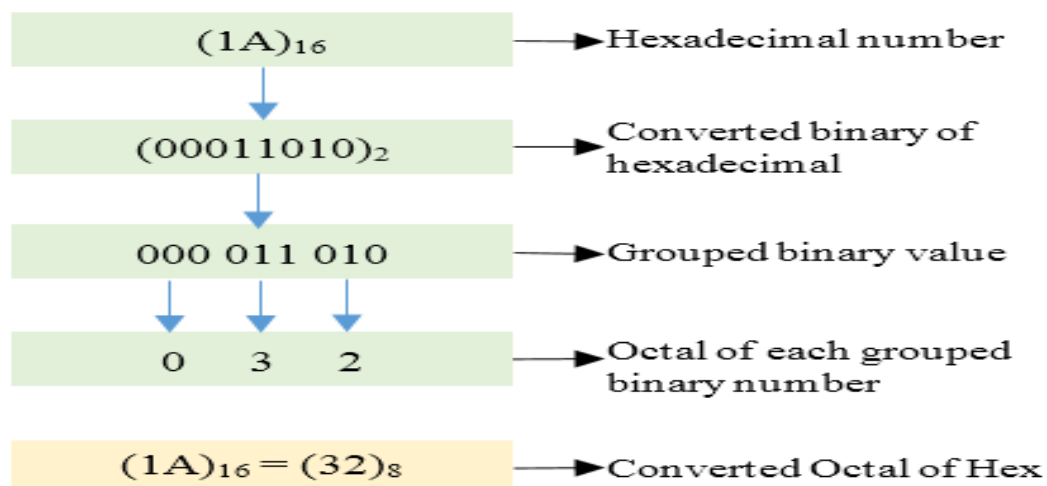
## 10.HEXADECIMAL TO BINARY



## OCTAL TO HEXADECIMAL



## HEXADECIMAL TO OCTAL



## BINARY ARITHMETIC

	Addition	Subtraction	Multiplication	Division
i)	$0 + 0 = 0$	$0 - 0 = 0$	$0 \times 0 = 0$	$0 / 1 = 0$
ii)	$0 + 1 = 1$	$1 - 0 = 1$	$0 \times 1 = 0$	$1 / 1 = 1$
iii)	$1 + 0 = 1$	$1 - 1 = 0$	$1 \times 0 = 0$	$0 / 0 = \text{not allowed (not valid)}$
iv)	$1 + 1 = 10$	$1 - 0 = 10 - 1$ (with borrow 1) = 1	$1 \times 1 = 1$	$1 / 0 = \text{not allowed (not valid)}$



## Negative Binary Numbers

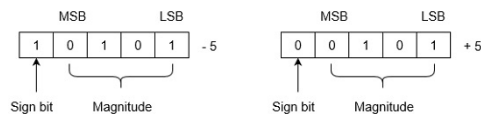
Negative numbers can be distinguishable with the help of extra bit or flag called sign bit or sign flag in Binary number representation system for signed numbers. That's why we use this extra bit called sign bit or sign flag. The value of sign bit is 1 for negative binary numbers and 0 for positive numbers.

When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number.

When the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways: **Sign-Magnitude method, 1's Complement method, and 2's complement method.**

### Sign-Magnitude method

In this method, number is divided into two parts: Sign bit and Magnitude. If the number is positive then sign bit will be 0 and if number is negative then sign bit will be 1. Magnitude is represented with the binary form of the number to be represented.



## Complements

Complements are used in digital computers to simplify the subtraction operation and for logical manipulation. Simplifying operations leads to simpler, less expensive circuits to implement the operations.

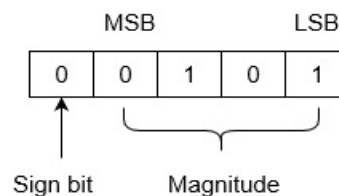
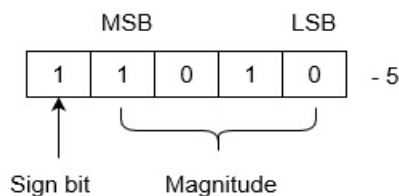
Generally, there are two types of complement of Binary number: 1's complement and 2's complement.

### 1's Complement Method:

Positive numbers are represented in the same way as they are represented in sign magnitude method. If the number is negative then it is represented using 1's complement. First represent the number with positive sign and then take 1's complement of that number. To get 1's complement of a binary number, simply invert the given number.

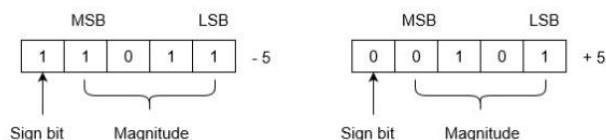
Invert all bits. Each 1 becomes a 0, and each 0 becomes a 1.

Original Value	One's Complement
0	1
1	0
1010	0101
1111	0000
11110000	00001111
10100011	01011100
11110000 10100101	00001111 01011010



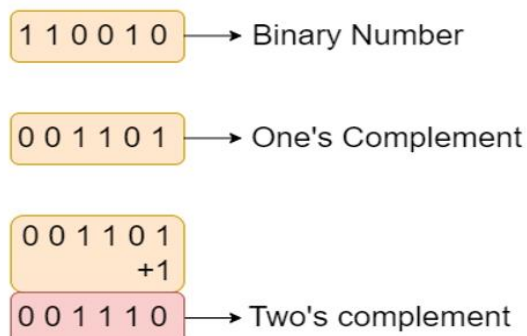
## 2's Complement Method:

Positive numbers are represented in the same way as they are represented in sign magnitude method. If the number is negative then it is represented using 2's complement. First represent the number with positive sign and then take 2's complement of that number.



2's complement of a binary number is obtained by adding 1 to the 1's complement

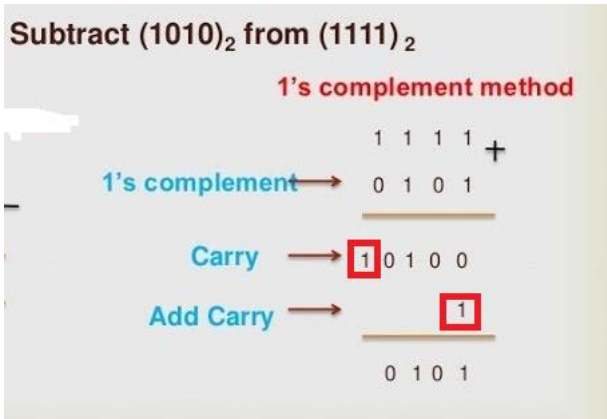
2's complement = 1's complement + 1



## Binary Subtraction with compliments

If we want to find the difference between M and N

ie Difference=M-N

	1's Complement	2's Complement
M>N	<p>Determine 1's complement of 'N'</p> <p>Add this to M</p> <p>Add carry to the result</p> <p>Result= 1'sCompliment(N)+M+ Carry</p>  <p><b>Subtract <math>(1010)_2</math> from <math>(1111)_2</math></b></p> <p><b>1's complement method</b></p> <p>1's complement <math>\rightarrow</math> 0 1 0 1</p> <p>Carry <math>\rightarrow</math> 1 0 1 0 0</p> <p>Add Carry <math>\rightarrow</math> 0 1 0 1</p>	<p>Determine 2's complement of 'N'</p> <p>Add this to M</p> <p>Ignore Carry</p> <p>Eg: Find 1111-1010</p> <p>M=1111 , N=1010</p> $  \begin{array}{r}  1111 + \\  2's\ Complement \rightarrow 0110 \\  \hline  10101  \end{array}  $ <p>Ignore Carry 1</p> <p>Result: 0 1 0 1</p>
M<N	<p>Determine 1's complement of 'N'</p> <p>Add this to M</p> <p>Find 1's complement of result</p> <p>Put -ve sign in the result</p> <p>Eg: Find 1010 - 1111</p> <p>M=1 0 1 0 , N=1 1 1 1</p> $  \begin{array}{r}  1010 + \\  1's\ Complement \rightarrow 0000 \\  \hline  1010  \end{array}  $ <p>1's Complement (1 0 1 0) <math>\rightarrow</math> 0 1 0 1</p> <p>Put -ve Sign <math>\rightarrow</math> - 0 1 0 1</p>	<p>Determine 2's complement of 'N'</p> <p>Add this to M</p> <p>Find '2s complement of result</p> <p>Put -ve sign in the result</p> <p>Eg: Find 1010 -1111</p> <p>M=1 0 1 0 , N=1 1 1 1</p> $  \begin{array}{r}  1010 + \\  2's\ Complement \rightarrow 0001 \\  \hline  1011  \end{array}  $ <p>2's Complement (1011) <math>\rightarrow</math> 0 1 0 1</p> <p>Put -ve Sign <math>\rightarrow</math> - 0 1 0 1</p>

## **BINARY CODES**

- ✓ Digital systems require to handle the data which may be numeric, alphabets or special characters.
- ✓ As these digital systems operate in a binary system and hence the numerals and alphabets are to be converted into binary format
- ✓ Process of converting into binary is called coding
- ✓ Codes are classified into 2 groups.

<b>Weighted Codes</b>	<b>Unweighted Codes</b>
Exhibit a specific weights assigned to the bit position eg– BCD, 2421, 5211	Exhibit no specific weights assigned to the bit position Eg—Excess 3 code , Gray code

## **BCD and its ADDITION**

**BCD** or **binary-coded decimal** is a special kind of representation of a decimal number in binary numbers. In binary-coded decimal each individual digit of a number is converted into a binary number, and then by combining them all, the BCD code is generated. But always remember that a binary-coded decimal is not a binary representation of a decimal number.

DECIMAL	BCD 8 4 2 1 (weights)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
11	0001 0001
.	.
.	.
.	.

Example: The **BCD** or **binary-coded decimal** of the number **15** is 00010101. The 0001 is the binary code of 1 and 0101 is the binary code of 5.

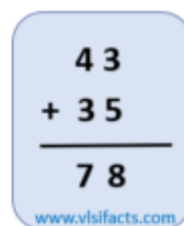
Any single decimal numeral [0-9] can be represented by a four bit pattern. The procedure of encoding digits is called "**Natural BCD**" (**NBCD**), where each decimal digit is represented by its corresponding four-bit binary value

- **Step 1:** Add the two BCD numbers using the rules for binary addition.
- **Step 2:** If a 4-bit sum is equal to or less than 9, it is a valid BCD number.
- **Step 3:** If a 4-bit sum is greater than 9 or if a carry-out of the 4-bit group is generated, it is an invalid result. Add 6 (0110) to the 4-bit sum in order to skip the six invalid BCD code words and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

**Example 1:** Find the sum of the BCD numbers 01000011 + 00110101

$$01000011_{\text{BCD}} = 43_{10} \text{ and } 00110101_{\text{BCD}} = 35_{10}$$

$$\begin{array}{r} 0100\ 0011 \\ + 0011\ 0101 \\ \hline 0111\ 1000 \end{array}$$


$$\begin{array}{r} 43 \\ + 35 \\ \hline 78 \end{array}$$

www.vlsifacts.com

In the above example, all the 4-bit BCD additions generate valid BCD numbers, which means less than 9. So, the final correct result is  $78_{10} = 01111000_{\text{BCD}}$ .

Let's take an example where the addition generates an invalid BCD number.

**Example 2:** Find the sum of the BCD numbers 01110101 + 00110101

Solution:

The decimal number of the given BCD numbers are as below:

$$01110101_{\text{BCD}} = 75_{10} \text{ and } 00110101_{\text{BCD}} = 35_{10}$$

$$\begin{array}{r}
 0111 \quad 0101 \\
 + 0011 \quad 0101 \\
 \hline
 1010 \quad 1010 \\
 + 0110 \quad + 0110 \\
 \hline
 0001 \quad 0001 \quad 0000 \\
 \underbrace{\hspace{1cm}} \quad \underbrace{\hspace{1cm}} \quad \underbrace{\hspace{1cm}} \\
 1 \quad 1 \quad 0
 \end{array}$$

Both left and right BCD numbers are invalid. So we would add 6 to both the BCD numbers.

75

+ 35

110

[www.vlsifacts.com](http://www.vlsifacts.com)

In the above example, both the BCD code addition generated result larger than 9, so invalid. To get the correct result, we added 6 ( $0110_2$ ) to both the invalid sum. Notice that the carry generated from the left group is forwarded to the right group.

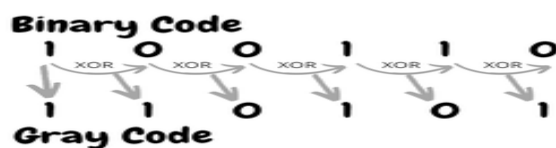
The final correct result is  $110_{10} = 000100010000_{\text{BCD}}$

## GRAY CODE

The code which exhibits only a single bit change from one code number to the next. As there are no specific weight assigned to the bit positions and hence this code is an **unweighted code**

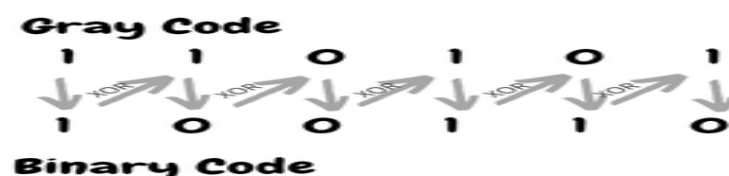
### **Convert binary to Gray code**

First MSB in the binary code as that of the corresponding digit in the gray code and next perform ex-or operation



### Gray to Binary

First MSB in the gray code as that of the corresponding digit in the binary code and next perform ex-or operation in diagonal manner



## ALPHA NUMERIC CODE

Digital system require the handling of data that consist not only of numbers but also letters of the alphabet and certain special characters (=, +, -, \*, #)

A code which contains letters, numbers and other symbols known as Alpha numeric codes

### **2 types of alpha numeric codes are there;**

#### **1. ASCII:(American Standard Code for Information Interchange)**

It is a 7 bit code in which the decimal digits are represented by 8421 coded preceded by 011

eg: Letter B is represented by 1000010 -  $(42)_{16}$

“esc” is represented by 0011011 -  $(1B)_{16}$

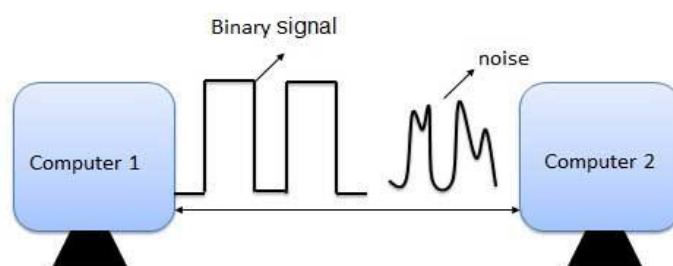
#### **2. EBCDIC:(Extended Binary Coded Decimal Interchange Code)**

It is an 8 bit alpha numeric code , the decimal digits are represented by 8421 code preceded by 111

## Error Correction

What is Error?

- ✓ Error is a condition when the output information does not match with the input information.
- ✓ During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from one system to other.
- ✓ That means a 0 bit may change to 1 or a 1 bit may change to 0.



## Parity Method for Error Detection Code

Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted.

To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if an error occurred during transmission of the message.

A simple example of error-detecting code is **parity check**.

A parity bit is attached to a group of bits to make the total number of 1s in a group always even or always odd

### How to Detect and Correct Errors?

- ✓ To detect and correct the errors, additional bits are added to the data bits at the time of transmission.
- ✓ The additional bits are called **parity bits**. They allow detection or correction of the errors.
- ✓ The data bits along with the parity bits form a **code word**.
- ✓ There are 2 types of parity, even parity & odd parity
- ✓ Even parity : no. of 1's should be even including parity bit (Fig.(a))
- ✓ Odd parity : no. of 1's should be odd including parity bit (Fig.(b))

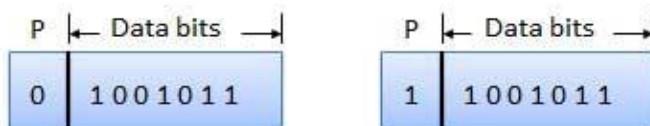


Fig. (a)

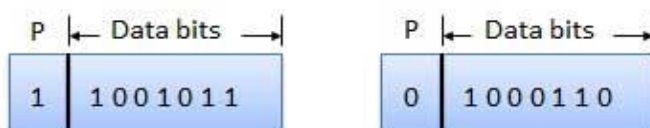


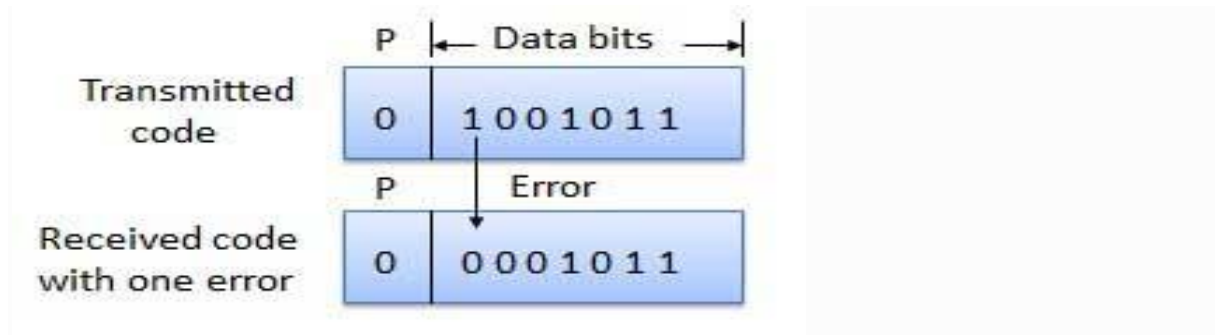
Fig. (b)

### How Does Error Detection Take Place?

- ✓ Parity checking at the receiver can detect the presence of an error if the parity of the receiver signal is different from the expected parity.
- ✓ That means, if it is known that the parity of the transmitted signal is always going to be "even" and if the received signal has an odd parity, then the receiver can conclude that the received signal is not correct.



- ✓ If an error is detected, then the receiver will ignore the received byte and request for retransmission of the same byte to the transmitter.



### Error correction --- Hamming Code

- ✓ Hamming code is a set of error-correction codes that can be used to detect and correct the errors that can occur when the data is moved or stored from the sender to the receiver.
- ✓ It is a technique developed by R.W. Hamming for error correction
- ✓ It is a 7 bit code in which 4 data bits can be transmitted
- ✓ This code contains 4 data bits (D3, D5, D6, D7) and 3 parity bits (P4, P2, P1)

D7      D6      D5      P4      D3      P2      P1

(1) Construct the even-parity seven-bit Hamming code to transmit the data bits 1101.

**Solution :**

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	P <sub>4</sub>	D <sub>3</sub>	P <sub>2</sub>	P <sub>1</sub>
1	1	0		1		

- For P<sub>1</sub> bit, examine the number of 1's in D<sub>3</sub>, D<sub>5</sub> and D<sub>7</sub>. There are two number of 1's. Hence P<sub>1</sub> bit is 0.
- For P<sub>2</sub> bit, examine the number of 1's in D<sub>3</sub>, D<sub>6</sub> and D<sub>7</sub>. There are three number of 1's. Hence P<sub>2</sub> bit is 1.
- For P<sub>4</sub> bit, examine the number of 1's in D<sub>5</sub>, D<sub>6</sub> and D<sub>7</sub>. There are two number of 1's. Hence P<sub>4</sub> bit is 0.

Therefore, the final code is ⇒

⇒ 1100110

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	P <sub>4</sub>	D <sub>3</sub>	P <sub>2</sub>	P <sub>1</sub>
1	1	0	0	1	1	0

