

**KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY**

**Housefed Complex, Dispur, Guwahati - 781 006**



**Master of Computer Applications**

**DIGITAL LOGIC**

**CONTENTS**

**UNIT 1 : NUMBER SYSTEMS**

**UNIT 2 : BOOLEAN ALGEBRA**

**UNIT 3 : LOGIC GATES**

**UNIT 4 : COBINATIONAL CIRCUITS**

**UNIT 5 : SEQUENTIAL CIRCUITS**

**UNIT 6 : MEMORY ORGANIZATION**

---

## Subject Experts

---

1. Prof. Anjana Kakati Mahanta, Deptt. of Computer Science, Gauhati University
2. Prof. Jatindra Kr. Deka, Deptt. of Computer Science and Engineering, Indian Institute of Technology, Guwahati
3. Prof. Diganta Goswami, Deptt. of Computer Science and Engineering, Indian Institute of Technology, Guwahati

---

## Course Coordinators

---

Tapashi Kashyap Das, Assistant Professor, Computer Science, KKHSOU

Arabinda Saikia, Assistant Professor, Computer Science, KKHSOU

---

## SLM Preparation Team

---

UNITS	CONTRIBUTORS
2, 3	Prof. Jyotiprakash Goswami Deptt. of Computer Applications, Assam Engineering College, Guwahati, Assam
4, 5	Chakradhar Das, Lecturer (Selection Grade), Deptt. of Electrical Engineering, Bongaigaon Polytechnique, Assam
1, 6	Sangeeta Kakoty, Lecturer, Deptt. of Computer Science, Jagiroad College, Assam

---

**July, 2011**

© Krishna Kanta Handiqui State Open University.

No part of this publication which is material protected by this copyright notice may be produced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior written permission from the KKHSOU.

The university acknowledges with thanks the financial support provided by the <b>Distance Education Council, New Delhi</b> , for the preparation of this study material.
---

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University.

---

Housefed Complex, Dispur, Guwahati- 781006; Web: [www.kkhsou.org](http://www.kkhsou.org)

---

## COURSE INTRODUCTION

---

This is a course on **Digital Logic**. Digital logic has fascinated many people over the years. Everything in the digital world is based on the binary number system. Numerically, this involves only two symbols: 0 and 1. Digital Logic is a method by which electrical circuits are provided with a limited ability to make decisions. The most common use of digital logic today is in the control and arithmetic functions of digital computers, without which modern life would grind to a halt.

The course consists of six units :

The *first* unit discusses various number systems like decimal, binary, octal, hexadecimal and their conversion from one form to another. The unit also includes the methods of addition and subtraction of binary numbers, complements and fixed/floating point representations. Concept of BCD, ASCII, EBCDIC, Gray code etc are discussed at the end.

The *second* unit is on Boolean Algebra. The unit discusses various concept associated with Boolean Algebra like Boolean operators, Boolean expression, representation of Boolean expression in Canonical form, Karnaugh Map, Don't care condition etc.

The *third* unit discusses various logic gates, their conversion, truth tables, and the most important De-Morgan's theorem.

The *fourth* unit focuses on combinational circuits. This unit gives us the concept various adders and subtractors, multiplexers, demultiplexers, encoders, decoders etc. with their applications.

The *fifth* unit deals with sequential circuits which includes the concept of flip-flops, counters, registers.

The *sixth* unit is the last unit of this course which is on memory organization. This unit gives us the concept of RAM and its types, 2D and 3D organization of RAM, ROM, types of ROM, organization of simple ROM cell etc.

Each unit of this course includes some along-side boxes to help you know some of the difficult, unseen terms. Some "EXERCISES" have been included to help you apply your own thoughts. You may find some boxes marked with: "LET US KNOW". These boxes will provide you with some additional interesting and relevant information. Again, you will get "CHECK YOUR PROGRESS" questions. These have been designed to self-check your progress of study. It will be helpful for you if you solve the problems put in these boxes immediately after you go through the sections of the units and then match your answers with "ANSWERS TO CHECK YOUR PROGRESS" given at the end of each unit.

# MASTER OF COMPUTER APPLICATIONS

## Digital Logic

### DETAILED SYLLABUS

	Marks	Page No.
<b>UNIT 1 : Number Systems</b>	<b>15</b>	<b>5-44</b>
Decimal, Binary, Hexadecimal and Octal. It's Conversion: Decimal to Binary/ Hexadecimal/Octal and vice versa. Addition/ Subtraction on Binary Numbers, Complement: r's and (r-1)'s complement. Fixed Point representation and Floating point representation, BCD, ASCII, EBCDIC, Gray code.		
<b>UNIT 2 : Boolean Algebra</b>	<b>15</b>	<b>45-60</b>
Boolean operators, Rules (postulates and basic theorems) of Boolean algebra, Dual and complement of Boolean expression, representation of Boolean expression in Canonical form, Boolean expression and their simplification by algebraic method and Karnaugh Map, Don't care condition.		
<b>UNIT 3 : Logic Gates</b>	<b>15</b>	<b>61-79</b>
Logic Gates(OR, AND, NOT, NAND, NOR, XOR, XNOR), Truth Tables, De-Morgan's theorem, Conversion of the logic gates.		
<b>UNIT 4 : Combinational Circuits</b>	<b>20</b>	<b>80-113</b>
Introduction to Combinational Circuits; Half-adder, Full-adder, Binary Parallel Adder, 4-bit Binary Parallel Adder, Serial Adder; Half-subtractor, Full-subtractor; Multiplexer: Basic 2-Input Multiplexer, 4-Input-Multiplexer, 8-to-1 Multiplexer, 16-to-1 Multiplexer, Multiplexer Applications; Demultiplexer: Parallel-to-Serial Converter, Data Distributors, 1-to-4 Demultiplexer; Encoder: Octal-to-Binary Encoder, Decimal-to-BCD Encoder; Decoder: Basic Binary Decoder, 3-line-to-8-Line Decoder, Magnitude Comparator		
<b>UNIT 5 : Sequential Circuits</b>	<b>20</b>	<b>114-141</b>
Sequential Circuits; Flip-Flops: RS, D, JK, MS; Counters: Asynchronous , Synchronous; Registers and its types, Shift Registers: Serial in-Serial out Registers, Shift-Left Register, Shift-Right Register, Serial-in-Parallel-out Shift Registers, Parallel-in-Serial-out Shift Registers, Parallel-in-Parallel-out Register, Applications of Shift Registers.		
<b>UNIT 6 : Memory Organization</b>	<b>15</b>	<b>142-158</b>
Random Access Memory : Types of RAM; Static RAM : Static RAM cell and its structure; DRAM : basic structure of DRAM; Organization of RAM : 2D organization, 3D organization; ROM : Types of ROM, organization of simple ROM cell.		

---

# UNIT 1 : NUMBER SYSTEMS

---

## UNIT STRUCTURE

- 1.1 Learning Objectives
- 1.2 Introduction
- 1.3 Number System
  - 1.3.1 Decimal Number System
  - 1.3.2 Binary Number System
  - 1.3.3 Octal Number System
  - 1.3.4 Hexadecimal Number System
- 1.4 Number System Conversion
  - 1.4.1 Binary to Decimal Conversion
  - 1.4.2 Decimal to Binary Conversion
  - 1.4.3 Octal to Decimal conversion
  - 1.4.4 Decimal to Octal Conversion
  - 1.4.5 Hexadecimal to decimal Conversion
  - 1.4.6 Decimal to Hexadecimal Conversion
- 1.5 Complement of Numbers
  - 1.5.1  $(r-1)$ 's Complement
  - 1.5.2  $r$ 's complement
- 1.6 Data Representation
  - 1.6.1 Fixed Point Representaion
  - 1.6.2 Floating Point Representation
- 1.7 Binary Arithmetic
  - 1.7.1 Addition
  - 1.7.2 Subtraction
  - 1.7.3 Multiplication
  - 1.7.4 Division
- 1.8 Computer Codes
  - 1.8.1 BCD
  - 1.8.2 ASCII
  - 1.8.3 EBCDIC
  - 1.8.4 Gray Code

- 1.9 Let Us Sum Up
- 1.10 Answers to Check Your Progress
- 1.11 Further Readings
- 1.12 Model Questions

---

## 1.1 LEARNING OBJECTIVES

---

After going through this unit, you will be able to :

- define and describe number system
- identify how data is represented in computers
- convert a number from one number system to another
- describe how computers perform binary arithmetic
- describe the different types computer codes

---

## 1.2 INTRODUCTION

---

Number system is a fundamental concept used in micro computer system. They are of different types and can represent by some digit symbols. The knowledge of binary, octal and hexadecimal number system is essential to understand the operation of a computer. This unit deals with all this system. In this unit we will discuss about all the representation of this number system and their conversion from one number system to its equivalent other number system. The arithmetic operations of all the system is very much important to know how a system can operate our data inside. This is also discussed in this unit. In addition you will get the internal data representation method which is called computer codes. Different computer codes like BCD code, ASCII, EBCDIC etc are introduced here in this unit.

---

## 1.3 NUMBER SYSTEM

---

We are familiar with the decimal number system which is used in our day-to-day work. Ten digits are used to form decimal number. To represent these decimal digits, ten separate symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 are used. But a digital computer stores, understands and manipulates

information composed of only zeros and ones. So, each decimal digits, letters, symbols etc. written by the programmer (an user) are converted to binary codes in the form of 0's and 1's within the computer. The number system is divided into different categories according to the base (or radix) of the system as binary, octal and hexadecimal. If a number system of base  $r$  is a system, then the system have  $r$  distinct symbols for  $r$  digits. The knowledge of the number system is essential to understand the operation of a computer.

### 1.3.1 Decimal Number System

Decimal no. system have ten digits represented by 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. So, the base or radix of such system is 10.

In this system the successive position to the left of the decimal point represent units, tens, hundreds, thousands etc. For example, if we consider a decimal number 1257, then the digit representations are :

1	2	5	7
↓	↓	↓	↓
thousands	hundreds	tens	units
positions	position	position	position

The weight of each digit of a number depends on its relative position within the number.

#### Example 1.1 :

The weight of each digit of the decimal no. 6472

$$6472 = 6000 + 400 + 70 + 2 = 6 \times 10^3 + 4 \times 10^2 + 7 \times 10^1 + 2 \times 10^0$$

The weight of digits from right hand side are :

$$\text{Weight of 1st digit} = 2 \times 10^0$$

$$\text{Weight of 2nd digit} = 7 \times 10^1$$

$$\text{Weight of 3rd digit} = 4 \times 10^2$$

$$\text{Weight of 4th digit} = 6 \times 10^3$$

The above expressions can be written in general forms as the weight of  $n^{\text{th}}$  digit of the number from the right hand side :



**Decimal Number System** uses 10 digits from 0 to 9 to represents its system.

$$= n^{\text{th}} \text{ digit} \times 10^{n-1}$$

$$= n^{\text{th}} \text{ digit} \times (\text{base})^{n-1}$$

The no. system in which the weight of each digit depends on its relative position within the number is called positional number system. The above form of general expression is true only for positional number system.



A **Binary Number System** uses only digit 0 and 1

### 1.3.2 Binary Number System

Only two digits 0 and 1 are used to represent the binary number system. So the base or radix is two (2). The digits 0 and 1 are called bits (Binary Digits). In this number system the value of the digit will be two times greater than its predecessor. Thus the value of the places are :

$$\leftarrow \leftarrow 32 \leftarrow 16 \leftarrow 8 \leftarrow 4 \leftarrow 2 \leftarrow 1$$

The weight of each binary bit depends on its relative position within the number. It is explained by the following example--

#### Example 1.2 :

The weight of bits of the binary number 10110 is :

$$= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

$$= 16 + 0 + 4 + 2 + 0 = 22 \text{ (decimal number)}$$

The weight of each bit of a binary no. depends on its relative pointer within the no. and explained from right hand side as :

$$\text{Weight of 1st bit} = 1^{\text{st}} \text{ bit} \times 2^0$$

$$\text{Weight of 2nd bit} = 2^{\text{nd}} \text{ bit} \times 2^1$$

.....  
.....

and so on.

The weight of the  $n^{\text{th}}$  bit of the number from right hand side

$$= n^{\text{th}} \text{ bit} \times 2^{n-1}$$

$$= n^{\text{th}} \text{ bit} \times (\text{Base})^{n-1}$$

It is seen that this rule for a binary number is same as that for a decimal number system. The above rule holds good for any other



positioned number system. The weight of a digit in any positioned number system depends on its relative position within the number and the base of the number system.

Table 1.1 shows the binary equivalent numbers for decimal digits.

**Table 1.1 : Binary equivalent of decimal numbers**

Decimal Number	Equivalent Binary Number
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

**Binary Fractions :** A binary fractions can be represented by a series of 1s and 0s to the right of a binary point. The weight of digit positions to the right of the binary point are given by  $2^{-1}$ ,  $2^{-2}$ ,  $2^{-3}$  and so on.

**Example 1.3 :** Show the representation of binary fraction 0.1101.

**Solution :** The binary representation of 0.1101 is :

$$\begin{aligned}
 0.1101 &= 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \\
 &= 1 \times 0.5 + 1 \times 0.25 + 0 \times 0.125 + 1 \times 0.0625 \\
 &= 0.8125
 \end{aligned}$$

$$\text{So, } (0.1101)_2 = (0.8125)_{10}$$

### 1.3.3 Octal Number System

A commonly used positional number system is the Octal Number System. This system has eight (8) digit representation as 0,1,2,3,4,5,6 and 7. The base or radix of this system is 8. The values increase from left to right as 1,8,64,512, 4096 etc. The decimal



**Octal Number System**  
uses 8 digits from 0 to 7.

value 8 is represented in octal as 10, 9 as 11, 10 as 12 and so on. As  $8=2^3$ , an octal number is represented by a group of three binary bits. For example 3 is represented as 011, 4 as 100 etc.

**Table 1.2 The octal number and their binary representations.**

Decimal Number	Octal Number	Binary Coded Octal No.
0	0	000
1	1	001
.	.	.
.	.	.
7	7	111
8	10	100 000
15	17	001 111



**Hexadecimal System**  
groups numbers by 16  
and power of 16.

### 1.3.4 Hexadecimal Number System

The hexadecimal number system is now extensively used in computer industry. Its base (or radix) is 16, ie. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The hexadecimal numbers are used to represent binary numbers because of ease of conversion and compactness.

As  $16 = 2^4$ , hexadecimal number is represented by a group of four binary bits. For example, 5 is represented by 0101. Table 2.3 shows the binary equivalent of a decimal number and its hexadecimal representation.

**Table 1.3 : Hexadecimal number and their Binary representation**

Decimal No.	Hexadecimal No.	Binary coded Hex. No
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101

6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

## 1.4 NUMBER SYSTEM CONVERSION

As the computer uses different number systems, there is a process of converting generally used decimal number systems to other number systems and vice-versa.

### 1.4.1 Binary to Decimal Conversion

To convert a binary number to its decimal equivalent we use the following expression. The weight of the  $n^{\text{th}}$  bit of the number from right hand side

$$= n^{\text{th}} \text{ bit} \times 2^{n-1}$$

First we mark the bit position and then we give the weight of each bit of the number depending on its position. The sum of the weight of all bits gives the equivalent number.

**Example 1.4 :** Convert binary  $(100101)_2$  to its decimal equivalent.

$$\begin{aligned}
 \text{Solution : } (100101)_2 &= 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^0 \\
 &= 32 + 0 + 0 + 4 + 0 + 1 \\
 &= 37
 \end{aligned}$$

$$\text{So, } (100101)_2 = (37)_{10}$$

Mixed number contain both integer and fractional parts and can convert to its decimal equivalent is as follows :

**Example 1.5 :** Converting  $(11011.101)_2$  to its equivalent decimal no.

**Solutaion :**

$$\begin{aligned}
 (11011.101)_2 &= (1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) + \\
 &\quad (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}) \\
 &= (16 + 8 + 0 + 1) + (0.5 + 0 + 0.125) = 27.625 \\
 \text{So, } (11011.101)_2 &= (27.625)_{10}
 \end{aligned}$$

### 1.4.2 Decimal to Binary Conversion

There are different methods used to convert decimal number to binary numbers. The most common method is, repeatedly divide the decimal number to binary number by 2, then the remainders 0's and 1's obtained after division is read in reverse order to obtain the binary equivalent of the decimal number. This method is called **double-double method**.

**Example 1.6 :** Convert  $(75)_{10}$  to its binary equivalent.

<b>Solution :</b>	2   75		Remainder
	2   37	LSB	1
	2   18		1
	2   9		0
	2   4		1
	2   2		0
	1	MSB	0

$$\text{So, } (75)_{10} = (001011)_2$$

**Example 1.7 :** Convert decimal fraction  $(25.625)_{10}$  to its equivalent binary no.

<b>Solution :</b>	2   25	Remainder	MSB	0.625
	2   12	1		$\times 2$
	2   6	0		1.250
	2   3	0		$\times 2$
	1	1		0.500
	$(25)_{10} = (11001)_2$			$\times 2$
				1.000

$$(0.625)_{10} = (0.101)_2$$

$$\text{So, } (25.625)_{10} = (11001.101)_2$$

### 1.4.3 Octal to Decimal Conversion

The method of converting octal numbers to decimal numbers is simple. The decimal equivalent of an octal number is the sum of the numbers multiplied by their corresponding weights.

**Example 1.8 :** Find decimal equivalent of octal number  $(153)_8$

**Solution :**  $1 \times 8^2 + 1 \times 8^1 + 1 \times 8^0 = 64 + 40 + 3 = 107$

So,  $(153)_8 = (107)_{10}$

The fractional part can be converted by multiplying it by the negative powers of 8 as shown in the following example.

**Example 1.9 :** Find decimal equivalent of octal number  $(123.21)_8$

**Solution :**  $(1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0) + (2 \times 8^{-1} + 1 \times 8^{-2})$   
 $= (64 + 16 + 3) + (0.25 + 0.0156) = 83.2656$

So,  $(123.21)_8 = (83.2656)_{10}$

### 1.4.4 Decimal to Octal Conversion

The procedure for conversion of decimal numbers to octal numbers is exactly similar to the conversion of decimal number to binary numbers except replacing 2 by 8.

**Example 1.10 :** Find the octal equivalent of decimal  $(3229)_{10}$

**Solution :**                      Remainders

8   4121			
8   515	1	↑ read from MSB to LSB	
8   64	3		
8   8	0		
1	0		

So,  $(4121)_{10} = (10031)_8$

The fractional part is multiplied by 8 to get a carry and a fraction as shown in the following example.

**Example 1.11 :** Find the octal equivalent of  $(.123)_{10}$

**Solution :** Octal equivalent of fractional part of a decimal number as follows :

$$\begin{array}{rcl}
 8 \times 0.123 = 0.984 & 0 & \\
 8 \times 0.984 = 7.872 & 7 & \downarrow \text{read from LSB} \\
 8 \times 0.872 = 6.976 & 6 & \downarrow \text{to MSB} \\
 8 \times 0.976 = 7.808 & 7 & 
 \end{array}$$

Read the integer to the left of the decimal point.

The calculation can be terminated after a few steps if the fractional part does not become zero.

The octal equivalent of  $(0.123)_{10} = (0.0767)_8$

**NOTE :** The octal to binary and binary to octal conversion is very easy. Since, 8 is the third power of 2, we can convert each octal digit into its three-bit binary form and vice versa.

**Example 1.12 :** Convert  $(567)_8$  to its binary form.

$$\begin{array}{ccc}
 \text{Solution :} & 5 & 6 & 7 \\
 & \downarrow & \downarrow & \downarrow \\
 & 101 & 110 & 111
 \end{array}$$

$$\text{So, } (567)_8 = (101\ 110\ 111)_2$$

Conversion from binary to octal is just opposite of the above example.

### 1.4.5 Hexadecimal to Decimal Conversion

The method of converting Hexadecimal numbers to decimal number is simple. The decimal equivalent of an Hexadecimal number is the sum of the numbers multiplied by their corresponding weights.

**Example 1.13 :** Find the decimal equivalent of  $(4A8C)_{16}$

**Solution :**

$$\begin{aligned}
 (4A8C)_{16} &= (4 \times 16^3) + (10 \times 16^2) + (8 \times 16^1) + (12 \times 16^0) \\
 &= 16384 + 2560 + 128 + 12 \\
 &= (19084)_{10}
 \end{aligned}$$

$$(4A83)_{16} = (19084)_{10}$$

**Example 1.14 :** Find the decimal equivalent of  $(53A.0B4)_{16}$

**Solution :**

$$\begin{aligned}
 (53A.0B4)_{16} &= (5 \times 16^2) + (3 \times 16^1) + (10 \times 16^0) + (0 \times 16^{-1}) \\
 &\quad + (11 \times 16^{-2}) + (4 \times 16^{-3})
 \end{aligned}$$

$$\begin{aligned}
 &= 1280 + 48 + 10 + 0 + 0.04927 + 0.0009765 \\
 &= (1338.0439)_{10} \\
 (53A.0B4)_{16} &= (1338.0439)_{10}
 \end{aligned}$$

### 1.4.6 Decimal to Hexadecimal Conversion

The procedure for conversion from decimal no. and decimal fraction no. to hexadecimal equivalent is exactly similar to the conversion of decimal to binary no. except replacing 2 by 16.

**Example 1.15 :** Convert decimal  $(1234.675)_{10}$  to hexadecimal.

**Solution :** 1st consider  $(1234)_{10}$

	Remainder	
	Decimal	Hexadecimal
16 <u>1234</u>	2	2
16 <u>77</u>	13	D
16 <u>4</u>	4	4

$$(1234)_{10} = (4D2)_{16}$$

Conversion of  $(0.675)_{10}$ :

	Decimal	Hexadecimal
$0.675 \times 16 = 10.8$	10	A
$0.800 \times 16 = 12.8$	12	C
$0.800 \times 16 = 12.8$	12	C
$0.800 \times 16 = 12.8$	12	C

$$(0.675)_{10} = (0.ACC)_{16}$$

$$\text{Hence } (1234.675)_{10} = (4D2.ACC)_{16}$$

If the decimal number is very large, it is tedious to convert the number to binary directly. So it is always advisable to convert the number into hex first, and then convert the hex to binary.



### CHECK YOUR PROGRESS

Q.1. What is the largest number that can be represented using 8 bits?

.....

Q.2. What is the weight of 1 in  $(10000)_2$ .

.....

Q.3. Convert the following:

a)  $(565.25)_{10}$  to its equivalent binary number.

b)  $(256.24)_8$  to decimal equivalent.

c)  $(A3B.BB)_{16}$  to decimal equivalent.

d)  $(10010.110)_2$  to decimal equivalent.

e)  $(3964.63)_{10}$  to octal equivalent.

## 1.5 COMPLEMENT OF NUMBERS

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. The complement of a binary number is obtained by inverting its all the bits.

For example, the complement of 10011 is 01100 and 00101 is 11010 etc. The complement again depends on the base of the number.

There are two types of complements for a number of base  $r$ . These are :

- $r$ 's complement and
- $(r-1)$ 's complement,

For example, for decimal numbers the base is 10. Therefore, complements will be 10's complement and  $(10-1)=9$ 's complement. For binary numbers, the complements are 2's complement and 1's complement since base is 2.



### 1.5.1 (r-1)'s Complement

Given a number  $N$  in base  $r$  having  $n$  digits, the  $(r-1)$ 's complement of  $N$  is defined as  $(r^n - 1) - N$ .

**9's Complement** : For decimal numbers  $r=10$  and  $r-1=9$ , so the 9's complement of  $N$  is  $(10^n - 1) - N$ .

For example, with  $n = 4$  we have  $10^4 = 10000$  and  $10^4 - 1 = 9999$ . It follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9.

For example, 9's complement of 49 is  $(99 - 49) = 50$

9's complement of 127 is  $(999 - 127) = 872$

**1's Complement** : For binary numbers,  $r = 2$  and  $(r-1) = 1$ , so the 1's complement of  $N$  is  $(2^n - 1) - N$ . Again,  $2^n$  is represented by a binary number that consists of a 1 followed by  $n$  0's.  $2^n - 1$  is a binary number represented by  $n$  1's. For example, with  $n = 4$ , we have  $2^4 = (10000)_2$  and  $2^4 - 1 = (1111)_2$ . Thus the 1's complement of a binary number is obtained by subtracting each digit from 1.

However, the subtraction of a binary digit from 1 causes the bit to change from 0 to 1 or from 1 to 0. Therefore, the 1's complement of a binary number is formed by changing 1's into 0 and 0's into 1's. For example, 1's complement of 1010111 is 0101000.



The 7's and 15's complement of a number is found by subtracting each digit of the number from 7 and 15 respectively.

### 1.5.2 r's Complement

The  $r$ 's complement of a  $n$ -digit number  $N$  in base  $r$  is defined as  $r^n - N$  for  $N \neq 0$  and 0 for  $N = 0$ . Comparing with the  $(r-1)$ 's complement, we note that the  $r$ 's complement is obtained by adding 1 to the  $(r-1)$ 's complement.

**10's Complement** : The 10's complement of a decimal number is equal to the 9's complement of the number plus 1

i.e. 10's complement of decimal number = its 9's complement + 1

So, 10's complement of 49 is  $(99 - 49) + 1 = 50 + 1 = 51$

10's complement of 127 is  $(999 - 127) + 1 = 872 + 1 = 873$



Like 8's complement and 16's complement of a number is found by adding 1 to the LSB of the 7's and 15's complement of an octal and hexadecimal number respectively.

**2's Complement :** It is obtained by adding 1 in the 1's complement form of the binary numbers.

i.e. 2's complement of binary number = 1's complement of that number + 1

For example, 2's complement of 1010111 is  $0101000 + 1 = 0101001$



### CHECK YOUR PROGRESS

Q.4. Find 9's complement 10's complement of decimal numbers 44 and 182.

.....  
 .....

Q.5. Find 1's complement and 2's complement of binary numbers 1101001 and 0000.

.....  
 .....

## 1.6 DATA REPRESENTATION

Data are usually represented by using the alphabets A to Z, numbers 0 to 9 and various other symbols. This form of representation is used to formulate problem and fed to the Computer. The processed output is required in the same form. This form of representation is called *external data representation*. However, the computer can understand data only in the form of 0's and 1's. The method of data representation in a form suitable for storing in the memory and for processing by the CPU is called the internal data representation on digital computer.

Data, in general, are of two types : Numeric and non-numeric (Character data). The numeric data deals only with numbers and arithmetic operations and non numeric data deals with characters, names addresses etc. and non-arithmetic operations.

### 1.6.1 Fixed Point Representaion

A fixed point numbers in binary system uses a sign bit. A positive number has a sign bit 0 while the negative number has a sign bit 1. A negative number can be represented in one of the following ways.

- Signed magnitude representaion
- Signed 1's complement representaion
- Signed 2's complement representaion

Assume that the size of the register is 7 bit and the 8<sup>th</sup> position bit in used for error checking and correction or other purposes.

#### a) Signed magnitude representation

	+6	–6	
	000110	1000110	
↑		↑	No change in the
Sign bit		Sign bit	magnitude, only the
			sign bit changes

#### b) Signed 1's complement representation

	+6	–6
0	000110	111001

Here 0 and 1 are sign bits. 1's complement is getting for the –ve integer is by taking complement of all the bits of +ve no. including sign bit.

#### c) Signed 2's complement representation

	+6	–6	
0	000101	111011	
↑		↑	2's complement of the
Sign bit		Sign bit	positive number
			including sing bit

The signed magnitude system is easier to interpret but computer arithmetic with this is not efficient. The circuits for handling numbers are simplified if 1's or 2's complement systems are used and as a result one of these is almost always adopted.

**Note 1 :** In 1's and 2's complements, all positive integers are represented as sign magnituded system.

**Note 2 :** When all the bits of the computer word are used to represent the number and no bit is used for signed representation, it is called unsigned representation of the number.

### 1.6.2 Floating Point Representation

A number which has both an integer part as well as a fractional part is called real number or floating point number. A floating point number is either positive or negative. Examples of real decimal numbers are 156.65, 0.893, -235.75, -0.253 etc. Examples of binary real numbers are 101.101, 0.11101, -1011.101, -0.1010 etc.

The first part of the number is a fixed point number which is called mantissa. It can be an integer or a fraction.

The second part specifies the decimal or binary point position and is termed exponent. It is not a physical point. Therefore, whenever we are representing a point and is termed as an exponent. It is only the assumed position. For example, for decimal o. +15.37, the typical floating point notation is :

$$51.47 = 0.5147 \times 10^2 \text{ or } 5147 \times 10^{-2}$$

Now, The floating point representation of  $0.5147 \times 10^3$  is :

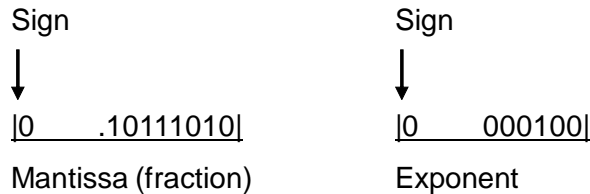
Sign	Sign
↓	↓
0 .5147	0 02
Mantissa (fraction)	Exponent

The floating point representation of  $5147 \times 10^{-2}$  is

Sign	Sign
↓	↓
0 5147	0 02
Mantissa (Integer)	Exponent

Similarly for example a floating point binary number 1011.1010 can be represented as :  $1011.1010 = 0.10111010 \times 2^4$

This can be represented in a 16 bit register as follows



The mantissa occupies 9 bits (1 bit for sign and 8 bits for value) and the exponent 7 bits (1 bit for sign and 6 bits for value). The binary point (.) is not physically indicated in the register, but it is only assumed (position) to be there.

In general form, the floating point numbers is expressed as :

$$N = M \times R^e$$

Where, M – Mantissa

R – Radix (or base)

e – Exponent

The mantissa M and exponent e are physically present in register. But the radix R and the point (decimal or binary point) are not indicated in the register. There are only assumed for computation and manipulation.

**Normalized Floating point Number :** Floating point numbers are often represented in normalized forms. A floating point number where mantissa does not contain zero as the most significant digit of the number is considered to be in normalized form. For example,  $0.00038695 \times 10^5$  and  $0.0589 \times 10^{-4}$  are not normalized numbers. But  $0.38695 \times 10^2$  and  $0.589 \times 10^{-5}$  are normalized numbers. Similarly, for binary number also,  $0.0011001 \times 2^8$  and  $0.0001011 \times 2^{-5}$  are not non-normalized binary numbers. But  $0.11001 \times 2^6$  and  $0.1011 \times 2^{-8}$  are normalized binary numbers.

A zero cannot be normalized as all the digits in the mantissa is zero.

Arithmetic operations involved with floating point numbers are more complex. It takes larger time for execution and requires complex hardware. But floating point representation is frequently used in scientific calculations.

**Overflow and Underflow :** When the result is too small to be presented by the computer, an overflow or underflow condition exists. When two floating-point numbers of the same sign are added, a carry may be generated out of high-order bit position. This is known as mantissa overflow. In case of addition or subtraction floating point numbers are aligned. The mantissa is shifted right for the alignment of a floating point number. Sometimes, the low order bits are lost in the process of alignment. This is referred as mantissa underflow. To perform the multiplication of two floating point numbers, the exponents are added. In certain cases the sum of the exponents maybe too large and it may exceed the storing capacity of the exponent field. This is called exponent overflow. In case of division the exponent of the divisor is subtracted from the exponent of the dividend. The result of subtraction may be too small to be represented. This is called exponent underflow.

Overflow or underflow resulting from a mantissa operation can be corrected by shifting the mantissa of the result and adjusting the exponent. But the exponent overflow or underflow can not be corrected and hence, an error indication has to be displayed on the computer screen.



### CHECK YOUR PROGRESS

Q.6. Represent 10 by the following fixed point representation method.

- a) Signed magnitude representation.
- b) Signed 1's complement representation.
- c) Signed 2's complement representation.

Q.7. Represent  $(1010.1010)_2$  with floating point representation method.

.....

## 1.7 BINARY ARITHMETIC

### 1.7.1 Addition

Binary addition is performed in the same manner as decimal addition. Since, in binary system only two digit 0s and 1s are used, the addition will be like—

$$0 + 0 = 0$$

$$0 + 1 = 1 = 1 + 0$$

$$1 + 1 = 0, \text{ Carry } 1 \text{ to the next left column}$$

$$1 + 1 + 1 = 1, \text{ Carry } 1 \text{ to the next column.}$$

Carry overs are performed in the same manner as in decimal arithmetic.

**Example 1.16 :** Add the binary numbers

(i) 1011 and 1001

(ii) 10.011 and 1.001

**Solution :**

(i)	Binary no.	Equivalent decimal no.
	11    carry	
	1011	11
	<u>+1001</u>	<u>9</u>
	10100	20
(ii)	10.011	2.375
	<u>1.001</u>	<u>1.125</u>
	11.100	3.500

Since the circuit in all digital systems actually can handle two numbers to performs addition, it is not necessary to consider the addition of more than two binary numbers. When more than two numbers to be added, the first two are added first and then their sum is added to the third and so on.

The complexity may rise when to add combination of positive and negative binary number. In this case the arithmetic addition is dependent on the representation of

- a) Signed magnitude
- b) Signed 1's complement
- c) Signed 2's complement

This will be more clear if we discuss through the following example:

**Example 1.17 :** Add 25 and -30 in binary using 7 bit register in signed magnitude representation

- a) Signed 1's complement representaiton
- b) Signed 2's complement representaiton

**Solution :** Here, 25 is  $+25 = 0011001$  in binary system  
 $-30 = 1011110$  in binary system

To do the arithmetic addition with one negative number we have to check the magnitude of the numbers. The number having smaller magnitude is there subtracted from the bigger number and the sign of bigger number is selected. To implement such a scheme in digits, hardware will require a long sequence of control decisions as well as circuits that will add, compare and subtract numbers. The better attentative of arithmetic with one negative number is signed 2's complement.

**In signed 2's complement representation :**

We get that  $+30$  is  $0 \ 011110$

$-30$  is  $1 \ 011110$

Now, 2's complement of  $-30$  (including sign bit)  $1 \ 100010$

$+25$  is  $0 \ 011001$

Addition

$+25 \quad 0 \quad 011 \quad 001$

$\underline{-30} \quad 1 \quad \underline{100} \quad \underline{010}$

$-05 \quad 1 \quad 111 \quad 011$  (Just add the numbers)

The result for negative number will store n signed 2's complement form. So the above result in signed 2's complement form. So the above result in signed 2's complemnt form including sign bit is

$1 \quad 000 \quad 100 \quad +1 \quad = \quad 1 \quad 000 \quad 101$

Which is -05 in decimal system.



From the above example it is noticed that, signed 2's complement representation is simpler than signed magnitude representation. This procedure requires only one central decision and only one circuit for adding the two numbers. But it puts additional condition that the negative numbers should be stored in signed 2's complement form in the register. This can be achieved by complementing the positive number bit by bit then incrementing the resultant by 1 to get signed 2's complement.

**In signed 1's complement representation :** This method is also simple. The rule is that, add the two numbers including the sign bit. If carry of the most significant bit or sign bit is one, then increment the result by 1 and discard the carry over.

Addition :

$$\begin{array}{rcl} +25 & = & 0 \ 011 \ 001 \\ -30 & = & \underline{1} \ \underline{100 \ 001} \text{ (1's complement of } -30) \\ -5 & = & 1 \ 111 \ 010 \end{array}$$

The result will store in 1's complement format. So, 1111 010 in 1's complement format including the sign bit is 1 000 101 which is the required result.

**Example 1.18 :** Add -25 and +30 using 7-bit register.

**Solution :**

$$\begin{array}{rcl} -25 & 1 & 100 \ 110 \text{ (1's complement of 25)} \\ +30 & \underline{0} & \underline{011} \ \underline{110} \\ +5 & 1 \ 0 & 000 \ 100 \end{array}$$

↑

Carry bit, so add 1 to the sum and discard the carry.

This sum is now = 0 000 101 which is +5

**Example 1.19 :** Add -25 and -30 using 7-bit register.

**Solution :**

$$\begin{array}{rcl} -25 & 1 & 100 \ 110 \text{ (1's complement of 25)} \\ -30 & \underline{1} & \underline{100} \ \underline{001} \text{ (1's complement of 30)} \\ -55 & 1 \ 1 & 000 \ 111 \end{array}$$

↑

Carry bit, so add 1 to sum and discard the carry.

Now the sum is = 1 001 000, which is -55

Since, +55 is 0 110 111

So, -55 is in 1's complement 1 001 000

The interesting feature about these representation is the representation of 0 in signed magnitude and 1's complement. There are two representation for zero are :

Signed magnitude	+0	-0
	0 000000	1 000000

Signed 1's complement	0 000000	1 111111
-----------------------	----------	----------

But in signed 2's complement, there is just one zero and there are no positive or negative zero.

+0 000000

-0 in 2's complement is +0 = 1 111111

	1	
		0 000000



discard this carry

Thus, both +0 and -0 are same in 2's complement notation. This is an *added advantage* in favour of 2's complement notation. The maximum number which can be accommodated in registers also depends on the type of representation. In general, in a 8 bit register 1 bit is used as sign. Therefore, the rest of 7 bit are used for representing the value. The value of maximum and minimum number which can be represented are :

For signed magnitude representation  $2^7 - 1$  to  $-(2^7 - 1)$   
 $= 127 - 1$  to  $-(127 - 1)$   
 $= 127$  to  $-127$ ,

which is for signed 1's complement representation. For signed 2's complement representation is from + 127 to -128. The -128 is represented in signed 2's complement notation as 10000000.

### 1.7.2 Subtraction

Though there are other method of performing subtraction, we will consider the method of subtraction known as complementary subtracton. This is a more efficient method of subtraction while using electronics circuits. The following three steps have to follow to subtract binary numbers.

**In 1's complement method :**

1. Find the 1's complement of the number which is subtracting.
2. Add the number which is subtracting from with the complement value obtained from step 1.
3. If there is a carry of 1, add the carry with the result of addition. Else, take complement again of the result and attach a negative sign with the result.

**Example 1.20 :** Subtract  $5 - 6$  by 1's complement method.

**Solution :** 5 Binary equivalent is 101  
               6 Binary equivalent is 110

Step 1 : 1's complement of 6 is 001

Step 2 : Adding 001 with 101 give the result as

$$\begin{array}{r} 001 \\ + 101 \\ \hline 110 \end{array}$$

Step 3 : Since there is no carry in step 2, we take the complement again which will be 001 and after attaching negative sign the required result will be  $-001$  which is  $-1$ .

**In 2's complement method :** It is same as 1's complement method except the step 3. The steps are :

Step 1 : Find the 2's complement of the number which is subtracting.

Step 2 : Add the number which is subtracting from with the complement value obtained from step 1.

Step 3 : If there is a carry of 1, Ignore it. Else, take 2's complement of the result again and attach a negative sign with the result.

**Example 1.21 :** Subtract 5 – 7 by 2's complement method.

**Solution :**     5    Binary equivalent is    101

                  7    Binary equivalent is    111

Step 1 : The 2's complement of 7 is  $000 + 1 = 001$

Step 2 : Adding 001 with 101 will give result as :

     001

101

     110 (No carry)

Step 3 : Since no carry, the 2's complement of 110 is  $001 + 1 = 010$  and attaching a negative sign the required result is  $-10$ .

**Overflow :** An overflow is said to have occurred when the sum of two  $n$  digits number occupies  $(n+1)$  digits. This definition is valid for both binary as well as decimal digits. But what is the significance of overflow for binary numbers since it is not a problem for the cases when we add two numbers? Well, the answer lies in the limits or representation of numbers. Every computer employs a limit for representation number eg. in our examples we are using 8 bit registers of calculating the sum. But what will happen if the sum of the two numbers can be accommodated in 9 bits? Where are we going to store the 9th bit? The problem will be more clear by the following example. In case of a +ve no. added to a –ve no., the sum of result will always be smaller than the two numbers. An overflow always occurs when the added nos. are both +ve or both -ve.

**Example 1.22 :** Add the numbers 65 and 75 in 8 bit register in signed 2's complement notation.

**Solution :**    65    0    1000001

                  75    0    1001011

                  140    1    0001100

This is a -ve number and the 2's complement of the result is equal to -115 which obvious is a wrong result. This has occurred because of overflow.

**Detection of Overflow :** Overflow can be detected as :

If the carry out of the MSBs of number (or, carry into the sign bit) is equal to the carry out of the sign bit then overflow must have occurred. For example

-65	1	0111111	-65	1	0111111
-15	1	1110001	-75	1	0110101
-80	1 1	0110000	-140	1 0	1110100
Carry		= 1	Carry		= 10
Carry from MSB		= 1	Carry from MSB		= 0
Carry from sign bit		= 1	Carry from sign bit		= 1
Sign bit is		= 1	Sign bit is		= 0
No overflow			Therefore, overflow		

Thus, overflow has occurred, i.e. the arithmetic results so calculated have exceeded the capacity of the representation. This overflow also implies that the calculated results might be erroneous.

---

### 1.7.3 Multiplication

---

Multiplication in binary follow the same rules that are followed in the decimal system. The table to be remembered is :

$$\begin{array}{ll} 0 \times 0 = 0 & 1 \times 0 = 0 \\ 0 \times 1 = 0 & 1 \times 1 = 1 \end{array}$$

For example, multiplying  $10101 \times 11001$

$$\begin{array}{r} 10101 \\ \times 11001 \\ \hline 10101 \\ 10101 \\ 10101 \\ \hline 10101 \\ \hline 100001101 \end{array}$$

---

### 1.7.4 Division

---

The process of binary division is same as the decimal long division. In binary division we have:

$$0/1 = 0 \quad 1/1 = 1$$

The steps for binary division are :

1. Start from the left of the divided.
2. Perform subtraction in which the divisor is subtracted from the dividend.
  - a) If subtraction is possible put a 1 in the quotient and subtract the divisor from the corresponding digits of the dividend put a 0 in the quotient
  - b) Bring down the next digit to the right of the remainder.
3. Execute step 2 till there are no more digits left to bring down from the dividend.

**Example 1.23 :** Divide 1011001 by 110

**Solution :**

$$\begin{array}{r}
 \underline{0111} \quad \text{(Quotient)} \\
 \text{(Divisor) } 110 \overline{)1011001} \quad \text{(Dividend)} \\
 \underline{110} \\
 1010 \\
 \underline{110} \\
 1000 \\
 \underline{110} \\
 0101
 \end{array}$$

Here 111 is the answer and 101 is the Remainder.



### CHECK YOUR PROGRESS

Q.8. Add 35 and -40 in binary using 7 bit register in 2's complement representation.

.....

Q.9. Add binary no. 110011.010 and 1000.10

.....

Q.10. Subtract  $(1010101)_2 - (1001001)_2$  by 2's complement method.

.....

Q.11. Multiply  $(11001)_2$  by  $(101)_2$ .

.....

Q.12. Divide  $(101101.101)_2$  by  $(110)_2$ .

.....

## 1.8 COMPUTER CODES

A **code** is a symbol or group of symbols that represents discrete elements. Coding of characters has been standardised to enable transfer of data between computers. *Numeric data* is not the only form of data handled by a computer. We often require to process *alphanumeric data* also. An alphanumeric data is a string of symbols, where a symbol may be one of the letters A, B, C, ..., Z, or one of the digits 0, 1, 2, ..., 9, or a special character, such as +, −, \*, /, ., ( ) = (space for blank) etc. However, the bits 0 and 1 must represent any data internally. Hence, computers use binary coding schemes to represent data internally. In binary coding, a group of bits represent every symbol that appears in the data. The group of bits used to represent a symbol is called a *byte*. To indicate the numbers of bits in a group, sometimes a byte is referred to as “n-bit byte”, where the group contains n bits. However, the term “byte” commonly means an 8-bit byte because most modern computers use 8 bits to represent a symbol.

### 1.8.1 BCD

In computing and electronic systems, **Binary-Coded Decimal (BCD)** is a way to express each of the decimal digits with a binary code. Its main virtue is that it allows easy conversion to decimal digits for printing or display and faster decimal calculations. Decimal numbers with their BCD equivalent are given in the table 2.4 :

**Table 2.4 : BCD Code**

Decimal	BCD
0	0000
1	0001
2	0010

3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Unlike binary encoded numbers, BCD encoded numbers can easily be displayed by mapping each of the nibbles(4-bits) to a different character. Conversion of decimal to BCD and BCD to decimal are shown below:

**Example 1.24 :** Convert the decimal numbers 47 and 180 to BCD.

**Solution :** 4 = 0100 and 7 = 111

$$47 = 0100111$$

Similarly, 180=0001 1000 0000

**Example 1.25 :** Convert each of the BCD code 1000111 to decimal.

**Solution :** First we have to divide the whole BCD code by a set of 4-bits from right to left. Then from left to right we can put the corresponding decimal numbers.

$$\begin{array}{cc} 0100 & 0111 \\ 4 & 7 \end{array}$$

Thus, 1000111(in BCD) = 47 (in Decimal)

**BCD addition :** BCD is a numeric code and can be used in arithmetic operations. Addition is the most important operation because the other three operations (subtraction, multiplication, and division) can be accomplished by the use of addition. Here is how to add two BCD numbers:

Step1 : 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 states 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 1.26 :** Add the following BCD numbers :

- a) 0001 + 0100
- b) 10000111 + 01010011

**Solution :** The decimal number addition are shown for comparison.

$$\begin{array}{r} \text{(a)} \quad 0001 \qquad 1 \\ +0100 \qquad +4 \\ \hline 0101 \qquad 5 \end{array}$$

∴ 0001 + 0100 = 0101 Which is valid BCD no. (Value < 9)

$$\begin{array}{r} \text{(b)} \quad 1000 \quad 0111 \qquad 87 \\ +0101 \quad 0011 \qquad +53 \\ \hline 1101 \quad 1010 \quad \text{Both groups are invalid (>9) +140} \\ +0110 \quad +0110 \quad \text{Add 6 (i.e., 0110) to both groups} \\ \hline 0001 \quad 0100 \quad 0000 \quad \text{Valid BCD number which is 140} \\ \qquad \qquad \qquad \text{in decimal.} \end{array}$$

### 1.8.2 ASCII

ASCII stands for American Standard Code for Information Interchange. It is a very well-known fact that computers can manage internally only 0s (zeros) and 1s (ones). By means of sequences of 0s and 1s the computer can express any numerical value as its binary translation, which is a very simple mathematical operation.

However, there is no such evident way to represent letters and other non-numeric characters with 0s and 1s. Therefore, in order to do that, computers use *ASCII tables*, which are tables or lists that contain all the letters in the roman alphabet plus some additional characters. In these tables each character is always represented by the same order number. For example, the ASCII code for the capital letter "A" is always represented by the order number 65, which is easily representable using 0s and 1s in binary: 65 expressed as a

binary number is 1000001. ASCII has 128 character codes (from 0 to 127) and symbols represented by a 7-bit binary code.

ASCII is the common code for microcomputer equipment. The first 32 characters in the ASCII-table are unprintable control codes and are used to control peripherals such as printers. Examples of the control characters are "NULL", "line feed", "start of text" and "escape". The other characters are graphic symbols that can be printed or displayed and that include the letters of the alphabet (lowercase and upper case), the ten decimal digits, punctuation signs, and other commonly used symbols.

---

### 1.8.3 EBCDIC

---

EBCDIC is the abbreviation for **Extended Binary-Coded Decimal Interchange Code**. EBCDIC is an IBM code for representing characters as numbers. It uses 8 bits per character. Thus 256 characters can be represented with 8 bits. The 9th position bit can be used for parity. The EBCDIC code is used in IBM mainframe models and other similar machines.

In EBCDIC, the first 4 bits are known as zone bits and remaining 4 bits represent digit values. Electronic circuits are available to transform characters from ASCII to EBCDIC and vice-versa.

---

### 1.8.4 Gray Code

---

The gray code is an unweighted code not suited for arithmetic operations, but useful for input output devices, analog to digital converters etc. The **Gray code**, named after *Frank Gray*, is a binary numeral system where two successive values differ in only one digit. It is sometimes referred to as reflected binary, because the first eight values compare with those of the last 8 values, but in reverse order.

The Gray code was originally designed to prevent spurious output from electromechanical switches. Today, Gray codes are widely used

to facilitate error correction in digital communications such as digital terrestrial television and some cable TV systems.

**Table 2.5 : Gray Code**

Decimal	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

**Conversion from Binary to Gray code :** The following rules explain how to convert from a binary number to a Gray code :

- The most significant bit (left most) in the gray code is the same as the corresponding most significant bit in the binary code.
- Going from left to right, add each pair of adjacent pair of binary code to get the next Gray code bit. Discard carries.

For example, the conversion of the binary number 1100 to gray code is as follows:

Binary code      1      1      0      0

MSB (Most significant bit)      LSB (Least significant bit)

MSB of Gray code will be same as MSB of Binary code. Here, it will be 1.

Now , addition of each pair of adjacent bits of Binary code:

$$1 + 1 = 10 = 0, \text{ discarding the carry } 1$$

$$1 + 0 = 1, \text{ no carry}$$

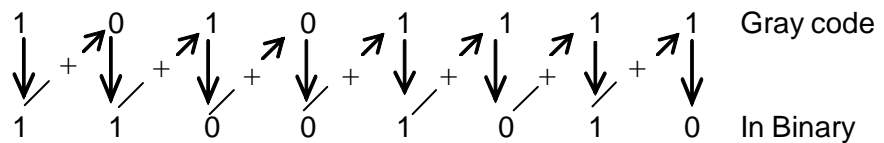
$$0 + 0 = 0, \text{ no carry}$$

$\therefore$  1100 in gray code is 1010

**Conversion from Gray to Binary code :** The following rules apply:

- The most significant bit (left most) in the binary code is the same as the corresponding most significant bit in the gray code.
- Add each binary code bit generated by Gray code bit in the next adjacent position. Discard carries.

For example, binary 10101111 in gray code will be like this :



### CHECK YOUR PROGRESS

Q.13. Convert the following binary no to gray code no.

(i) 11011

(ii) 10110

Q.14. Convert the following Gray codes to Binary codes.

(i) 11011

(ii) 100111

Q.15. (i) The American Standard Code for Information Interchange is a standard \_\_\_\_\_ bits code.

(ii) EBCDIC uses \_\_\_\_\_ bits per character.

(iii) Code is a representation of \_\_\_\_\_.

(iv) In both binary and gray code, the \_\_\_\_\_ is same.

Q.16. Add the following BCD numbers :

(i) 00100011 + 00010001

(ii) 1001 + 0100

.....  
 .....  
 .....

- We have learnt 4 different number systems used in digital systems, their use, conversion of one number system to another and the arithmetic operation of binary system. We also have learnt about the representation of data.
- In our day-to-day life, we use the decimal number system. In this system, base is equal to 10.
- The 1's complement of a binary number is formed by changing 1's into 0 and 0's into 1's
- The 2's complement of a binary number is obtained by adding 1 in the 1's complement form of the binary number.
- BCD, ASCII, EBCDIC, and Gray Code are the commonly used computer codes.



---

37

**Ans. to Q. No. 13 :** (i) 10110 (ii) 11101

**Ans. to Q. No. 14 :** (i) 10010 (ii) 111010

**Ans. to Q. No. 15 :** (i) 7 (ii) 8 (iii) Discrete elements  
(iv) MSB-bit

**Ans. to Q. No. 16 :** (i) Solution :

0010 0011 35

+0001 0001 +17

0011 0100 +52 Valid BCD no. (since both <9)

(ii) Solution :

1001 9

+0100 +4

1101 Invalid BCD number (>9) +13

+0110 Add 6 (i.e., 0110)

0001 0011 A valid BCD number which  
is 13 in decimal



## 1.11 FURTHER READINGS

- *Computer Fundamentals Architecture and Organization*, B. Ram, New Age International (P) Ltd.
- *Introduction to Computer Science*, ITL Education Solution Limited, Pearson Education
- *Digital Logic and Computer Design*, M. M. Mano, PHI
- *Computer Fundamentals*, P K Sinha, BPB Publication.
- *Digital Techniques*, Dr. Pranhari Talukdar, N.L. Publications.
- *Computer Fundamentals and C programming*, Rath, Jagdev and Swain, SCITECH.



## 1.12 MODEL QUESTIONS

Q1. What is binary number system? How do you perform binary subtraction using 1's and 2's complement method?

- Q.2. What is meant by 'Base' or 'radix' of a number system?
- Q.3. What is a floating point number? What are the advantages of it?
- Q.4. What do you understand by normalized floating point number?
- Q.5. Convert the decimal numbers 15, 275 to BCD.
- Q.6. Add the two BCD numbers 10000111 and 01010011.
- Q.7. Find 8's and 16's complement of the following:
- (i) 67 (ii) 5672
- Q.8. Find the 7's and 15's complement of the following:
- (i) 643 (ii) 15AB
- Q.9. Write BCD for the following decimal number:
- (i) 679 (ii) 45.96
- Q.10. Perform the following with binary arithmetic:
- (i)  $110111 + 10011$  (ii)  $1110.001 + 1010.101$   
(iii)  $11111 - 1101$  (iv)  $1010.001 - 110.1$   
(v)  $1101 \times 101$  (vi)  $1011.11 \times 1.1$   
(vii)  $10011 / 111$  (viii)  $101101.1101 / 11.1$
- Q.11. Convert the following to its binary equivalent:
- (i)  $(67)_8$  (ii)  $(A8D)_{16}$  (iii)  $(81B6.F)_{16}$   
(iv)  $(64.3)_{10}$  (v)  $(765.45)_8$  (vi)  $(1725.23)_8$
- Q.12. Convert the following decimal number to octal number:
- (i) 104 (ii) 457 (iii) 7562
- Q.13. Convert the following binary numbers to Octal and then Hexadecimal equivalent:
- (i) 1101101101 (ii) 1010110.001 (iii) 1101.1011
- Q.14. Convert the following decimal numbers to octal and then to its binary equivalent:
- (i) 100 (ii) 0.57 (iii) 1011.3
- Q.15. Perform the following by 2's complement method:
- (i)  $10101 - 11011$  (ii)  $100011 - 1111$

---

## UNIT 2 : BOOLEAN ALGEBRA

---

### UNIT STRUCTURE

- 2.1 Learning Objective
- 2.2 Introduction
- 2.3 Boolean Operators
- 2.4 Basic Theorems and Postulates of Boolean Algebra
- 2.5 Representation of Boolean Expression
- 2.6 Simplification of Boolean Functions
  - 2.6.1 Algebraic Manipulation Method
  - 2.6.2 Map method
  - 2.6.3 Don't Care Conditions
- 2.7 Let Us Sum Up
- 2.8 Further Readings
- 2.9 Answers to Check Your Progress
- 2.10 Model Questions

---

### 2.1 LEARNING OBJECTIVES

---

After going through this unit, you will be able to :

- Define Boolean Algebra
- Explain basic concepts of Boolean Algebra
- Define the basic theorems and postulates of Boolean Algebra
- Define Boolean function
- Define canonical and standard forms
- Simplify a Boolean function by algebraic manipulations and K-maps.

---

### 2.2 INTRODUCTION

---

In this unit, you will be able to learn about the fundamentals of Boolean Algebra and logic operations. Three basic logic operations are AND, OR and NOT operations, where AND is logical multiplication, OR is



logical addition and NOT is logical complementation. The logical AND operation between two variables X and Y is written as " $X \bullet Y$ ". The result of  $X \bullet Y$  is logical 0 for all cases except when both X and Y are logical 1. The logical OR operation is written as " $X + Y$ ", and produces a logical 1, when X or Y or both are logical 1. The logical NOT operation changes logical 1 to logical 0 and vice versa.

You will also learn the basic theorems and postulates of Boolean algebra, the principles of Duality, De Morgains theorem. Finally, you will learn about **sum of product** and **product of sums** simplification of Boolean functions using K-map.

---

## 2.3 BOOLEAN OPERATORS

---

Boolean algebra may be defined with the help of three sets of components viz,

- i) a set of elements,
- ii) a set of operators and
- iii) a number of unproved axioms or postulates. A set of elements is any collection of objects having a common property.

A set of operators here is the binary operators, which are the rules that assigns to each pair of elements of the set a unique element from the set. The axioms or postulates form the basic assumptions from which it is possible to deduce the rules, theorems and properties of Boolean algebra. A variable in Boolean algebra can take only two values, 1 (TRUE) or 0 (FALSE). i.e. TRUE is represented by 1 and FALSE is represented by 0. Boolean algebra is used for designing and analysing digital circuits. There are three basic operations in Boolean algebra and these operations are done with the help of three operators, viz. AND, OR and NOT. (These are also called basic logic operation).

**AND operation :** Boolean AND operator for two variables A and B can be represented as:

A and B or  $A.B$  or  $AB$

It results 1 or TRUE if both the operands A and B are 1 (TRUE), otherwise the result of 0 (FALSE).

**OR Operation :** The OR operator for the same variables can be represented as :

$$A \text{ OR } B \text{ or } A+B$$

The result of this operation is 0 (FALSE) if both the variables are 0 (FALSE); otherwise the result is 1 (TRUE)

**NOT Operation :** The NOT operation for a variable A can be represented as:

$$\text{NOT } \bar{A} \text{ or } A \text{ or } A'$$

It returns the opposite value of the variable i.e. returns 0 (FALSE) if A is 1 (TRUE) and vice versa.

The results of these Boolean operations can be represented in a tabular form, which is referred to as the "**truth table**".

**Truth Table 2.1 for AND OR and NOT operation**

A	B	(A AND B) A.B	(A OR B) A+B	NOT A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

**Note :** Boolean addition is same as the OR operation and Boolean multiplication is same as AND operation.

**Truth Table :** A truth table is a table which gives the output for all combination of input values. A truth table can be drawn for a particular function/operation. In the truth table 2.1, three tables are merged into a single one.

In addition to these three basic Boolean operations, three more operators have been defined for Boolean algebra. They are: XOR (Exclusive OR), NOR (Not+OR) NAND (Not+AND).

---

## 2.4 BASIC THEOREMS AND POSTULATES OF BOOLEAN ALGEBRA

---

The theorems and postulates are the most basic relationships in Boolean algebra. Six theorems and four postulates of Boolean algebra are

listed in table 2.2. The postulates are basic axioms of the algebraic structure and need no proof, but the theorems must be proven with the help of the postulates. Both the postulates and theorems are listed in pairs: one is the dual of the other.

**Table 2.2****Postulates and Theorems of Boolean Algebra**

Postulate 2	(a) $x + 0 = x$	(b) $x \cdot 1 = x$
Postulate 5	(a) $x + x' = 1$	(b) $x \cdot x' = 0$
Theorem 1	(a) $x + x = x$	(b) $x \cdot x = x$
Theorem 2	(a) $x + 1 = 1$	(b) $x \cdot 0 = 0$
Theorem 3, Involution $(x')' = x$		
Postulate 3, Commutative	(a) $x + y = y + x$	(b) $xy = yx$
Theorem 4, Associative	(a) $x + (y + z) = (x + y) + z$	(b) $x(yz) = (xy)z$
Postulates 4, Distributive	(a) $x(y + z) = xy + xz$	(b) $x+yz = (x+y)(x+z)$
Theorem 5, De Morgan	(a) $(x + y)' = x' y'$	(b) $(xy)' = x' + y'$
Theorem 6, Absorption	(a) $x + xy = x$	(b) $x(x + y) = x$

**Duality Principle :** This is an important property of Boolean algebra.

It states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operator and identity elements are interchanged. In a two-valued Boolean algebra (which is defined on a set of two elements,  $B = \{0, 1\}$ , with rules for the two binary operators  $+$  and  $\bullet$ ), the identity elements and the elements of the set  $B$  are same: 1 and 0. If we need the dual of an algebraic expression we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

The proofs of the theorems with one variable are given below:

Theorem 1 (a)  $x + x = x$

$$\begin{aligned}
 x + x &= (x + x) \cdot 1 \text{ by postulate} & : & \quad 2 \text{ (b)} \\
 &= (x + x) \bullet (x + x') & : & \quad 5 \text{ (a)} \\
 &= x + x x' & : & \quad 4 \text{ (b)} \\
 &= x + 0 & : & \quad 2 \text{ (a)} \\
 &= x
 \end{aligned}$$

Theorem 1 (b) :  $x \cdot x = x$

$$\begin{aligned}
 x \cdot x &= x \cdot x + 0 && \text{by postulate} && : && 2 \text{ (a)} \\
 &= xx + x x' && && : && 5 \text{ (b)} \\
 &= x (x + x') && && : && 4 \text{ (a)} \\
 &= x \cdot 1 && && : && 5 \text{ (a)} \\
 &= x && && : && 2 \text{ (a)}
 \end{aligned}$$

If we observe carefully we see that theorem 1 (b) is the dual of theorem 1 (a) and each step of the proof in part (b) is the dual of part (a). Thus any dual theorem can be similarly derived from the proof of its corresponding pair.

Theorem 2 (a):  $x + z = 1$

$$\begin{aligned}
 x + 1 &= 1 \cdot (x + 1) && \text{by postulate} && : && 2 \text{ (b)} \\
 &= (x + x') (x + 1) && && : && 5 \text{ (a)} \\
 &= x + x' \cdot 1 && && : && 4 \text{ (b)} \\
 &= x + x' && && : && 2 \text{ (b)} \\
 &= 1 && && : && 5 \text{ (a)}
 \end{aligned}$$

Theorem 2 (b):  $x \cdot 0 = 0$  by duality.

Theorem 3:  $(x')' = x$

We have  $x \cdot x' = 0$  from postulate 5 (b), which defines complement of  $x$ .

The complement of  $x'$  is  $x$  and is also  $(x')'$ . Since the complement is unique, therefore we have  $(x')' = x$ .

We can prove the theorems which involve two or three variables, algebraically, from the postulates and theorems which have already been proven. Let us consider the absorption theorem.

Theorem 6 (a) :  $x + xy = x$

$$\begin{aligned}
 x + xy &= x \cdot 1 + xy && \text{by postulate 2 (b)} \\
 &= x (1 + y) && \text{by postulate 4 (a)} \\
 &= x (y + 1) && \text{by postulate 3 (a)} \\
 &= x \cdot 1 && \text{by theorem 2 (a)} \\
 &= x && \text{by postulate 2 (b)}
 \end{aligned}$$

Theorem 6 (b):  $x (x + y) = x$  by duality.

The theorems of Boolean algebra can also be proved/verified easily with the help of truth table. The following truth table verifies the theorem 6 (b).

Table 2.3

1	2	3	4
x	y	$x + y$	$x(x + y)$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Here we see that, column 4 and column 1 are same, i.e.  $x(x + y) = x$ . Since the algebraic proof of the De Morgan's theorem and the associative law are very long, we can show their validity with truth tables easily. Let us consider the De Morgan's theorem:  $(x + y)' = x' y'$

Now, the truth table for this is shown below.

Table 2.4

1	2	3	4	5	6	7
x	y	$x + y$	$(x + y)'$	$x'$	$y'$	$x' y'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Column 4 is equal to column 7, so,  $(x + y)' = x' y'$

**Complement of a Boolean Function :** The complement of a function  $F$  is  $F'$ . It can be obtained from an interchange of 0's for 1's and 1's for 0's in the value of  $F$ . Algebraically, the complement of a function may be derived through De Morgan's theorem. The De Morgan's theorem can be extended to any number of variables.

The theorem can be generalized as :

$$(A + B + C + \dots + G)' = A' B' C' \dots G'$$

$$(ABC \dots G)' = A' + B' + C' + \dots + G'$$

This generalized form of De Morgan Theorem states that the complement of a function is obtained by interchanging AND and OR operators and complementing each literal. (A literal is a primed or unprimed variable). Let  $F_1 = A'BC + A'B'C'$ . Now  $F_1'$  i.e. complement of the function can be obtained by applying the De Morgan's theorem as follows:

$$\begin{aligned}
 F_1' &= (A'BC + A'B'C')' \\
 &= (A'BC)' (A'B'C')' \\
 &= (A+B'+C') (A+B+C)
 \end{aligned}$$

An easier procedure for deriving the complement of a function is to take the dual of the function and complement each literal. Thus, the dual of  $F_1 = (A'+B+C) (A'+B'+C')$  and then complementing each literal,  $F_1' = (A+B'+C') (A+B+C)$ .



### CHECK YOUR PROGRESS

Q.1. How are logical AND and OR operations of two variables denoted?

.....

.....

.....

Q.2. What is a truth table? Create a truth table for AND and OR operation?

.....

.....

.....

Q.3. Define the principle of Duality.

.....

.....

.....

Q.4. State and prove the De Morgan's theorem for two variables. (Using truth table)

.....

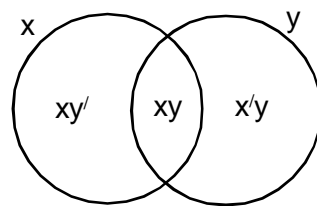
.....

.....

## 2.5 REPRESENTATION OF BOOLEAN EXPRESSION

A Boolean function can be expressed in three ways, viz, i) in canonical form, ii) in standard form and in iii) non standard form. To understand these forms we must have the idea on Minterms and Maxterms.

**Minterms and Maxterms :** Let us consider two binary variables  $x$  and  $y$ , which may appear in their normal form or its complement form. Now combining these two variables with an AND operation, we get four possible combinations:  $x'y'$ ,  $x'y$ ,  $xy'$ ,  $xy$ . Each of these AND terms represents one of the distinct areas in the Venn diagram of the following fig 2.1, and is called a **minterm** or a **standard product**.



**Fig. 2.1 : Venn Diagram for two variables**

In this way,  $n$  variables can be combined to form  $2^n$  minterms. Each minterm is obtained from an AND term of the  $n$  variables, with each variable being primed if the corresponding bit of the binary number is a 0 and unprimed if a 1. Each minterm is designated as  $m_j$ , where  $j$  denotes the decimal equivalent of the binary number of the minterm designated.

**Table 2.5 : Minterms and Maxterms for three binary variables**

Minterms					Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	$m_0$	$x + y + z$	$M_0$
0	0	1	$x'y'z$	$m_1$	$x + y + z'$	$M_1$
0	1	0	$x'y z'$	$m_2$	$x + y' + z$	$M_2$
0	1	1	$x'y z$	$m_3$	$x + y' + z'$	$M_3$
1	0	0	$x y' z'$	$m_4$	$x' + y + z$	$M_4$
1	0	1	$x y' z$	$m_5$	$x' + y + z'$	$M_5$
1	1	0	$x y z'$	$m_6$	$x' + y' + z$	$M_6$
1	1	1	$xyz$	$m_7$	$x' + y' + z'$	$M_7$

Similarly, with these  $n$  variables an OR term can be formed with each variables being primed or unprimed. Thus, we may have  $2^n$  possible combinations and these are called **Maxterms** or **standard sums**. In the table 2.5, the eight maxterms for three variables  $x$ ,  $y$  and  $z$ , together with their symbolic designation, are listed. Any  $2^n$  maxterms for  $n$  variables may be determined similarly. Each maxterm is obtained from an OR term of the  $n$  variables, with each variable being unprimed if the corresponding bit is a 0 and primed if a 1. Each maxterm is the complement of its corresponding minterm and vice versa.

If the truth table for a Boolean function is given we can express the Boolean function algebraically by forming a minterm for each combination of the variables which produces a 1 in the function, and then taking the OR of all those terms.

In the table 2.6 two functions  $f_1$  and  $f_2$  are listed.

**Table 2.6 : Functions of three variables**

a	b	c	function $f_1$	function $f_2$
0	0	0	1	0
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1
1	1	0	1	0
1	1	1	0	1

From the table, we see that, function  $f_1$  is 1 for four combinations of  $a$ ,  $b$ ,  $c$  i.e.  $a'b'c'$ ,  $a'bc$ ,  $ab'c$  and  $abc'$ . Thus, for each one of these minterms the function  $f_1=1$ , and we can express the function  $f_1$  as:

$$\begin{aligned}
 f_1 &= a'b'c' + a'bc + ab'c + abc' \\
 &= m_0 + m_3 + m_5 + m_6 \\
 &= \sum (0, 3, 5, 6)
 \end{aligned}$$

(The symbol ' $\sum$ ', is used to mean sum of minterms. Similarly, we can easily verify that –



$$\begin{aligned}
 f_2 &= a' b c' + a b' c' + a b' c + a b c \\
 &= m_2 + m_4 + m_5 + m_7 \\
 &= \sum (2, 4, 5, 7)
 \end{aligned}$$

From here we observe an important property of Boolean algebra. Any Boolean function can be expressed as a sum of minterms (sum means ORing to terms)

Now consider again the table 2.3 to read the complement of  $f_1$ . We can obtain the complement of  $f_1$  by forming a minterm for each combination of  $a, b, c$ , that produce a 0 in the function and then ORing to those terms.

$$\text{Thus } f_1' = a' b' c + a' b c' + a b' c' + a b c$$

Now taking the complement of  $f_1'$ , we obtain the function  $f_1$ :

$$\begin{aligned}
 (f_1')' &= f_1 = (a' b' c + a' b c' + a b' c' + a b c)' \\
 &= (a + b + c') (a + b' + c) (a' + b + c) (a' + b' + c') \\
 &= M_1 M_2 M_4 M_7 \\
 &= \pi (1, 2, 4, 7) \text{ (}\pi \text{ is used to mean product of maxterms)}
 \end{aligned}$$

This demonstrates a second important property of Boolean algebra: Any Boolean function can be expressed as product of maxterms (product means ANDing of terms).

Now, we are in a position to define the canonical form of a Boolean function. Boolean functions expressed as a sum of minterms or product of maxterms (i.e. each term of the function must have all the literals) are said to be in canonical form.

**Sum of Minterms :** It is sometimes convenient to express the Boolean function in its sum-of-minterms form. It can be done by first expanding the expression into a sum of AND terms. Then each term is inspected to see if it contains all the variables/literals. If one or more variable is missing in a term, it is ANDed with an expression  $(a + a')$ , where  $a$  is one of the missing variables. Let us take an example to clarify this procedure.

**Note :**  $a + a' = 1$

**Example :** Express the Boolean function  $xy + xz' + y'$  in a sum of minterms. Here we see that the function has 3 variables and there are 3 AND terms. Two terms missing one variable and one term missing two variables.

**Solution :** In term  $xy$ ,  $z$  is missing: therefore:  $xy (z + z')$

$$= x y z + x y z'$$

Similarly in  $x z'$ ,  $y$  is missing  $\therefore x z' (y + y') = x y z' + x y' z'$

In  $y'$  two variables  $x$  &  $z$  are missing, therefore:

$$y' (z + z') = y' z + y' z', \text{ still missing } x$$

$$\therefore y' z (x + x') + y' z' (x + x')$$

$$= x y' z + x' y' z + x y' z' + x' y' z'$$

Now, combining all the terms

$$F = xy + xz' + y'$$

$$= x y z + x y z' + x y' z + x y' z' + x' y' z + x' y' z' + x' y' z'$$

Eliminating duplicate terms: ( $x + x = x$ )

$$F = x y z + x y z' + x y' z + x y' z' + x' y' z + x' y' z'$$

Rearranging the minterms in ascending order

$$F = x' y' z' + x' y' z + x y' z' + x y' z + x y z' + x y z$$

$$= \sum (0, 2, 4, 5, 6, 7)$$

**Products of Maxterms :** A Boolean function can also be expressed as a product of maxterms. For this, the function first be brought into a form of OR terms. It can be done by using the distributive law  $x + yz = (x + y) (x + z)$ . Then for any missing variable  $x$  in each OR term, the term is ORed with  $xx'$ . The process will be clear from the following example.

**Example :** Express the Boolean function  $f = x y' + x z'$  in a product of maxterm form.

First we convert the function into OR terms using the distributive law.

$$f = xy' + x z' \text{ (Let } x y' = x, y = x \text{ \& , } z = z')$$

$$= (x y' + x) (x y' + z')$$

$$= (x + x) (x + y') (x + z') (y' + z')$$

$$= x (x + y') (x + z') (y' + z')$$

Now ORing each OR term with  $x x'$  (where  $x$  is the missing variable)

$$x = x + y y' = (x + y) (x + y')$$

$$x + y = x + y + z z' = (x + y + z) (x + y + z')$$

$$x + y' = x + y' + z z' = (x + y' + z) (x + y' + z')$$

$$x + z' = x + z' + y y' = (x + y + z') (x + y' + z')$$

$$y' + z' = y' + z' + x x' = (x + y' + z') (x' + y' + z')$$

Combining all the terms and removing those that appear more than once we finally obtain:

$$\begin{aligned} f &= (x + y + z) (x + y + z') (x + y' + z) (x + y' + z') (x' + y' + z') \\ &= M_0 M_1 M_2 M_3 M_7 \\ &= \pi (0, 1, 2, 3, 7) \text{ (The symbol } \pi \text{ denotes the ANDing of maxterms)} \end{aligned}$$

**Conversion between canonical forms :** Boolean functions represented in one canonical form can easily be converted to the other canonical form. It will be clear from the following example.

**Example :** Convert the function  $F(x, y, z) = \sum (2, 4, 5, 7)$  into product of maxterm form.

**Solution :** We have  $F(x, y, z) = \sum (2, 4, 5, 7) = m_2 + m_4 + m_5 + m_7$

Taking complement  $F'(x, y, z) = \sum (0, 1, 3, 6) = m_0 + m_1 + m_3 + m_6$

Now, if we take the complement of  $F'$  by De Morgan's theorem,

$$\begin{aligned} \text{We obtain } F &= (m_0 + m_1 + m_3 + m_6)' \\ &= m_0' m_1' m_3' m_6' \\ &= M_0 M_1 M_3 M_6 \\ &= \pi (0, 1, 3, 6) \end{aligned}$$

**Note :** From table 2.5, it is clear that relation  $m_j' = M_j$  is true i.e. a maxterm is a complement of the corresponding minterm and vice versa.

The simplest way to convert from one canonical form to other is: Interchange the symbols  $\sum$  and  $\pi$  and list those numbers missing from the original form.

**Standard form :** In this form, the terms that form the Boolean function may contain one, two or any number of literals. Two types of standard forms are: **the sum of product** and **product of sums**.

The sums of products is a Boolean expression containing, AND terms, called product terms, of one or more literals each.

$$\text{eg: } F_1 = a + b c' + a b' c$$

A product of sums is a Boolean expression containing OR terms, called sum terms. Here also each term may have any number of literals.

$$\text{eg: } F_2 = y (x + y') (x' + z) (x + y' + z')$$

A Boolean function may also be expressed in a non standard form.

$$\text{eg: } F_3 = (x' y + x' z) (x' y + x' y')$$

This function is neither in sum of product nor in product of sums. Using distributive law it can be converted into a standard form.

---

## 2.6 SIMPLIFICATION OF BOOLEAN FUNCTIONS

---

We see in previous section that representation of a Boolean function is not unique. The same function can be represented by different numbers of literals and terms. These Boolean functions have to be finally implemented by digital logic gates. The complexity of the digital logic gates is directly related to the complexity of the algebraic expression from which the function is implemented. So, it is an important task to simplify the Boolean functions so that they can be implemented easily in a cost effective manner. In this section, we will discuss two methods of simplifying a Boolean function, viz, ***algebraic manipulation method*** and the ***map method***.

---

### 2.6.1 Algebraic Manipulation

---

On implementing a Boolean function with logic gates, each literal/variable in the function designates an input to a gate, and each term is implemented with a gate. The minimization of the number of literals and the number of terms results in a circuit with less equipment, but it is not always possible to minimize both simultaneously. We will consider now literal minimization. Literal minimization can be done by algebraic manipulation. There are no specific rules to follow which will ensure the final answer. The only method available is a cut-and-try procedure using the postulates, the basic theorems and any other manipulation method. To illustrate this procedure let us consider the following examples: Simplify the following Boolean functions to a minimum number of literals.

$$\begin{aligned}
 1) \quad & x y + x y' \\
 &= x (y + y') \\
 &= x \cdot 1 \\
 &= x
 \end{aligned}$$

$$\begin{aligned}
 2) \quad & x y z + x' y + x y z' \\
 &= x y (z + z') + x' y \\
 &= x y + x' y \\
 &= y (x + x') \\
 &= y.
 \end{aligned}$$

$$\begin{aligned} 3) & \quad xy + x'z + yz \\ &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z \end{aligned}$$

In these examples, literals are minimized in 1 and 2, and in 3 a term is minimized.

The algebraic manipulation procedure of minimizing a Boolean function is awkward because it lacks specific rules to predict each succeeding step in the process.

---

### 2.6.2 Map Method

---

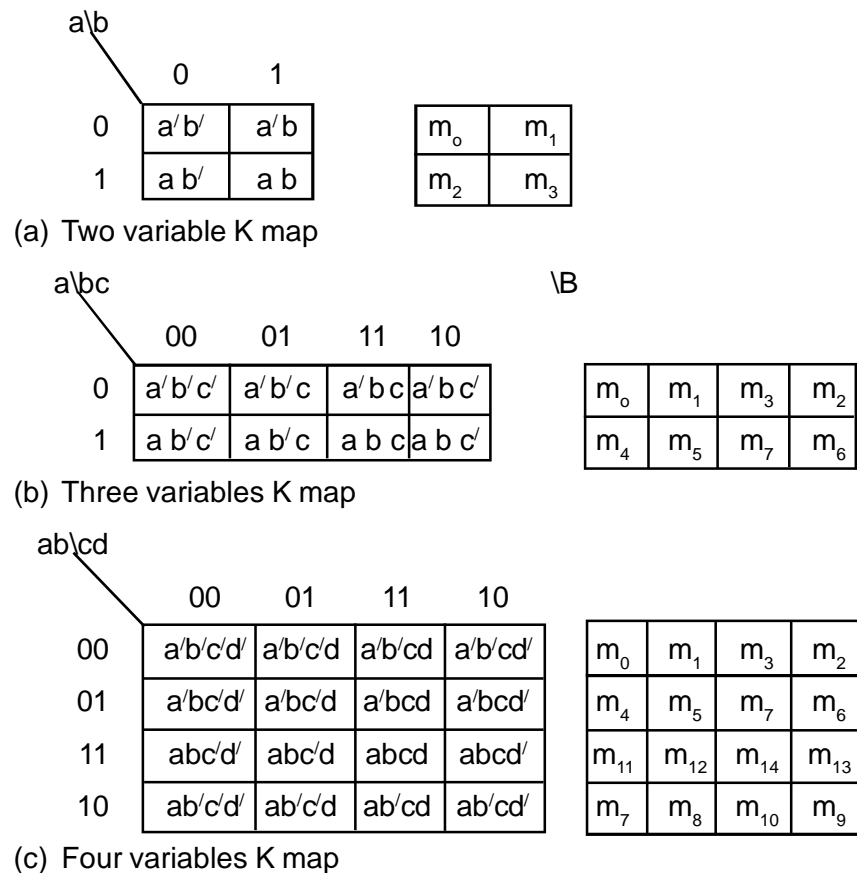
The map method provides a simple straight forward method for minimizing Boolean functions. It was first proposed by Veitch and slightly modified by Karnaugh and known as the **Veitch diagram** or the “**Karnaugh Map**”. This method may be regarded as the pictorial form of a truth table.

The map is a diagram made up of squares. Each square represents one minterm. As we have already seen that any Boolean function can be represented as a sum of minterms so a Boolean function is recognized graphically in the map from the area enclosed by those squares whose minterms are included in the function.

The number of squares in a K-map depends upon the number of variables of the Boolean expression. For  $n$  variables, we have  $2^n$  minterms (possible combinations of  $n$  variables) and hence  $2^n$  squares in the map. However, the map method is convenient only for small number of variables, i.e. upto six variables. In this section we will consider up to four variables map.

#### Two, Three and Four variables maps :

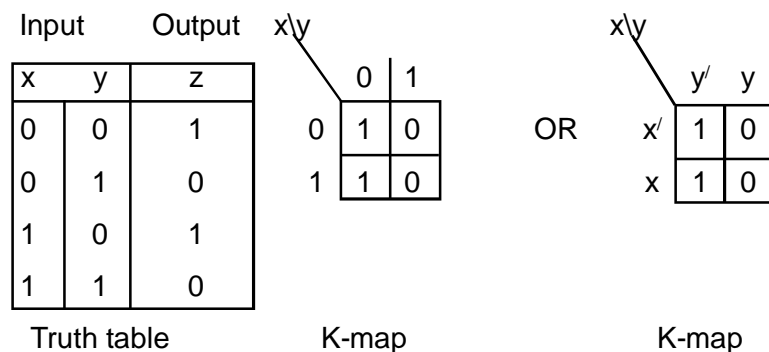
K-maps for two, three and four variables map are shown in Fig. a, b & c.

**Fig. 2.2 : K-map**

There are four minterms for two variables; hence the map consists of four squares, one for each minterm. Similarly, for three and four variables there are eight and sixteen minterms respectively and hence the maps consist of eight and sixteen squares.

### Representation of Truth Table on K-maps :

We have already known that truth table defines the relationship between output and input. Truth table – involves two variables and its K-map representation is shown in the figure.



Here, we see that we put 1's in the squares whose minterms are 0 and 2. This is because, from the truth table we see that the output is 1, only for the input combinations  $x=0, y=0$  and  $x=1$  and  $y=0$ . For others the output is 0, and we put 0's in the map accordingly. Similarly, we can represent a truth table having  $n$  variables with a K-map of  $2^n$  squares.

**Plotting a Boolean expression on the Karnaugh Map :** As we already have seen that each square of the K map represents one minterm, so the expression to be plotted must be in sum of product form. The following example clarifies the point.

**Example :** Plot the Boolean function  $f=yz + xy'z' + xyz'$  on a K-map.

The term  $y z$  missing one variable, so

$$y z = y z (x + x') = x y z + x' y z$$

$$\text{Now, } F = x' y z + x y z + x y' z' + x y z'$$

$$= m_3 + m_7 + m_4 + m_6$$

$$= m_3 + m_4 + m_6 + m_7$$

Since the expression has three variables, we need a K-map of  $2^3 = 8$  squares.

$x \backslash yz$		00	01	11	10
0		0	0	1	0
1		1	0	1	1

We plot a 1 in the squares of minterms 3, 4, 6 and 7, and a zero in the other squares.

**Simplification of Boolean Function by K-map :** We have already seen how a Boolean function can be plotted on the K-map. Now, we will see the process of simplification of Boolean function using K-map. If you observe carefully the K-maps of two, three and four variables in figure you will see that the neighbouring squares are adjacent to each other. Two squares are said to be adjacent if they differ in only one variable. Consider the following three variables k-map.

a\bc	00	01	11	10		a\bc	00	01	11	10
0	a'b'c'	a'b'c	a'bc	a'bc'	OR	0	m <sub>0</sub>	m <sub>1</sub>	m <sub>3</sub>	m <sub>2</sub>
1	ab'c'	ab'c	abc	abc'		1	m <sub>4</sub>	m <sub>5</sub>	m <sub>7</sub>	m <sub>6</sub>

In this map,  $m_0$  ( $a'b'c'$ ) is adjacent to  $m_1$  ( $a'b'c$ ) because only one variable i.e. "c" appears in  $m_0$  in its complement form and in  $m_1$ , appears in normal form, other variables are same. Similarly,  $m_0$  is adjacent with  $m_4$  and  $m_2$ ;  $m_1$  is adjacent with  $m_3$ ,  $m_0$  and  $m_5$ ;  $m_4$  is adjacent with  $m_0$ ,  $m_5$ , and  $m_6$ ; and so on. When we are going to simplify a Boolean function using K-map, we will consider this adjacency of squares and try to combine as much as possible adjacent squares. We may combine the number of adjacent squares as two, four, eight and sixteen in a four variables map.

By combining two adjacent squares of a four variable map we get a term of three variables/literals.

- combining four adjacent squares we get a term of two literals.
- combining eight adjacent squares we get a term of one literal and combining sixteen adjacent squares we get a 1 i.e. the function is equal to 1. (i.e. the sum of all minterms is equal to 1).

**Example :** Simplify the Boolean function:

$$F = x' y' z' + x' y' z + x y z' + x y z$$

x\yz	00	01	11	10
0	1	1		
1			1	1

**Fig. 2.3 :** K-map for the function  $F = x' y' z' + x' y' z + x y z' + x y z$

**Note :** Here, adjacent squares are marked with rectangles

After plotting the function, adjacent squares are combined with each other.

Combining the two squares on top we get the term  $x' y'$  (i.e. here variable  $z$  is changing, so omit it).

Similarly, combining the two squares on bottom, we get  $xy$ .

So, the given function is simplified to  $F = x' y' + xy$ .



**Example :**  $f(a, b, c) = \sum (0, 2, 4, 5, 6)$

a\bc				
	00	01	11	10
0	1			1
1	1	1		1

Observing the K-map we see that minterm  $m_0$  is adjacent with  $m_4$  and collectively they are adjacent to  $m_2$  and  $m_6$ . (This type of adjacency of squares may be determined easily as follows.

Consider the table as an open book, fold it in middle, the squares overlapped with each others are the adjacent squares).

Now, combining  $m_0, m_2, m_4$  and  $m_6$  we get  $c'$  (i.e. other variables/ literals are changing, only  $c'$  is common to all these squares).

Again, combining  $m_4$  and  $m_5$  we get  $a b'$ .

We may combine one square more than once, if necessary.

So, the simplified function is  $f(a, b, c) = c' + a b'$ .

**Example :** Simplify the Boolean function:

$$F = A' B' C' + B' C D' + A' B C D' + A B' C'$$

**Solution :** Here, we need a four variable map. The first, second and fourth terms are missing one variable each. So ANDing with  $(x + x')$  (Where x is the missing variable) we get the function:

$$F = A' B' C' D + A' B' C' D' + A' B' C D' + A B' C D' + A' B C D' + A B C D' + A B' C' D' + A B' C' D$$

Now, plotting the function to the K-map, we get

AB\CD				
	00	01	11	10
00	$m_0$ 1	$m_1$ 1	$m_3$	$m_2$ 1
01	$m_4$	$m_5$	$m_7$	$m_6$ 1
11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{11}$
10	$m_8$ 1	$m_9$ 1	$m_{11}$	$m_{10}$ 1

Combining mean terms i.e. squares in the four corners, i.e.  $m_0$ ,  $m_2$ ,  $m_8$  and  $m_{10}$ , we get  $B'D'$

Combining  $m_0, m_1$ , with  $m_8, m_9$  we get  $B'C'$  Now, only one square,  $m_6$  remains without combining. We can combine it with  $m_2$  ( $\therefore$  they are adjacent) and it gives the term  $A'BC'$ . Note that if we do not combine  $m_6$ , we have to write the term as  $A'BCD'$ , i.e. with four literals.

So, the simplified form of the function is  $F = B'D' + B'C' + A'CD'$

**Product of Sum Simplification :** In all the previous examples, the minimized Boolean functions derived from the K-maps were expressed in the sum of products (SOP) form. The product of Sums (POS) form can be obtained with a minor modification.

The procedure for obtaining a minimized function in product of sums follows from the basic properties of Boolean functions. The minterms included in the function are represented by placing 1's in the corresponding squares on the map. The minterms not included in the function denote the complement of the function. If we mark these empty squares by 0's and combine them into valid adjacent squares, we obtain a simplified expression of the complement of the function, i.e. of  $F'$ . If we take the complement of  $F'$ , we get back the function  $F$ . The function so obtained is automatically in the product of sums form, because of the De Morgan's theorem. The following example clarifies the procedure.

**Example :** Simplify the Boolean function

$F(A,B,C,D) = \sum (1, 5, 8, 9, 10)$  in (a) sum of product and (b) product of sums.

**Solution :** Sum of products simplification of left as an exercise, and we will show only POS simplification.

(b) AB\CD

	00	01	11	10
00	<u>0</u>	1	<u>0</u>	<u>0</u>
01	<u>0</u>	1	0	<u>0</u>
11	<u>0</u>	0	<u>0</u>	<u>0</u>
10	1	1	<u>0</u>	1

Fig. 2.4 : Map for example

The squares marked with 0's represents the minterms not included in F i.e., they denote the complement of F. Combining the 0's as shown in the map, we obtain the simplified complemented function:  $F' = AB + CD + A'D'$

Applying De Morgan's theorem, we obtain:

$$F' = AB + CD + A'D'$$

$$(F')' = F = (AB + CD + A'D')'$$

$$= (AB)' (CD)' (A'D')'$$

$$= (A' + B') (C' + D') (A + D)$$

which is the simplified function in product of sums.

Answer of Example (a)  $F = A'B'C + AB'C' + AB'D'$

### 2.6.3 Don't Care Condition

The function is said to be completely specified when for every possible combination of input variables, we know an output value. Accordingly, these output values (i.e. 1's and 0's) are plotted in the map. The combinations are usually obtained from a truth table that lists the conditions under which the function is a 1. The function is assumed equal to 0 under all other conditions. But this assumption is not always true. There are some applications where certain combinations of input variables never occur. For example, we use 4 bit code to represent the decimal numbers, (0 to 9). Thus, we use the codes from 0000 to 1001, but other 6 codes from 1010 to 1111 are never used. These combinations are called Don't care conditions. These Don't care conditions can be used on a map to provide further simplification of the function.

It is to be remembered that a Don't care condition cannot be marked with a 1 on the map because it would mean that the function always produces a 1 for such input combinations. Similarly, for the same reason, we cannot mark these by a 0. To distinguish from 1's and 0's don't care conditions are marked by an X. The X's may be assumed to be either 0 or 1, in choosing adjacent squares to simplify the function in the map depending on which give the simplest expression. An X, which does not contribute to cover a larger area i.e. more minterms/squares, need not be used.

**Example :** Simplify the Boolean function

$$F(A,B,C,D) = \sum (1,3,7,11,15) \text{ with the don't care conditions:}$$

$$d(A,B,C,D) = \sum (0, 2, 5) \text{ in (a) SOP and (b) POS.}$$

**Solution :**

Step 1 : Construct a four variables K-map

Step 2 : Squares that represent the minterms of the function are marked with 1s.

Step 3 : The squares for don't care conditions are marked with Xs.

Step 4 : The 1s and Xs are combined in any convenient manner so as to enclose the maximum number of adjacent squares.

AB\CD	00	01	11	10
00	X	1	1	X
01		X	1	
11			1	
10			1	

**Fig. 2.5 (a)**

AB\CD	00	01	11	10
00	X			X
01	0	X		0
11	0	0		0
10	0	0		0

**Fig. 2.5 (b)**

Step 5 : Combining the squares with 1s and Xs on top in fig. (a), we get the term:  $A'B'$

Combining the four 1s we get the term:  $CD$

$F(A, B, C, D) = A'B' + CD$  is the solution in SOP.

Step 6 : The square marked with 0's represent the complement of the function. Combining 0's and Xs in fig (b) we get:

$$F' = D' + AC'$$

$$(F')' = F = (D' + AC')'$$

$$= D (A' + C) \text{ - (applying De Morgan's theorem)}$$

This is the simplified POS form of the function.



### CHECK YOUR PROGRESS

Q.5. What is a minterm?

.....

.....

.....

Q.6. What is a maxterm?

.....

.....

.....

Q.7. What is a Karnaugh map?

.....

.....

.....

Q.8. What do you understand by sum of products and product of sums?

.....

.....

.....

Q.9. How are the Don't care conditions useful?

.....

.....

.....

---

## 2.7 LET US SUM UP

---

- The three basic logic functions and operations are AND (logical multiplication), OR (logical addition), and the NOT (logical complementation) operation.
- Boolean addition is same as logical OR and Boolean multiplication is same as logical AND operation.
- The logical NOT operation changes logical 1 to 0 and vice versa.
- To get dual of any Boolean expression, you have to replace every 0 with 1 and every 1 with 0, and replacing every operator (+) with (.) and every (.) with (+).
- Any Boolean expression obtained by interchanging 0s and 1s and the operator (of an expression) is called the dual expression. This is the duality principle of Boolean algebra.
- De Morgan's theorem can be extended to any number of variables. It is useful in obtaining the complement of a Boolean function.
- We can simplify a Boolean function using algebraic manipulation method and map method. There is no straight forward procedure in the algebraic manipulation method.
- The K-map method is simple systematic and straight forward.
- Any Boolean function can be expressed as a sum of minterms (SOP) and as a product of maxterms (POS). Minterms are AND terms and maxterms are OR terms.
- The variables in a product term (minterm) and in a sum term (maxterm) may appear either in a complemented form or a normal form.
- A function is said to be completely specified, when for every possible combination of input variables the output is defined, otherwise the function is incompletely specified.
- In an incompletely specified function those combination of input variables for which output values are not known i.e. which minterms or maxterms are not used as parts of the output function are called Don't care terms.

- Don't care terms may be considered as 1s or as 0s, in simplifying a Boolean function.



## 2.8 FURTHER READINGS

1. Computer System Architecture, M.M. Mano, PHI
2. Digital logic and Computer Design, M.M. Mano, PHI
3. Computer Fundamentals Architecture and Organization, B. Ram, New Age International (P) Ltd.



## 2.9 ANSWER TO CHECK YOUR PROGRESS

**Ans. to Q. No. 1 :** Let the two variables are A and B.

The logical AND operations is denoted by the multiplication symbol ( $\cdot$ ). Thus,  $A \text{ AND } B = A \cdot B$ .

The logical OR operation is denoted by the symbol ( $+$ ), and  $A \text{ OR } B = A + B$

**Ans. to Q. No. 2 :** A truth table is a table, which has two sides, viz, input and output. In input side, we have all the possible combinations of input variables, i.e. for 2 variables, we have  $2^2 = 4$  input combinations, for 3, we have 8 combinations and so on. The output side gives us the result of the function or operation either 1 (TRUE) or 0 (FALSE). Output 1 indicates that the corresponding input combination (minterm) is present in the function.

Truth table for AND and OR operations

Input		Output	
A	B	$A \cdot B$	$A + B$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

**Ans. to Q. No. 3 :** The principle of duality states that for any Boolean expression, another valid expression can be obtained

by replacing each 0 with 1 and 1 with 0 and interchanging the binary operators (.) and (+). For any pair of expression so obtained, the original one is called the primal and the new one is called the dual expression. For example – the expression  $a b' + cd = 1$ , its dual is  $(a + b')(c + d) = 0$ .

**Ans. to Q. No. 4 :** The De Morgan's theorem for two variables A and B can be state as:

I.  $(A + B)' = A' B'$

II.  $(AB)' = A' + B'$

i.e. complement of an expression can be obtained by complementing each literal and interchanging the binary operators (.) and (+).

Truth Table

Proof :

A	B	A'	B'	A'B'	(A+B)	(A+B)'	AB	(AB)'	A'+B'
0	0	1	1	1	0	1	0	1	1
0	1	1	0	0	1	0	0	1	1
1	0	0	1	0	1	0	0	1	1
1	1	0	0	0	1	0	1	0	0

**Ans. to Q. No. 5 :** A minterm is an AND term also called product term. Minterms are obtained by combining the variables, which may appear in their normal form or complement form, with an AND operation. For example, for the two variables “a” and “b”, we have four minterms, viz,  $a'b'$  ( $m_0$ ),  $a'b$  ( $m_1$ ),  $a b'$  ( $m_2$ ) and  $a b$  ( $m_3$ ).

**Ans. to Q. No. 6 :** A maxterm, on the other hand an OR term, also called sum term. Maxterms are obtained by combining the variables with an OR operation. As for example, for two variables “a” and “b”, we have four maxterms and they are:  $a+b$  ( $M_0$ ),  $a+b'$  ( $M_1$ ),  $a'+b$  ( $M_2$ ), and  $a'+b'$  ( $M_3$ ).



**Note :** In case of maxterm, observe that – when  $a=0$ ,  $b=0$  the maxterm is:  $a + b$  i.e.  $a$  and  $b$  appears in normal form.

**Ans. to Q. No. 7 :** The Karnaugh map is a pictorial or graphical method used to simplify a Boolean function in an orderly and systematic manner. It is convenient to use 2 to maximum 6 variable map, but complexity in handling the map increases as the number of variables increase.

**Ans. to Q. No. 8 :** We have already seen that minterms are product terms. One of the important properties of Boolean algebra is that: any Boolean function can be expressed as a sum of minterms/product terms (SOP). Those minterms, which are present in a function are combined with OR operation and this is called sum of product representation of the function. Example:  $F(x, y, z) = x'y'z + x'y'z + x'yz$ .

On the other hand, product of sum (POS) form of a Boolean function is obtained by combining the maxterms present in the function with AND operations, Example:  $F(x, y, z) = (x + y' + z)(x' + y + z)(x' + y' + z)$ .

**Note :** Since each maxterm is the complement of its corresponding minterms and vice versa, the SOP form of Boolean function is also the complement of its POS form and vice versa.

**Ans. to Q. No. 9 :** Don't care conditions are plotted in the K-map using the symbol "X". If we can get a larger area by considering these Xs, (i.e. more squares can be combined) then we get a more simplified form of the function. Thus, don't care condition helps in further simplification of a function.



## 2.10 MODEL QUESTIONS

### Short – Answer Questions :

- Q.1. What are the basic operations of Boolean algebra?
- Q.2. State the De Morgan's Theorem.
- Q.3. What are the different ways of representing a Boolean function?
- Q.4. How are the terms “maxterm” and “minterms” related to each other?
- Q.5. What is the canonical form of a Boolean function?
- Q.6. What is the standard form of a Boolean function?
- Q.7. How can you convert a function from one canonical form to other?
- Q.8. The squares/minterms of a 4 variable map are marked as:  $m_0, m_1, m_3, m_2$  (in the first row), not as  $m_0, m_1, m_2, m_3$ . Why?

### Long-Answer Questions :

- Q.1. Problem: Simplify the following Boolean functions in SOP form.
  - a)  $F(a, b, c) = (2, 3, 6, 7)$
  - b)  $F(w, x, y, z) = (7, 13, 14, 15)$
  - c)  $F(a, b, c, d) = (2, 3, 12, 13, 14, 15)$
  - d)  $xy z' + x y z + x' z'$
  - e)  $ab'c + abc' + a'b c + abc$
- Q.2. Problems: Consider the following truth table.

a	b	c	$F_1$
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

- a) Express the function  $F_1$  in product of sum form.
- b) Obtain the simplified function in sum of products.
- c) Obtain the simplified function in product of sums.

Q.3. Obtain the simplified expression in POS.

- a)  $F(a, b, c) = \pi(0, 1, 4, 5)$
- b)  $F(A, B, C, D) = \pi(0, 1, 2, 3, 4, 10, 11)$

Q.4. Simplify the Boolean function  $F$  using the Don't care condition  $d$  in

(i) SOP and (ii) POS.

(a)  $F = w'x'z' + w'y z + w'x y$

$$d = w'x y' z + w y z + w x' z'$$

(b)  $F = A' B' C + A' B' C' + A' B C D + B' C D' + A B' D'$

$$d = A' B C' D + A' B C D' + A C D$$

---

## UNIT 3 : LOGIC GATES

---

### UNIT STRUCTURE

- 3.1 Learning Objectives
- 3.2 Introduction
- 3.3 Logic Gates
  - 3.3.1 OR Gate
  - 3.3.2 AND Gate, 3.3.3 NOT Gate
  - 3.3.4 NAND Gate
  - 3.3.5 NOR Gate
  - 3.3.6 XOR Gate
  - 3.3.7 XNOR Gate
- 3.4 De Morgan's Theorem
- 3.5 Truth Table
- 3.6 Conversion of the Logic gates
- 3.7 Let Us Sum Up
- 3.8 FURTHER RBADING
- 3.9 Answer to Check Your Progress
- 3.10 Model Questions

---

### 3.1 LEARNING OBJECTIVES

---

After going through this unit, you will be able to :

- explain logic gates
- know about the three basic logic gates
- prepare truth table for AND, OR, NOT, NAND, NOR, XOR and XNOR gates
- obtain the boolean function from a truth table
- describe the NAND implementation of a Boolean function
- describe the NOR implementation of a Boolean function

---

## 3.2 INTRODUCTION

---

A logic gate is an electronic circuit, which is used or collectively can be used to transform a boolean function /algebraic expression into a logic diagram. Logic gates have only one output and atleast two inputs except for the NOT gate, which has only one input. The output signals appears only for certain combinations of input signals. Binary information available in the input lines are manipulated by the gates. Three basic logic circuits, commonly called gates are used to make logic decision : they are OR, AND and NOT circuit /gate. Logic gates are available in the form of various IC families and are the basic building block of various circuits. Each gate has a distinct graphic symbol and its operation can be described by means of an algebraic function. The input-output relationship of the binary variables for each gate can be represented in tabular form in a truth table.

---

## 3.3 LOGIC GATES

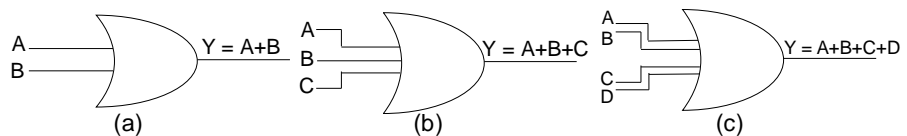
---

Before going to discuss about the functions of the logic gates, we have to know few basic things/terms that are associated with the functions of gates. Logic 1 and 0, that are applied as input or may be obtained as ouput of a gate, are represented by voltage levels. Positive logic (or active high levels) means that the most positive logic voltage level (also referred to as the high level) in defined to be the logic state1. On the other hand, the most negative logic voltage level (also referred to as the low level) is defined to be the logic state 0. Negative logic (or active law levels) is just the opposite, the most positive (high) level is 0, and the most negative (low) level is a 1. For instance, if the voltage levels are  $-0.1\text{v}$  and  $-5\text{v}$ , then in a positive logic system, the  $-5\text{v}$  level represents a zero and the  $-0.1\text{v}$  represents a 1. Conversely, if the voltage levels are  $0.1\text{v}$  and  $5\text{v}$ , then in a negative logic system, the  $5\text{v}$  levels represents a zero and the  $0.1\text{v}$  represents a one.

The choice of positive or negative logic is made by the individual logic designer. We cannot say one is advantageous over the other. It is common to see that most logic designers and text books on logic design use positive logic.

### 3.3.1 OR Gate

An OR gate has two or more inputs and a single output. It is an electronic circuit and the output of an OR gate is HIGH (logic1) if atleast one of the inputs is HIGH, otherwise (i.e. if all inputs are LOW) the output is LOW (0). Figure 3.1 shows the graphic symbol used for an OR gate.

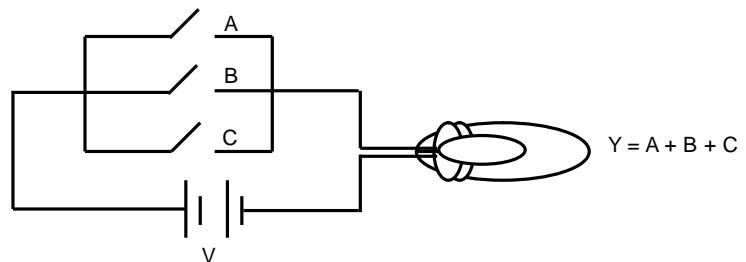


**Fig. 3.1 : Graphic symbols for (a) two inputs, (b) 3 inputs and (c) 4 inputs or gate**

The algebraic function for OR gate (2 inputs) is :  $F = x + y$ , and the truth table is :

**Table 3.1 Truth table of OR gate (2inputs)**

x	y	f
0	0	0
1	0	1
1	1	1



**Fig. 3.2 : Shows the switching circuit analogy of OR function**

The circuit of an OR gate is arranged in such a way that the output is in state 1, when anyone of the inputs is in state 1 ; i.e. when input A or input B or input C is 1 (in case of 3 inputs OR gate). The circuit can be illustrated by the analogy shown in Figure 3.2. The circuit consists of a battery, a lamp and three parallel switches connected in series. Battery switches are the inputs to the lamp and the light from the lamp represents the circuit output.

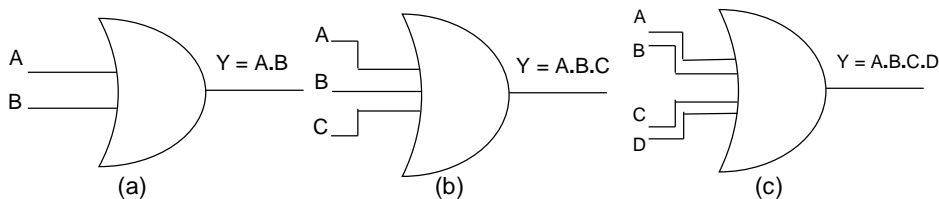
Let us define an open switch as a 0 state, i.e., 20 light represents 0 state, and a closed switch represents state 1, i.e., a glowing lamp as a 1 state. We can list the various combinations (8 combinations) of switch states as inputs to the circuit and the resulting output states in a truth table. It is clear from the truth table that all switches must be opened (0 state) for the light to be off (output in 0 state). This type of circuit is called an OR gate.

**Table 3.2 : Truth table of 3 inputs OR gate**

Inputs			Outputs
A	B	C	$Y = A + B + C$
0	0	0	0
0	0	1	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

### 3.3.2 AND Gate

An AND gate also may have two or more inputs and a single output. In order to have a HIGH (1) output, all the inputs of the AND gate must be HIGH (1), otherwise the output is LOW. Fig. 3.3 shows graphic symbols used for an AND gate.



**Fig. 3.3 : Graphic symbol for (a) two inputs (b) three inputs and (c) four inputs AND gate**

The algebraic function for a two inputs AND gate is :

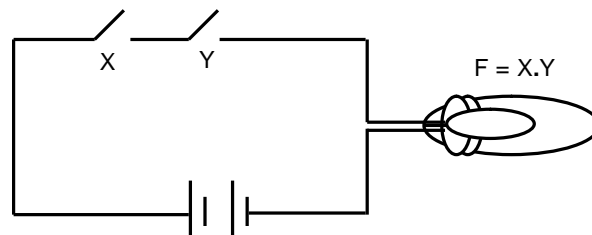
$$F = X + Y.$$

Table 3.3 shows the truth table for this function.

**Table 3.3 : Truth table of AND gate**

X	Y	F
0	0	0
0	1	0
1	0	0
1	1	1

The AND function can be explained by a series switching circuit as shown in fig. 3.4. It has two switches X and Y in series with a bulb and a power supply. The bulb will glow if and only if both the switches X and Y are simultaneously on.



**Fig. 3.4 : Switching circuit analogy of AND function**

In the truth table 3.3,

$x = y = 0$  represents that the switches are OFF

$x = y = 1$  represents that the switches are ON

$F = 0$  represents that the bulb will not glow

$F = 1$  represents that the bulb will glow.

### 3.3.3 NOT Gate

NOT gate circuit has a single input and single output. The NOT gate circuit is also called a complementary circuit or an inverter as it complements its input i.e. it accomplishes a logic negation. Figure 3.5 shows the different graphic symbols used for the NOT gate.



**Fig. 3.5 : Graphic symbols for NOT gate**

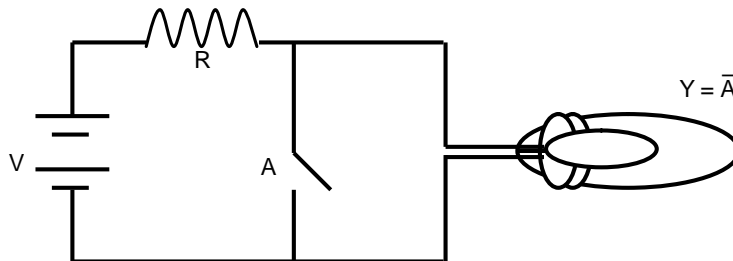


The algebraic function for a NOT gate is :  $F = x'$ . Table 3.4 is the truth table for a NOT gate.

**Table 3.4 : Truth table of NOT gate**

X	F
0	1
1	0

The NOT gate is understood by the short circuit switch A is closed, (ON) the bulb is bypassed and it does not glow, but when the switch is opened (OFF), the current will flow through the bulb and it would glow. i.e., when A is ON the bulb will be OFF and when A is OFF the lamp will be ON.



**Fig. 3.6 : Switching analogy of the NOT function**

### 3.3.4 NAND Gate

A NAND gate is the cascade combination of all AND and a NOT gate. It is an AND gate followed by an inverter. The NAND operation is the complement of the AND operation. Fig. 3.7 shows the graphic symbols for a NAND gate. The algebraic function is defined as :  $F = (xy)'$ . Table 3.5 shows the truth table for NAND operation.

**Table 3.5**

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

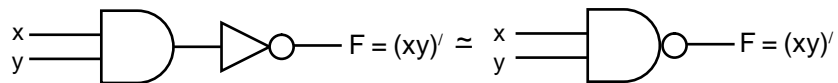


Fig. 3.7 : Graphic/Logic symbols of NAND gate

### 3.3.5 NOR Gate

The negation of the OR function is called NOT-OR or NOR. A NOR gate is the cascade combination of NOT and OR gates. The NOR operation is the complement of OR operation. The graphic symbols used normally for a NOR gate are shown in figure 3.8. The NOR function is defined as :  $F = (x+y)'$ .

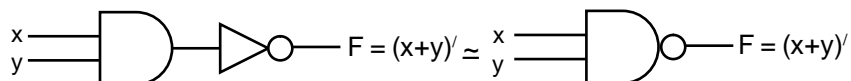


Fig. 3.8 : Graphic or logic symbols of NOR gate

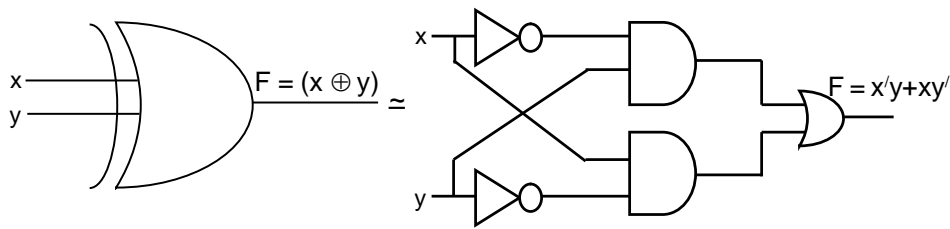
Table 3.6 : Truth table of NOR gate

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

**Note :** Any boolean function can be implemented/realized using NAND or NOR gates. So, NAND and NOR these two gates are called universal gates.

### 3.3.6 Exclusive - OR (XOR) gate

The exclusive - OR (XOR) gate has a graphic symbol similar to that of the OR gate, except for the additional curved line on the input side. The output of a two input XOR gate is a logic 1, if the input x or input y is a logic 1 exclusively, i.e., they are not 1 simultaneously. The graphic symbol is shown in fig. 3.9 and the XOR function can be written as  $F = X \oplus Y = X'Y + XY'$ .



**Fig. 3.9 : (a) Graphic symbol for XOR gate (b) XOR gate using basic gates**

The truth table of XOR operation is shown in table 3.7

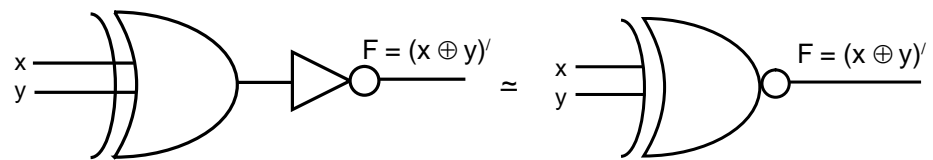
**Table 3.7 : Truth table for XOR operation**

X	Y	$F = X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

From the truth table, it is clear that the output is 1 (HIGH), when any one of the inputs is at 1 (HIGH). The output is 0 (LOW), when both the inputs are at 1 (HIGH) or at 0 (LOW), i.e. same. In case of more than two inputs, the output of a XOR gate is high when an odd number of inputs is HIGH, such as one or three or five etc. On the otherhand, when there is an even number of HIGH inputs, the output will be always LOW.

### 3.3.7 XNOR Gate

The XNOR i.e., exclusive NOR gate is the complement of the XOR gate, just discussed. XNOR function is also called equivalence function. The graphic symbol of XNOR gate is similar to that of the XOR gate, except for the additional inverter gate (or a small bubble) on the output side. The output of a two inputs XNOR gate is a logic 1 (HIGH), if both the inputs are either 1 (HIGH) or 0 (LOW). If the inputs are different (not same), the output is 0 (LOW). In general, we can say the output is 0 (LOW), when the inputs to an XNOR gate have an odd number of 1s. The graphic symbol of XNOR gate is shown in Fig. 3.10.



**Fig. 3.10 : Graphic symbols for XNOR or euivalence gate**

The truth table is given in Table 3.8. Here, we see that the output  $F$  is the complement of output of the XOR gate. The boolean function for XNOR gate is :

$$\begin{aligned}
 F = (x \oplus y)' &= (x'y + xy') \\
 &= (x'y)' (xy')' \text{ Applying DeMorgan's Theorem} \\
 &= (x + y') (x' + y) \\
 &= xx' + xy + x'y' + yy' \\
 &= xy + x'y'
 \end{aligned}$$

**Table 3.8 : Truth table for XNOR gate**

Inout		Output
x	y	$F = (x \oplus y)'$
0	0	1
0	1	0
1	0	0
1	1	1

### 3.4 DE MORGAN'S THEOREM

In unit –2, we have mentioned the De Morgan's Theorem and its proof for two variables also has been done using truth table. De Morgan, a great mathematician contributed two most important theorems of Boolean algebra. These two theorems are extremely useful in simplifying an expressing in which the product of the sum of variables is complementated. The two theorems can be extended to any number of variables and generalized as:

Theorem 1 :  $(A + B + C + \dots)' = A' \cdot B' \cdot C' \cdot \dots$

Theorem 2 :  $(A \cdot B \cdot C \cdot \dots)' = A' + B' + C' + \dots$

In words, we can write Theorem 1 as: The complement of an OR sum equals the AND product of the complements, and the theorem 2 as :

The complement of an AND product is equal to the OR sums of the complements.

The complement of any boolean function may be found by means of these theorems. It consists of two simple steps to form a complement of a function.

Step 1 : Interchange the symbols / operands “+” and “\*”.

Step 2 : Each and every term in the expression is complemented.

**Example : 3.1.** Find the complements of the following functions using De Morgan’s Theorem.

$$(i) \quad F = (A + B)' (A' + C') (B' + C)$$

$$(ii) \quad F = A'B + ABC + AB'C$$

**Solution :** (i)  $F = (A + B)' (A' + C') (B' + C)$

$$F' = [(A + B)' (A' + C') (B' + C)]$$

$$= (A + B) + (A' + C')' + (B' + C)'$$

$$\therefore F' = (A + B) + AC + BC' = A + B + AC + BC'$$

(ii)  $F = A'B + ABC + AB'C$

$$F' = (A'B + ABC + AB'C)'$$

$$= (A'B)' (ABC)' (AB'C)'$$

$$F' = (A + B') (A' + B' + C') (A' + B + C')$$

---

### 3.5 TRUTH TABLE

---

Basic idea of truth table is already given in UNIT –2. Here, we will see the process of deriving an expression from a truth table.

The general procedure for obtaining the expression from a truth table in sum of products (SOP) can be summarized as follows:

- 1 Write an AND term (minterm) for each combination of input variables in the table for which output is 1.
- 2 Each AND term contains each input variable either in normal form or in complemented form. If the corresponding variable is 0 then it is complemented in the AND term.
- 3 All the AND terms are then ORed together to produce the final output expression.

**Example : 3.2.** Obtain the logic function specified by the following truth table. Simplify it using algebraic manipulation and implement it with logic diagram.

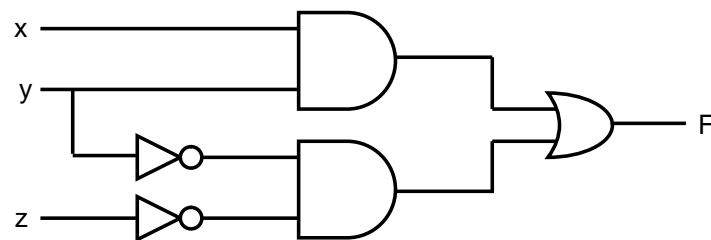
x	y	z	F
0	0	0	1 → $x'y'z'$
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1 → $xy'z'$
1	0	1	0
1	1	0	1 → $xyz'$
1	1	1	1 → $xyz$

**Solution :** The resultant boolean function is :

$$\begin{aligned}
 F &= x'y'z' + xy'z' + xyz' + xyz \\
 &= y'z' (x' + x) + xy (z' + z) \\
 &= y'z' + xy
 \end{aligned}$$

This is the simplified function.

The logic diagram is :



**Fig. 3.11 : Logic diagram for  $F = xy + y'z'$**

**Note :** Since the function has 2 AND terms, we need 2 AND gates and 1 OR gate to implement it.

### 3.6 CONVERSION OF LOGIC GATES

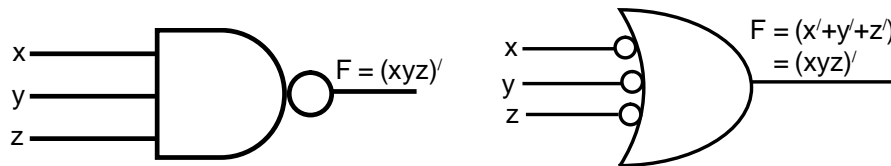
We already have discussed the functions of three basic gates, viz, OR, AND and NOT gates, which perform logical addition, logical multiplication and inversion operations respectively. Besides these, we also have come to know about NAND and NOR gates and their functions. In this

section, we will discuss how the digital circuits can be implemented with NAND or NOR gates.

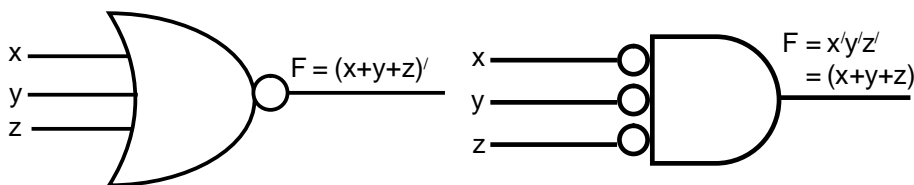
NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families. So, digital circuits are more frequently constructed with NAND or NOR gates than with AND and OR gates. Because of the prominence of NAND and NOR gates in the design of digital circuits, rules and procedures have been developed for the conversion from boolean functions given in terms of AND, OR and NOT into equivalent NAND or NOR logic diagram. Here, we will consider only the procedure for two level implementation.

First, we will define two other graphic symbols for NAND and NOR gates, which will make the conversion procedure easily understandable. Two equivalent symbols for the NAND gate are shown in Fig. 3.12 (a).

The AND invert symbol has been defined previously. It is possible to represent a NAND gate by an OR graphic symbol preceded by small circles in all the inputs. This symbol i.e. invert - OR symbol for the NAND gate follows from the De Morgan's theorem. and from the convention that small circle denote complementation.



(a) Two graphic symbols for NAND gate



(b) Two graphic symbols for NOR gate



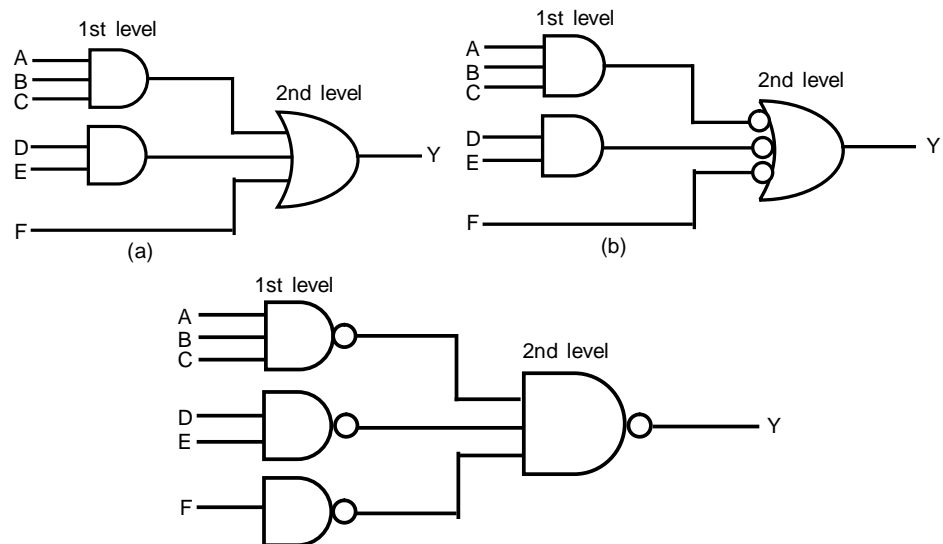
(c) Three graphic symbols for inverter

**Fig. 3.12 : Graphic symbols for NAND and NOR gates**

Two graphic symbol. The invert AND is an alternative that uses De Morgan's theorem and the convention that small circles in the inputs denote complementation.

A one input NOR gate or NAND gate is equivalent to an inverter. So, an inverter gate can be drawn in three ways as shown in fig. 3.12 (c).

**NAND Implementation :** To implement a boolean function with NAND gates, it is required that the function is to be simplified in the sum of products (SOP) form. We can see the relationship between a sum of products expression and its equivalent NAND implementation, by considering the logic diagrams of Fig. 3.13. All three diagrams are equivalent and implement the function :  $Y = ABC + DE + F$



**Fig. 3.13 : Three ways to implement  $Y = ABC + DE + F$**

The function is implemented in SOP form with AND and OR gates in Fig. 3.13 (a). The AND gates are replaced by NAND gates and the OR gate is replaced by a NAND gate with an invert OR symbol. The simple variable F is complemented and applied to the second level invert OR gate. A small circle represents complementation. Therefore two circles on the same line represent double complementation and both can be removed. The complement of F goes through a small circle which complements the variable again to produce the normal value of F. Thus, if we remove the small circles in the gates of fig. 3.13 (b), we get fig. 3.13 (a). Therefore, the two diagrams implement the same function and are equivalent.



In fig.3.13 (c), the output NAND gate i.e. the second level NAND gate is replaced with the conventional symbol. The one input NAND gate complements variable F. The diagram in (c) is equivalent to the one in (b), which in turn is equivalent to the diagram in (a). Thus, we implement the circuit, with NAND gates in fig. 3.13 (b) or 3.13 (c), which is first implemented with AND and OR gates in fig 3.13 (a).

The NAND implementation can also be verified algebraically as:

$$\begin{aligned} Y &= [(ABC)' \cdot (DE)' \cdot F']' \\ &= [(A' + B' + C') (D' + E') \cdot F']' \\ &= ABC + DE + F \end{aligned}$$

From, the transformation shown in Fig. 3.13 we see that a boolean function can be implemented with two levels of NAND gates. The rule for obtaining the NAND logic diagram from a boolean function is as follows:

1. Simplify the function in sum of products (SOP).
2. For each product term of the function that has atleast two literals, draw a NAND gate.
3. Draw a single NAND gate (using the AND invert or INVERT -OR graphic symbol) in the second level, with inputs coming from outputs of first -level gates.
4. A term with a single variable requires our inverter in the first level or may be complemented and applied as an input to the second-level NAND gate.

There is a second way to implement a boolean function with NAND gates. We have already shown in Unit -2 that combining the 0's in a map we obtain the simplified expression of the complement of the function in sum of products. The complement of the function so obtained can then be implemented with two levels of NAND gates using the above stated rules. To obtain the normal output of the circuit, it is required to insert a one input NAND or inverter gate to generate the true value of the output variable. When the designer wants to generate the complement of the function, the second method is preferred.

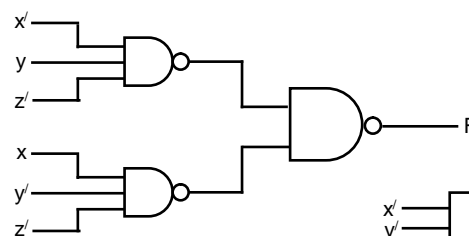
**Example 3.3 :** Implement the following function with NAND gates  $F(x, y, z) = \sum(2, 4)$ .

The first step is to simplify the function in sum of products form. We draw a map and plot the terms (Fig. 3.14).

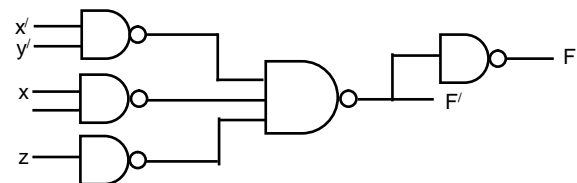
There are only two 1's in the map, and they cannot be combined. The simplified form of the function in SOP is  $F = x'yz' + xy'z'$

		00	01	11	10
0	$x'yz'$	0	0	0	1
1		1	0	0	0

(a) Map simplification in SOP  $F = x'yz' + xy'z'$



(b)  $F = x'yz' + xy'z'$



(c)  $F' = x'y' + xy + z$

**Fig. 3.14 : Implementation of Function in eg. 3.3 with NAND gates**

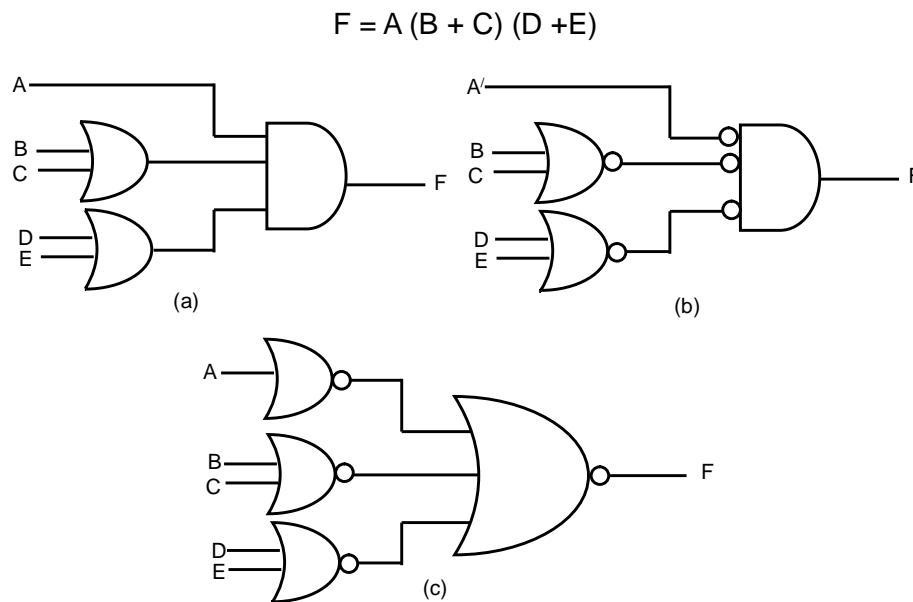
The two level NAND implementation is shown in Fig. 3.14 (a) Now, we try to simplify the complement of the function in SOP. This is done by combining to 0's in the map: Thus

The two-level NAND gate for generating  $F$  is shown in Fig 3.14 (c). If output  $F$  is required, we have to add a one input NAND gate to invert the function. (We assume that the input variables are available in both the normal and complement forms).

**NOR Implementation :** The NOR function is the dual of the NAND function. So, all procedures and rules for NOR logic are the dual of the corresponding procedures and rules developed for NAND logic.

The implementation of a boolean function with NOR gates requires that the function be simplified in product of sums (POS) form. A product of

sums expression specifies a group of OR gates for the sum terms, followed by an AND gate to produce the product. The transformation from the OR-AND to the NOR-NOR diagram is shown in Fig. 3.15, which is similar to the NAND transformation discussed already, except that here we use the product of sums expression :



**Fig. 3.15 : Implementation of the function  $F = A(B + C)(D + E)$**

The procedure for obtaining the NOR logic diagram from a boolean function can be derived from this transformation. It is similar to the three step NAND rule, except that the simplified expression must be in the product of sums and the terms for the first level NOR gates are the sum terms. A term with a single variable requires a one input NOR or inverter gate or may be complemented and directly applied to the second-level NOR gate.

Another way to implement a function with NOR gate is to use the expression for the complement of the function in product of sums. It gives a two level implementation for F and a three level implementation gives the normal output F.

Simplified product of sums can be obtained from a table by combining the 0's and then complementing the function. To obtain the simplified product of sums expression for the complement of the function, we have to combine the 1's in the map and then complement the function. The NOR gate implementation procedure is demonstrated in the following example.

**Example 3.4 :** Implement the function of e.g. 3.3 with NOR gates.

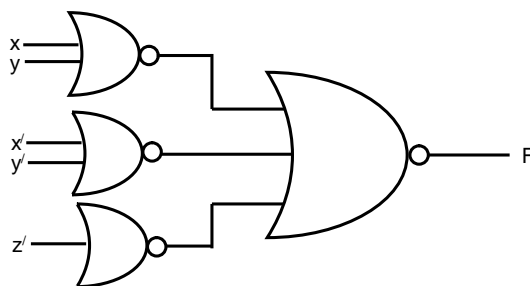
The map for this function is drawn in fig. 3.14 (a) Now, combining the 0's we obtain  $F' = x'y' + xy + z$

This is the complement of the function in SOP. To obtain it in POS form, as required for NOR implementation, complement  $F'$ .

$$(F')' = F = (x + y)(x' + y')z'$$

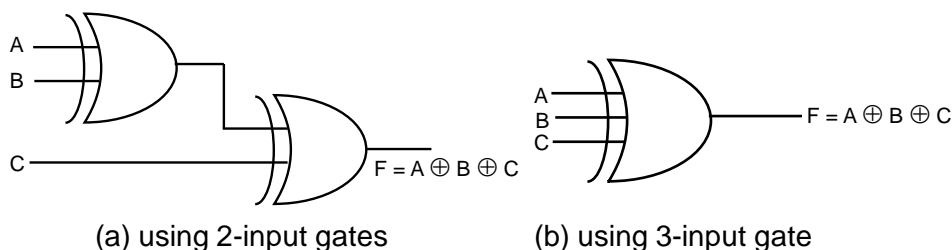
The two level implementation with NOR gate is shown in Fig. 3.16.

Another implementation that is possible from the complement of the function in POS is left as exercise.



**Fig. 3.16 : Implementation of function with NOR gates**

**Example 3.5 :** Draw the logic diagram for the function  $F = A \oplus B \oplus C$



### CHECK YOUR PROGRESS

Q.1. When the output of an OR gate is High?

.....

Q.2. When the output of an AND gate is HIGH?

.....

Q.3. What is the function of a NOT gate?

.....

Q.4. When the output of a XOR gate is HIGH?

.....

Q.5. Why the digital circuits are more frequently constructed with NAND or NOR gates?

.....

### 3.7 LET US SUM UP

- A logic gate is an electronic circuit that is used to implement a boolean function by a logic diagram.
- An OR gate produces a HIGH output when atleast one input is HIGH; whereas an AND gate produces a HIGH output when all inputs are HIGH.
- A NAND gate is an AND gate followed by an inverter. It produces a LOW output if all its inputs are HIGH.
- A NOR gate is an OR gate followed by an inverter. It produces a HIGH output when all its inputs are LOW.
- The realization of basic gates viz., AND, OR, and NOT can be made by using either NAND or NOR gates. For this reason, NAND and NOR gates are called universal gates.



### 3.8 FURTHER READING

- Mano, Morris 2007, *Digital Logic and Computer Design*. Pearson Education.
- Kumar, A. Anand. 2003, *Fundamental of Digital Circuit*, New Delhi, PHI.



### 3.9 ANSWER TO CHECK YOUR PROGRESS

**Ans. to Q. No. 1 :** If atleast one of the inputs is HIGH.

**Ans. to Q. No. 2 :** If all the inputs to it are HIGH.

**Ans. to Q. No. 3 :** The function of the NOT gate is to invert / complement its single line input variable or function.

**Ans. to Q. No. 4 :** If both the input variables to a XOR gate is not equal then the output is HIGH.

**Ans. to Q. No. 5 :** Because, NAND and NOR gates are easier to fabricate.



---

## 3.10 MODEL QUESTIONS

---

### Short - Answer Questions

- Q.1. What is a logic gate?
- Q.2. List the three basic logic operations.
- Q.3. What are the positive and negative logic?
- Q.4. What is the only set of input conditions that will produce a LOW output for an OR gate?
- Q.5. Write a truth table for a 3 input OR gate?
- Q.6. Write a Boolean expression for a 4 input AND gate
- Q.7. What is the only input combination that will produce a HIGH at the output of a 4 input AND gate?
- Q.8. What input/ logic level should be applied to the second input of a 2-input AND gate to inhibit the logic signal at the first input from reaching output?
- Q.9. Develop a truth table for a 3- input AND gate.
- Q.10. Is there any difference between  $1 \text{ OR } 1$  and  $1+1$  (binary addition) ?
- Q.11. Name the logic gate which has only one input, and show the logic / graphic symbol.
- Q.12. What are the NAND and NOR gates?
- Q.13. Write the logic symbols of NAND /NOR gates and develop its truth table.
- Q.14. What are the universal gates?
- Q.15. Draw the logic diagram of OR gate using NOR /NAND gate.
- Q.16. Draw the logic diagram of AND gate using NOR / NAND gate.

- Q.17. Draw the logic diagram of NOT gate using NOR / NAND gate
- Q.18. What is an XOR gate ? Write its truth table for 2 variables.
- Q.19. Show the logic diagram of an XOR gate using basic gates.
- Q.20. Draw the logic diagram of an XOR gate using NAND gates.
- Q.21. What is an XNOR gate ? Write its truth table.
- Q.22. Draw the logic diagram of an XNOR gate using basic gates.
- Q.23. Draw the symbol of an XNOR gate and its Boolean expression.

**Long - Answer Question :**

- Q.1. Implement the following functions with (i) NAND gates (ii) NOR gates.
- Q.2. Write the procedure for obtaining NAND logic implementation of a Boolean function.
- Q.3. Write the procedure for obtaining NOR logic implementation of a Boolean function.
- Q.4. Explain the operation of 3-input AND / OR gate and realize it using NAND /NOR gates
- Q.5. Explain the operation of 2-input XOR gate and realize it using NAND /NOR gates
- Q.6. Explain the operation of 2-input XNOR gate and realize it using NAND /NOR gates.

---

## UNIT 4 : COMBINATIONAL CIRCUIT

---

### UNIT STRUCTURE

- 4.1 Learning Objectives
- 4.2 Introduction
- 4.3 Adder Circuit
  - 4.3.1 Half-Adder
  - 4.3.2 Full-Adder
  - 4.3.3 Parallel Adder
  - 4.3.4 Serial Adder
- 4.4 Subtractor Circuit
  - 4.4.1 Half-Subtractor
  - 4.4.2 Full-Subtractor
- 4.5 Multiplexer
  - 4.5.1 2-to-1 Multiplexer
  - 4.5.2 4-to-1 Multiplexer
  - 4.5.3 8-to-1 Multiplexer
  - 4.5.4 16-to-1 Multiplexer
  - 4.5.5 Application of Multiplexer
- 4.6 Demultiplexer
  - 4.6.1 1-to-16 Demultiplexer
- 4.7 Encoder
  - 4.7.1 Octal-to-Binary Encoder
  - 4.7.2 Decimal-to-BCD Encoder
- 4.8 Decoder
  - 4.8.1 3-to-8 Decoder
- 4.9 Magnitude Comparator
- 4.10 Let Us Sum Up
- 4.11 Further Readings
- 4.12 Answer to Check Your Progress
- 4.13 Model Questions



---

## 4.1 LEARNING OBJECTIVES

---

After going through this unit, you will be able to :

- define a combinational circuit
- describe working principle of half-adder
- elaborate working principle of full-adder
- know about a parallel adder
- explain about a serial adder
- know the working of a half-subtractor
- describe working principle of full-subtractor
- explain the principle of multiplexer
- know about a demultiplexer
- describe working principle of encoder
- elaborate about a decoder

---

## 4.2 INTRODUCTION

---

In UNIT-3 of this block, you have already been accustomed with different logic gates and their associated Boolean expressions. Understanding of these would help you to follow the subject matter of this unit.

Various combinational circuits are discussed here so that you can understand different digital components used in many digital applications. One has to acquire knowledge on these components which will help you to understand the working principle of many digital devices.

The combinational circuits are those whose outputs are functions of its inputs. Its function can be described by one or more switching functions.

The advancement in technology has made it possible to fabricate cheap **SSI (Small Scale Integration)** circuits, **MSI (Medium Scale Integration)** functional devices and **LSI (Large Scale Integration)** systems. Example of these are single chip pocket calculator, single chip computer memories and single chip CPU etc. Many of these products now become building blocks to design other higher level digital devices.

In a combinational circuit since the output is related to the inputs by a Boolean expression, therefore a truth table is always associated with all combinational circuits. Conversely a Boolean expression can be obtained for a combinational circuit from its truth table. The linking of the logic circuit to its real circuit realization is done by mapping its logical inputs level of 0 and 1 to two-level voltage of the actual electronic circuit. The outputs are also available as either of the two level voltages which in turn represent logic 0 or 1.

### 4.3 ADDER CIRCUIT

Adder circuits are used to add binary bits. If we require to find sum of two 1-bit we use a circuit called *half-adder*. Some times it becomes necessary to add three 1-bit binary numbers. The circuit for this is called *full-adder*.

#### 4.3.1 Half-Adder

In the following table the *sum(S)* and *carry(C)* bits are shown as result which are obtained when two 1-bit number *X* and *Y* are added. The table contains all the possible combination of values of these two numbers. From the table two individual circuits can be constructed to obtain *sum* and *carry* in response to every combination of the two input numbers. Combining both a single circuit is constructed to treat it as a single combinational circuit to give us two outputs viz *sum* and *carry*. The circuit is generally termed as *half-adder*.

Hence we can say that a half-adder is a circuit that can add two binary bits. Its outputs are SUM and CARRY.

X	Y	CARRY (C)	SUM (S)
0	0	0	0
0	1	0	1 ( $\bar{X}Y$ )
1	0	0	1 ( $X\bar{Y}$ )
0	1	1 ( $XY$ )	0

Truth Table for a Half-Adder

The minterms for Sum and CARRY are shown in the bracket.

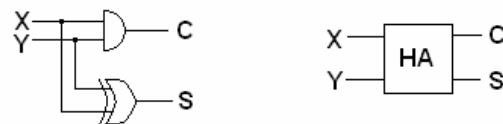
The Sum-Of-Product equation for SUM is :

$$S = \bar{X}Y + X\bar{Y} \dots\dots\dots (1)$$

Similarly the SOP equation for the CARRY is :

$$C = XY \dots\dots\dots (2)$$

Combining the logic circuits for equation ( 1 ) & ( 2 ) we get the circuit for Half-Adder as :



**Fig. 4.1 : Half-Adder Circuit and Symbol**

### 4.3.2 Full-Adder

To add two 2-bit binary numbers, we first add the two least significant bits of each number and take the carry ( if any ) that generate into the addition of the two most significant bits. Hence the second addition involves addition of three single bits. That necessitates another type of addition circuit which we will discuss here. This new type of adder circuit is known as *full-adder*.

Full-Adder is a logic circuit to add three binary bits. Its outputs are SUM and CARRY. In the following truth table X, Y, Z are inputs and  $\bar{C}$  and  $\bar{S}$  are CARRY & SUM.

X	Y	Z	CARRY ( $\bar{C}$ )	SUM ( $\bar{S}$ )
0	0	0	0	0
0	0	1	0	1 ( $\bar{X}\bar{Y}Z$ )
0	1	0	0	1 ( $\bar{X}Y\bar{Z}$ )
0	1	1	1 ( $\bar{X}YZ$ )	0
1	0	0	0	1 ( $X\bar{Y}\bar{Z}$ )
1	0	1	1 ( $X\bar{Y}Z$ )	0
1	1	0	1 ( $XY\bar{Z}$ )	0
1	1	1	1 ( $XYZ$ )	1 ( $XYZ$ )

**Truth Table for Full-Adder**

The minterms are written in the brackets for each 1 output in the truth table. From these the SOP equation for full summation can be written as :

$$\begin{aligned} S &= \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ \\ &= \bar{X}(\bar{Y}Z + Y\bar{Z}) + X(\bar{Y}\bar{Z} + YZ) \\ &= \bar{X}S + XS \dots\dots\dots (3) \end{aligned}$$

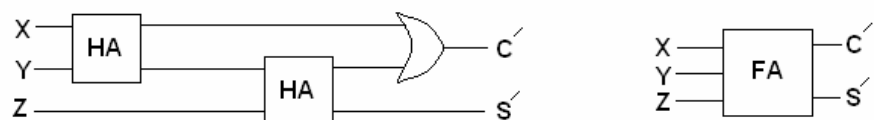
Here S is SUM of Half-Adder.

Again SOP equation for Full – Adder CARRY is :

$$\begin{aligned} C &= \bar{X}YZ + X\bar{Y}Z + XY\bar{Z} + XYZ \\ &= \bar{X}YZ + XYZ + X\bar{Y}Z + XY\bar{Z} \\ &= (\bar{X} + X)YZ + X(\bar{Y}Z + Y\bar{Z}) \\ &= YZ + XS \\ &= C + XS \dots\dots\dots (4) \end{aligned}$$

Here also C means CARRY of half-adder and S means SUM of half-adder.

Now using two half-adder circuits and one OR gate we can implement equation ( 3 ) and ( 4 ) to obtain a full-adder circuit as follows.



**Fig. 4.2 : Full-Adder Circuit and its Symbol**

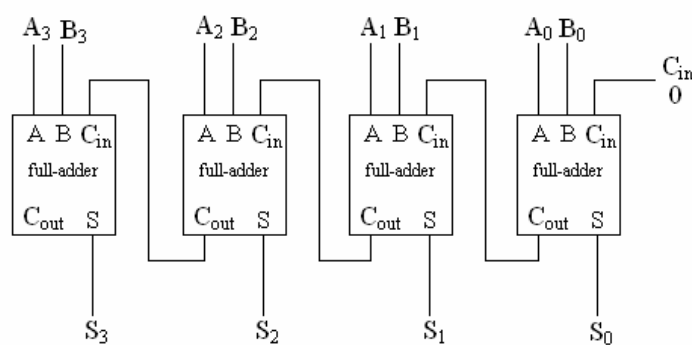
### 4.3.3 Binary Parallel Adder

When two multi-bit binary numbers are to be added under the situation that their bits are available in parallel form, then a parallel adder is used. It can add each corresponding bit of the numbers and simultaneously produce the sum bits as parallel output.

#### ● 4-BIT BINARY PARALLEL ADDER

Block diagram of a 4-bit binary parallel adder capable of adding two 4-bit numbers is shown in Fig 4.3. The numbers to be added are designated as  $A_3A_2A_1A_0$  and  $B_3B_2B_1B_0$  and the corresponding

sum bits are  $S_3S_2S_1S_0$ . To build the parallel adder, one full-adder is required for each pair of corresponding bits in the numbers to be added, except for the pair of least significant bits, for which a half-adder is sufficient. But to facilitate cascading the pair of least significant bits is also added by a full-adder. The carry out from each stage is taken as the carry into the next more significant stage. In figure 5.3 we use four full-adder to construct the 4-bit binary parallel adder where the carry in to the full-adder for the least significant bits is made logical 0.



**Fig. 4.3 : A 4-bit binary parallel adder**

#### 4.3.4 Serial Adder

A serial adder is used to add serial binary numbers. To implement a serial adder, one full-adder is required. In figure 4.4 a serial adder is shown. The bits to be added come serially one after another from both the numbers in synchronism, first  $A_0$  and  $B_0$ , then after one clock pulse  $A_1$  and  $B_1$  and so forth. The carry bit generated ( $C_{out}$ ) in the process is saved for one clock pulse by a flip-flop and it is added to the next higher order pair of input bits as  $C_{in}$ . It can be seen from the figure that the output sum bits are shifted into the output register as the input bits are shifted out of the input register. In practice, to minimize the circuit, the output bits are shifted in to one of the input registers behind the data being shifted out. The register which contains one of the binary numbers to be added before the commencement of addition and contains the sum after the addition process is completed, is called an *accumulator*.

It can be easily understood that the serial adder is slower than a parallel adder, as they require one clock pulse to add one pair of bits. A serial adder requires much lesser hardware than a parallel adder and hence it finds application in such devices where space is of much concern than speed. One such application is pocket calculator.

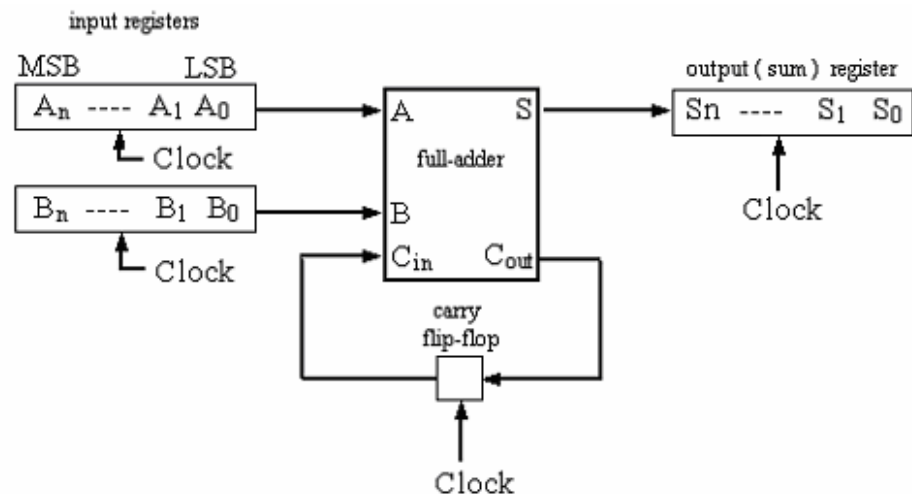


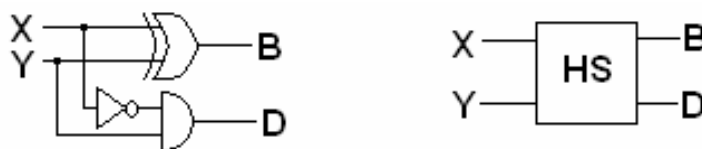
Fig. 4.4 : A serial adder

## 4.4 SUBTRACTOR CIRCUIT

In arithmetic expression subtraction is also a frequently occurring operation like addition. Hence it is necessary to design subtractor circuits involving two and three binary bits. Just like adder circuits, we have two subtractor circuits viz *half-subtractor* and *full-subtractor*.

### 4.4.1 Half-Subtractor

A half-subtractor subtracts one bit from another bit. It has two outputs viz DIFFERENCE (D) and BORROW (B).



Truth Table for Half-Subtractor

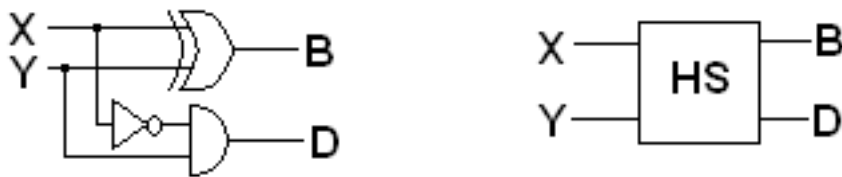
X	Y	BORROW (B)	DIFFERENCE (D)
0	0	0	0
0	1	1 ( $\bar{X}Y$ )	1 ( $\bar{X}Y$ )
1	0	0	1 ( $X\bar{Y}$ )
1	1	0	0

The mean terms are written within parenthesis for output 1 in each column. The SOP equations are :

$$D = \bar{X}Y + X\bar{Y}$$

$$= S \quad \dots\dots\dots (5)$$

$$B = \bar{X}Y \quad \dots\dots\dots (6)$$



**Fig. 4.5 : The half-subtractor circuit and the symbol**

#### 4.4.2 Full-Subtractor

A full-subtractor circuit can find difference and borrow arises on the subtraction operation involving three binary bits.

**Truth Table of Full-Subtractor**

X	Y	Z	BORROW (B')	DIFFERENCE(D')
0	0	0	0	0
0	0	1	1 ( $\bar{X}\bar{Y}Z$ )	1 ( $\bar{Y}\bar{X}Z$ )
0	1	0	1 ( $\bar{X}Y\bar{Z}$ )	1 ( $\bar{X}Y\bar{Z}$ )
0	1	1	1 ( $\bar{X}YZ$ )	0
1	0	0	0	1 ( $X\bar{Y}\bar{Z}$ )
1	0	1	0	0
1	1	0	0	0
1	1	1	1 ( $XYZ$ )	1 ( $XYZ$ )

The SOP equation for the DIFFERENCE is :

$$D' = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ$$

$$= \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XYZ + \bar{X}\bar{Y}Z$$

$$= (\bar{X}Y + X\bar{Y}) \bar{Z} + (XY + \bar{X}\bar{Y}) Z$$

$$= D\bar{Z} + DZ \dots\dots\dots (7)$$

And SOP equation for BORROW is :

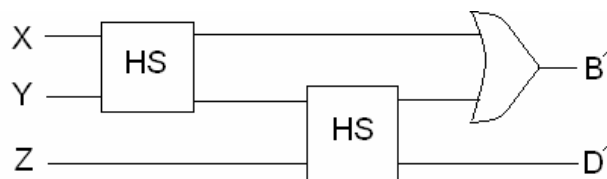
$$B/ = \bar{X}\bar{Y}Z + \bar{X}Y\bar{Z} + \bar{X}YZ + XY\bar{Z}$$

$$= \bar{X}\bar{Y}Z + XY\bar{Z} + \bar{X}Y\bar{Z} + \bar{X}YZ$$

$$= (\bar{X}\bar{Y} + XY) Z + \bar{X}Y (\bar{Z} + Z)$$

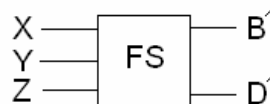
$$= \bar{D}Z + \bar{X}Y \dots\dots\dots (8)$$

In equation (7) and (8), D stands for DIFFERENCE of half-subtractor. Now from the equations (7) and (8) we can construct a full-subtractor using two half-subtractor and an OR gate.



**Fig. 4.6 : Full-Subtractor circuit**

The symbol of full-subtractor is :



### CHECK YOUR PROGRESS

- Q.1. A half-adder can add :
- Two binary number of 4 bit each
  - Two binary bit
  - Add half of a binary number
  - None of these
- Q.2. A full-adder is a logic circuit that has two output namely:
- product & Sum
  - sum & borrow
  - sum & carry
  - carry & borrow



- Q.3. A parallel adder can be implemented using
- (a) Two or more half-adder
  - (b) Two or more full-adder
  - (c) One full-adder and remaining half-adder
  - (d) none of these.
- Q.4. A serial adder use
- (a) One full-adder
  - (b) One half-adder
  - (c) One full-adder with a flip-flop
  - (d) One full-adder, one flip-flop, three shift registers
- Q.5. A half-subtractor can perform
- (a) subtraction of two binary bits
  - (b) product of two binary bits
  - (c) complement of half binary bits
  - (d) none of these
- Q.6. A full-subtractor has the ability to do
- (a) subtraction of two binary numbers
  - (b) subtraction of three binary bits
  - (c) product of three binary bits
  - (d) division three binary bits

---

## 4.5 MULTIPLEXER

---

In the simplest term multiplexer is a combinational circuit which perform sharing. A digital data multiplexer is a logic circuit that has many inputs and only one output. It can select any one of its many inputs by applying a control signal and steer the selected input to the output. A multiplexer is also called a data selector. Generalized block diagram of a multiplexer is shown in Figure 4.7

It has  $2^n$  inputs,  $n$  – numbers of control lines and only one output. A multiplexer is also called a many – to – one data selector.

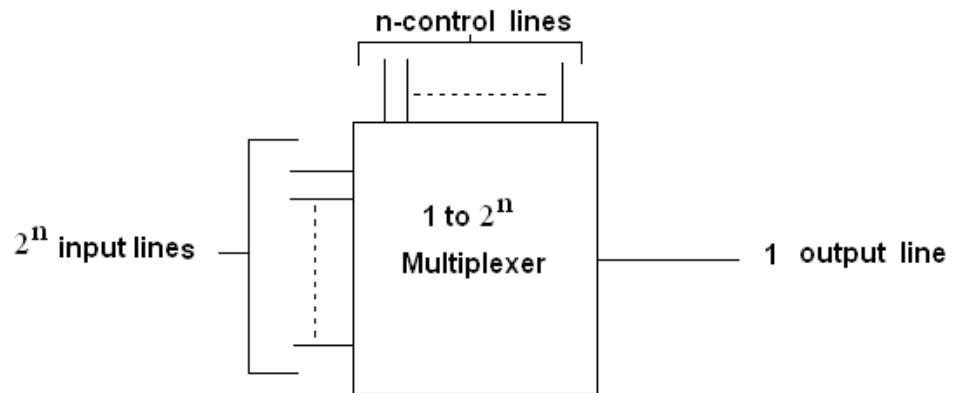


Fig. 4.7 : Block Diagram of Multiplexer

#### 4.5.1 2-to-1 Multiplexer

A 2-to-1 multiplexer has 2 inputs and a single control line as shown in figure 4.8. Here data inputs are denoted by  $I_0$  and  $I_1$ . The control signal  $A$  specifies which input is routed to the output. When  $A = 0$ , then  $I_0$  is routed to the output and when  $A = 1$  then  $I_1$  is routed to the output.

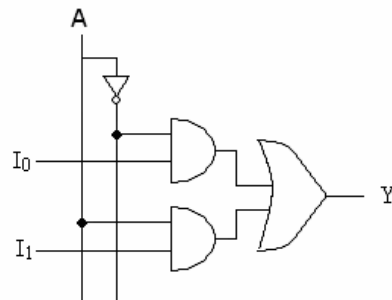
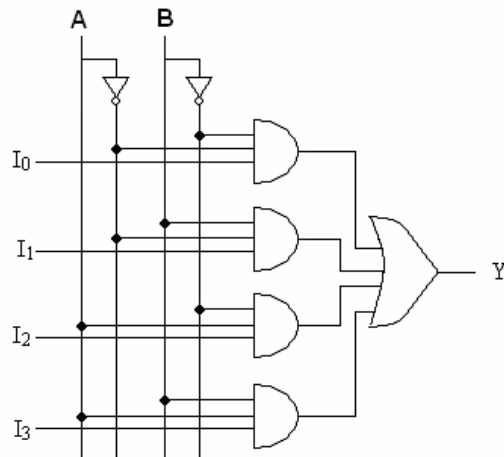


Fig. 4.8 : 2-to-1 Multiplexer

#### 4.5.2 4-to-1 Multiplexer

To construct a 4-to-1 multiplexer we use two control signals which can control four inputs. The two bit binary number at the control  $A$  and  $B$  specifies which of the four data inputs is to go to the output. The four data inputs are termed as  $I_0$ ,  $I_1$ ,  $I_2$  and  $I_3$ . If  $AB = 00$ , then  $I_0$  is allowed to go to the output. If  $AB = 01$ , the  $I_1$  is allowed to go to the output. In this manner  $AB = 10$  will take  $I_2$  and  $AB = 11$  will take  $I_3$  to the output. The circuit for the 4-to-1 multiplexer is shown in figure 4.9.

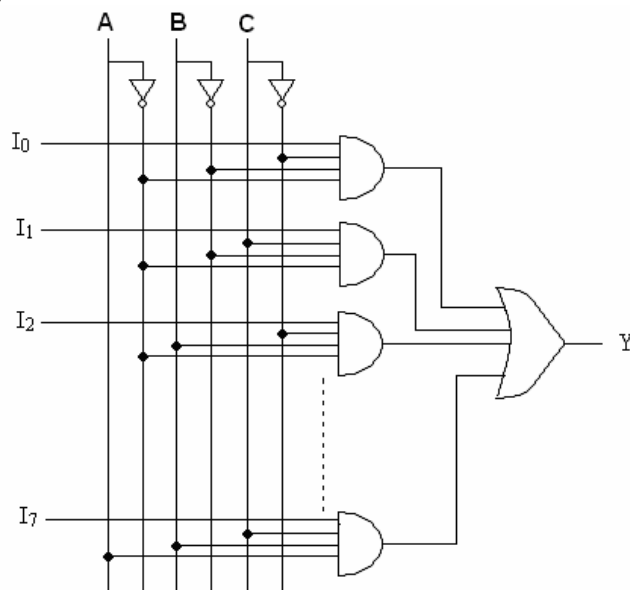


**Fig. 4.9 : 4-to-1 Multiplexer**

### 4.5.3 8-to-1 Multiplexer

Fig 4.10 shows a 8-to-1 multiplexer, where there are 8 inputs, 3 control lines and 1 output. The eight inputs are labeled as  $I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7$  and the control lines are as A, B, C. Which input is steered to the output depends on the value of ABC. As for example, if  $ABC = 000$

then the upper AND gate is enabled and all other AND gates are disabled. As a result, the input  $I_0$  alone is steered to the output. Similarly if  $ABC = 110$



**Fig. 4.10 : 8-to-1 Multiplexer**

Then the AND gate connected to the data line  $I_6$  is enabled while all the other AND gates are disabled. Therefore, the input  $I_6$  appears at the output. Hence when  $ABC = 110$ , the output is  $Y = I_6$

#### 4.5.4 16-to-1 Multiplexer

Fig 4.11 shows a 16 – to – 1 multiplexer. In this circuit, there are 16 data input lines, 4 control lines and 1 output. The input lines are denoted by  $I_0, I_1, I_2, I_3, \dots, I_{15}$  and the control lines are by ABCD. The output is denoted by Y.

Out of 16 input lines, only one is transmitted to the output depending upon the value of ABCD. If

$$ABCD = 0000$$

then  $I_0$  is steered to the output since the upper AND gate is enabled alone and all others are disabled. Similarly if

$$ABCD = 0010$$

then  $I_2$  appears at the output. If

$$ABCD = 1111$$

then the last AND gate is only enabled and therefore  $I_{15}$  appears at the output.

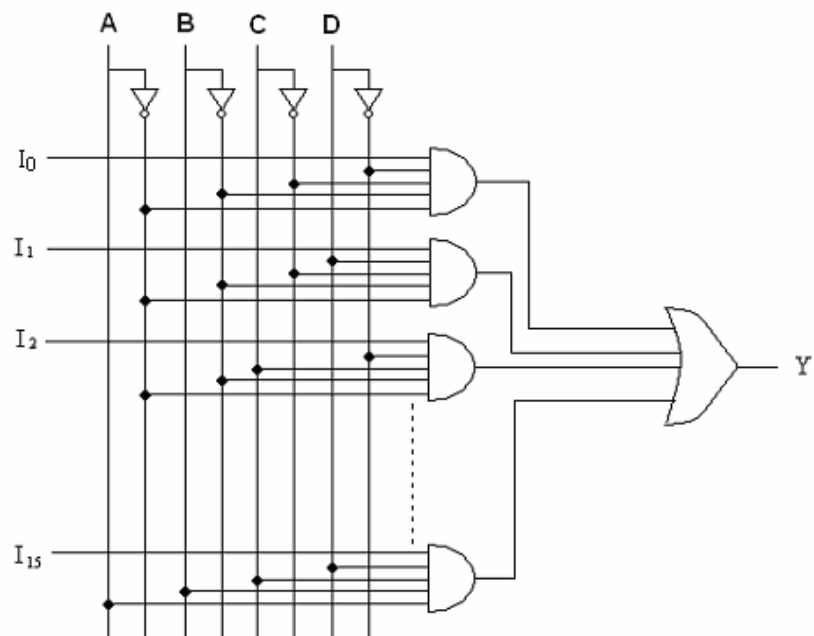


Fig. 4.11 : 16-to-1 Multiplexer

### 4.5.5 Multiplexer Application

- FUNCTION GENERATOR**

A multiplexer can perform any logic function which otherwise needs logic gates to implement. It can be connected in a manner so it duplicates the logic of any truth table. In such application, the multiplexer is viewed as a function generator. One advantage of this use of a multiplexer is that a single IC can perform a function that might need many ICs. Other advantage is that the implemented function can be easily changed if required due to changing necessity.

In the design of a function generator using a multiplexer first the truth table is to construct that is corresponding to the logic expression to be implemented. The next step is to connect logic 1 to each input of the multiplexer corresponding to each combination of input variables for which there is 1 in the output of the truth table. Then to connect logic 0 to all remaining inputs of the multiplexer. The variables of the truth table are used as control signals. The application can be best understood by one example. Here we implement the logic function

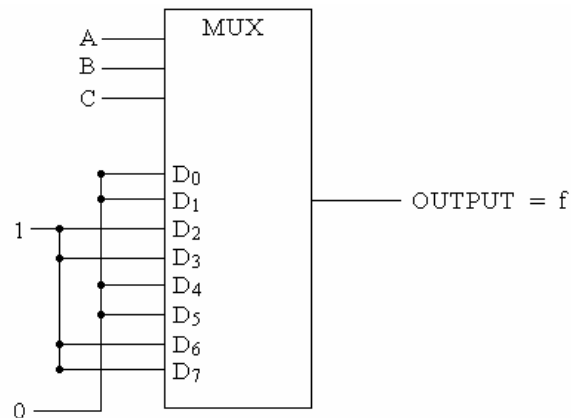
$$f = AB + \bar{A}BC + B\bar{C}$$

The truth table for the function is :

**Truth Table**

A	B	C	AB	$\bar{A}BC$	$B\bar{C}$	$f = AB + \bar{A}BC + B\bar{C}$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	1	1
0	1	1	0	1	0	1
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	1	1
1	1	1	1	0	0	1

Since the expression has three variables we use these three variables as three control or selector signals. As such the multiplexer must have  $2^3 = 8$  data inputs line. The Boolean expression has output 1 for  $ABC = 010, 011, 110$  and  $111$ . We therefore connect logical 1 to data inputs 2, 3, 6 and 7 of the multiplexer and logical 0 to all the remaining inputs. Figure 4.12 shows the circuit of the function generator for the chosen Boolean expression.



**Fig. 4.12 : Multiplexer as a function generator**

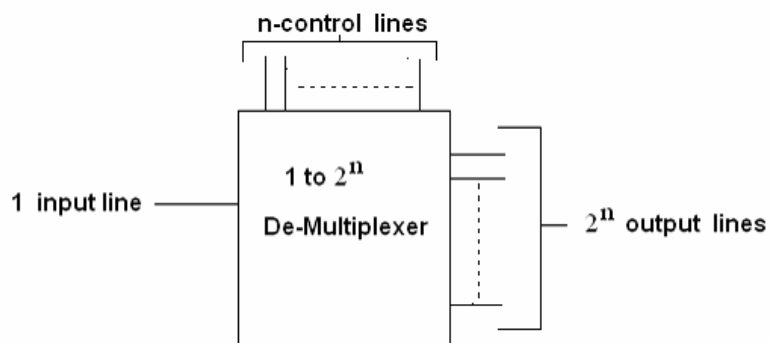
- **MULTIPLEXING SEVEN-SEGMENT DISPLAY**

Seven-segment displays are often used as optical output display in many applications. These displays consume considerable power. To minimize power consumption in small portable device like pocket calculator, where several such displays are to be illuminated simultaneously, multiplexers are used. In this technique all the displays are not illuminated simultaneously, instead they are illuminated one after another in a round robin fashion. If all the displays are illuminated about 30 times per second, the human eye will not be able to detect any flicker, rather the displays appear as illuminated steadily. The power consumption is greatly reduce in this technique.

## 4.6 DE-MULTIPLEXER

A demultiplexer steers a single input to one of many outputs. It is opposite to the multiplexer. De-multiplexer means One-to-Many. It can be

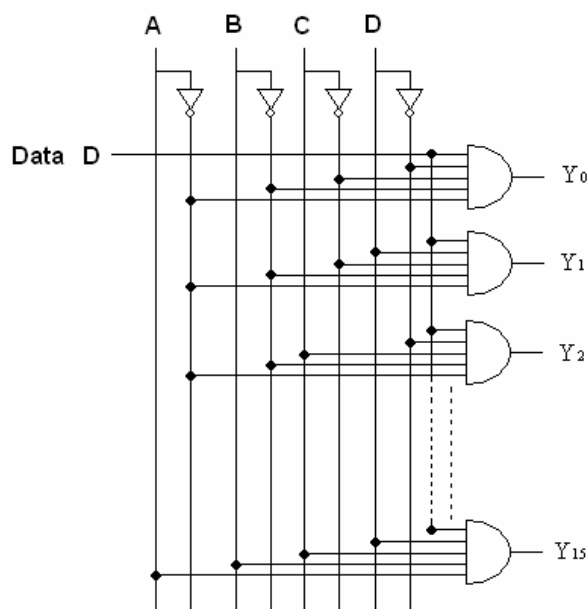
visualized as a distributor, since it distributes the same data to different output terminals. As such it has 1 input and many outputs. With the application of appropriate control signal, the common input data can be steered to one of the output lines. A good example of demultiplexer is the distributor of an automobile ignition system where electric pulse is distributed among different fuel ignition plugs. Fig 4.13 shows a generalized block diagram of a de-multiplexer. To have  $2^n$  output lines, there must be  $n$  control lines in a de-multiplexer.



**Fig. 4.13 : Block diagram of a De-Multiplexer**

#### 4.6.1 1-to-16 De-Multiplexer

In Fig 4.14 we have shown a 1-to- 16 de-multiplexer.



**Fig. 4.14 : 1-to-16 De-Multiplexer**

Here the data input line is denoted by D. This input line is connected to all the AND gates through which output appears. Depending upon the control signal, only one AND gate becomes enabled and the data input D appears through that AND gate. So, when  $ABCD = 0000$ , the upper AND gate is enabled and data input D appears at  $Y_0$  as output.

When  $ABCD = 1111$ , the bottom AND gate becomes enabled and D appears at  $Y_{15}$  as output. For other combination, D appears at other output terminal.



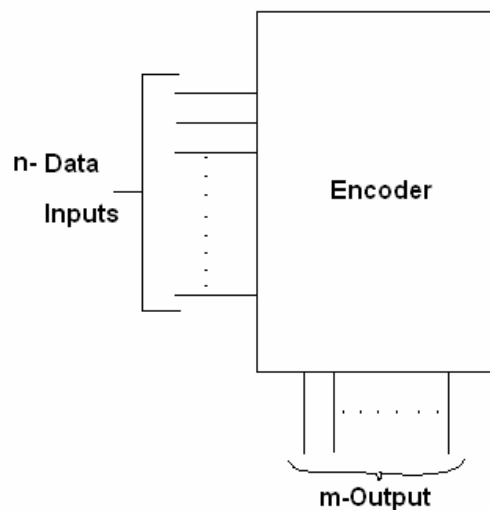
### CHECK YOUR PROGRESS

- Q.7. Multiplexer means :
- |                      |                 |
|----------------------|-----------------|
| (a) multiple to many | (b) one to many |
| (c) many to one      | (d) one to one  |
- Q.8. A 16-to-1 multiplexer has
- |                     |                     |
|---------------------|---------------------|
| (a) 1 control lines | (b) 2 control lines |
| (c) 3 control lines | (d) 4 control lines |
- Q.9. De-multiplexer means
- |                          |                 |
|--------------------------|-----------------|
| (a) deduct multiple bits | (b) one-to-many |
| (c) multiple-to-multiple | (d) one-to-one  |
- Q.10. To implement the logic function  $F = AC + BC + \overline{A}\overline{B}\overline{C}$  we need a
- |                        |                         |
|------------------------|-------------------------|
| (a) 2-to-1 multiplexer | (b) 4-to-1 multiplexer  |
| (c) 8-to-1 multiplexer | (d) 16-to-1 multiplexer |

## 4.7 ENCODER

An encoder is a device whose inputs are decimal digits and / or alphabetic characters and its outputs are the coded representations of the inputs. A generalized view of an encoder is shown in Fig 4.15.



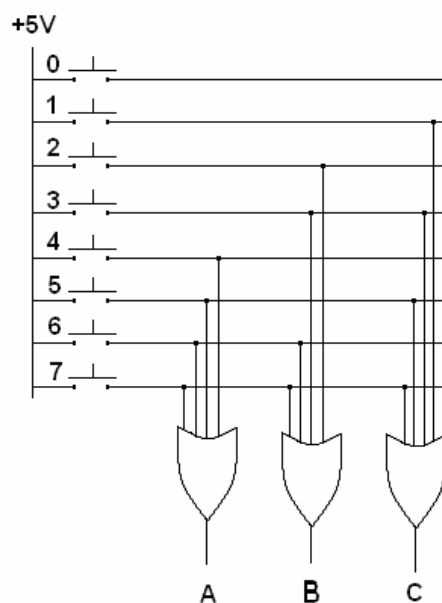


**Fig. 4.15 : Block Diagram of an Encoder**

In the figure we can see that there are  $2^n$  input lines and  $n$  – numbers of output lines. Out of inputs, only one input line is active at a time. Encoder generates a coded output which is unique for each of the active input.

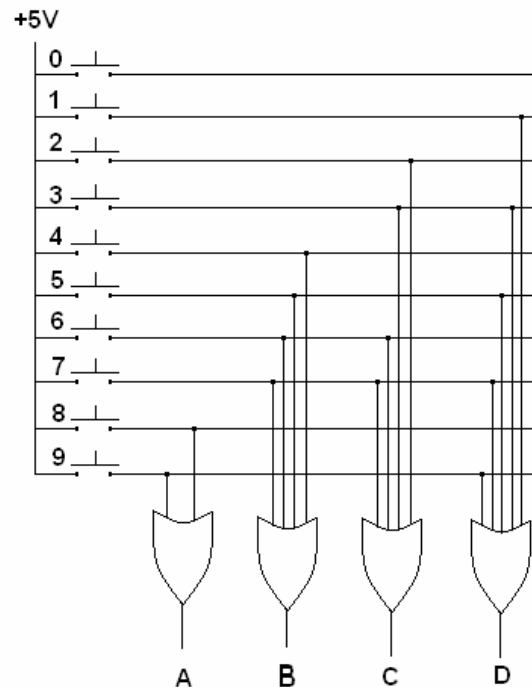
#### 4.7.1 Octal-To-Binary Encoder

An octal-to-binary encoder is a device which takes octal number as input and generates the equivalent binary number. Figure 4.16 shows a circuit of an octal-to-binary encoder. It is assumed here that any one input keys is pressed at a time. The keys are connected to a +5 Volt. When the input key is pressed, 5 volt signal is passed to the horizontal lines to which three OR gates are connected selectively.



**Fig. 4.16 : Octal-to-Binary encoder**

### 4.7.2 Decimal-To-BCD Encoder



**Fig. 4.17 : Decimal-to-BCD Encoder**

A decimal to BCD encoder is shown in Fig 4.17. This circuit generates BCD output when any one of the push button switches is pressed. As for example, if button 6 is pressed, the B and C OR gates have high inputs and the corresponding output becomes

$$ABCD = 0110$$

If button 8 is pressed, the OR gate A receives a high input and therefore the output becomes

$$ABCD = 1000$$

## 4.8 DECODER

A decoder is a digital circuit which has  $n$ -input lines and  $2^n$  output lines. A decoder and a de-multiplexer has similarity. In a de-multiplexer, there is a single input line connected to every output 'AND' gate whereas in a decoder the input line is absent. Various kinds of decoder are constructed to decode signals given in different forms. The input to a decimal decoder is a binary code and the output is the decimal digit the code represents. Some decoders take 3 bits binary inputs and give octal outputs.

In practical applications, decoders are used to select one of many devices in a computer system. For example a computer system has many peripheral devices like hard disk drive, DVD drive, printer etc for each of which there exist an unique address. Generally all the devices are connected to the computer system through a decoder. The computer generates the address of a particular device when it wants to access it in the form of binary code. The decoder receives that address as input and activates the lines connecting the particular device to make it enable.

#### 4.8.1 3-to-8 Decoder

Let us draw a 3 – to – 8 decoder as shown in Fig 4.18. It's truth table is shown below :

**Truth table for a 3-to-8 decoder**

Inputs			Outputs							
A	B	C	$Y_0$	$Y_1$	$Y_2$	$Y_3$	$Y_4$	$Y_5$	$Y_6$	$Y_7$
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

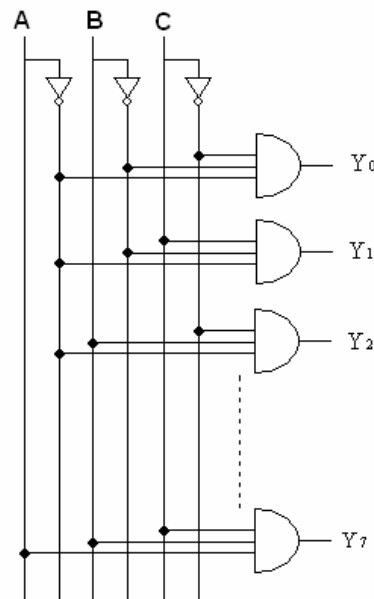


Fig. 4.18 : 3- to -8 decoder

## 4.9 MAGNITUDE COMPARATOR

A magnitude comparator circuit can compare two binary numbers to determine which one is greater than the other or their equality. Such a magnitude comparator has three output lines for  $A > B$ ,  $A = B$ ,  $A < B$  where  $A$  &  $B$  are two  $n$ -bits binary numbers. Every bit of one number is compared with the corresponding bit of the other number by ExOR gate.

A 4-bit magnitude comparator, SN 7485 is available in chip form the block diagram of which is shown in Fig 4.19. It compares two 4-bit binary numbers  $A_3A_2A_1A_0$  and  $B_3B_2B_1B_0$ . Three output terminals are available for  $A < B$ ,  $A = B$  and  $A > B$

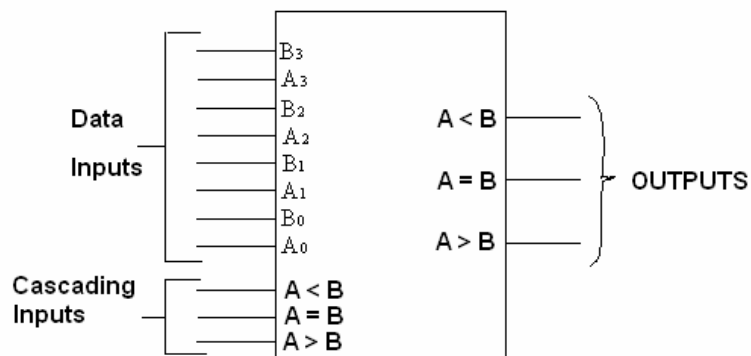


Fig 4..19 : Two 4-bit words magnitude comparator SN 7485

**FUNCTION TABLE SN 7485**

Comparing Inputs				Cascading Inputs			Outputs		
$A_3B_3$	$A_2B_2$	$A_1B_1$	$A_0B_0$	$A > B$	$A < B$	$A = B$	$A > B$	$A < B$	$A = B$
$A_3 > B_3$	X	X	X	X	X	X	H	L	L
$A_3 < B_3$	X	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	L	L	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	H	L	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	L	H	L	L	H

To compare any number having more than 4-bits, two or more such chip can be cascaded. The  $A > B$ ,  $A < B$  and  $A = B$  outputs of a stage that handles less significant bits are connected to the corresponding cascading inputs of the next stage that handles the more significant bits.

**CHECK YOUR PROGRESS**

Q.11. An encoder :

- (a) converts a digital input to another form of digital output.
- (b) converts analog input to digital output
- (c) selects one out of many inputs    (d) none of these.

Q.12. Decoder has  $n$  inputs line and

- (a)  $n$  output lines                      (b)  $2^n$  output lines.
- (c)  $n^2$  output lines                      (d) no output lines.

Q.13. Magnitude comparator

- (a) compares two multi bit binary number
- (b) magnify any digital signal
- (c) compress binary numbers.
- (d) check error in a binary number.

---

## 4.10 LET US SUM UP

---

- A combinational circuit is some combinations of logic gates as per specific relationship between inputs and outputs.
- Adder and subtractor circuits can perform binary addition and subtraction.
- A multiplexer is a combinational circuit which selects one of many inputs.
- Demultiplexer is opposite to a multiplexer.
- An encoder generates a binary code for  $2^n$  input variables.
- A decoder decodes an information receives from n input lines and transmits the decoded information to maximum outputs.
- A magnitude comparator circuit can compare two binary numbers to determine which one is greater than the other or their equality.



---

## 4.11 ANSWER TO CHECK YOUR PROGRESS

---

- Ans. to Q. No. 1 :** (b)  
**Ans. to Q. No. 2 :** (c)  
**Ans. to Q. No. 3 :** (b)  
**Ans. to Q. No. 4 :** (d)  
**Ans. to Q. No. 5 :** (a)  
**Ans. to Q. No. 6 :** (b)  
**Ans. to Q. No. 7 :** (c)  
**Ans. to Q. No. 8 :** (d)  
**Ans. to Q. No. 9 :** (b)  
**Ans. to Q. No. 10 :** (c)  
**Ans. to Q. No. 11 :** (a)  
**Ans. to Q. No. 12 :** (b)  
**Ans. to Q. No. 13 :** (a)



---

## 4.12 FURTHER READING

---

- Mano, M. M., *Digital Logic and Computer Design*, PHI
- Mano, M. M., *Computer System Architecture*, PHI
- Malvino, Albert Paul & Leach, Donald P., *Digital Principles and Applications*, Mcgraw-Hill International
- Lee, Samuel C, *Digital Circuits and Logic Design*, PHI
- Talukdar, Dr. Pranhari, *Digital Techniques*, N. L. Publications
- Bogart Theodore F. Jr, *Introduction to Digital Circuits*, Macmillan.



---

## 4.13 MODEL QUESTION

---

- Q.1. Define combinational circuit with example.
- Q.2. With truth table and logic diagram explain the working of a full-adder circuit.
- Q.3. With truth table and logic diagram explain the working of a full-subtractor circuit.
- Q.4. Draw and explain the operation of a 4-bit parallel adder.
- Q.5. With block diagram explain how a serial adder works.
- Q.6. What do you mean by multiplexer ? With diagram explain the working of a 8-to-1 multiplexer.
- Q.7. Use an 8-to-1 multiplexer to generate the logic function
- Q.8. Explain the principle of an encoder. Draw a decimal-to-BCD encoder.
- Q.9. What is a decoder ? Draw and explain the working of a 3-to-8 decoder.
- Q.10. What do you mean by magnitude comparator? Draw block diagram and the function table of the magnitude comparator SN 7485.

---

## UNIT 5 : SEQUENTIAL CIRCUIT

---

### UNIT STRUCTURE

- 5.1 Learning Objectives
- 5.2 Introduction
- 5.3 Flip-Flop
  - 5.3.1 RS Flip-Flop
  - 5.3.2 D Flip-Flop
  - 5.3.3 JK Flip-Flop
  - 5.3.4 MS Flip-Flop
- 5.4 Counters
  - 5.4.1 Asynchronous Counters
  - 5.4.2 Synchronous Counters
- 5.5 Registers and its Types
  - 5.5.1 Serial-in-serial-out Register
  - 5.5.2 Serial-in-parallel-out Register
  - 5.5.3 Parallel-in-parallel-out Register
  - 5.5.4 Parallel-in-serial-out Register
  - 5.5.5 Applications of Shift Register
- 5.6 Let Us Sum Up
- 5.7 Further Readings
- 5.8 Answers to Check your progress
- 5.9 Model Questions

---

### 5.1 LEARNING OBJECTIVES

---

After going through this unit, you would be able to :

- draw different flip-flops
- describe the working principle of flip-flop
- distinguish between asynchronous and synchronous counter
- define MOD of a counter
- know the name of different registers
- draw the different registers



---

## 5.2 INTRODUCTION

---

If the output of a circuit depends on its present inputs and immediate past output, then the circuit is called sequential circuit. To build a sequential circuit, we need memory circuits along with some other logic gates. Flip-Flop is used as memory circuit the application of which we would see in counter, register etc.

From the previous unit it became evident that a combinational circuit has no memory element. A combinational circuit has some basic differences with a sequential circuit. These differences are listed below:

Combinational Circuit	Sequential Circuit
<ul style="list-style-type: none"><li>• It contains no memory element</li><li>• The present value of its outputs is determined solely by the present values of its inputs</li><li>• Its behavior is described by the set of output functions</li></ul>	<ul style="list-style-type: none"><li>• It contains memory elements</li><li>• The present value of its outputs is determined by the present values of its inputs and its present state</li><li>• Its behavior is described by the set of next-state functions and the set of output functions.</li></ul>

In this unit, several sequential circuits are discussed to make you acquainted with their principle of operations and applications.

---

## 5.3 FLIP-FLOPS

---

A digital circuit that can produce two states of output, either high or low is called a multi-vibrator. Multivibrators are further divided into three types, viz. monostable, bi-stable and astable.

A Flip-Flop is a bi-stable multivibrator and therefore it has two stable states of output-either high or low. It is a device that can be triggered to high state or triggered to low state. Depending on its inputs and previous output, its new output is either high (or 1) or low (or 0). The applied inputs must be kept in the input terminals of the flip-flop so that the inputs can affect the output. Once the output is fixed, the inputs can be removed and then also the already fixed output will be retained by the flip/flop. Hence a

flip-flop can be used as a basic circuit of memory for storing one bit of data. To store multiple bits we can use multiple flip-flops. Such a set of flip-flops is called register.

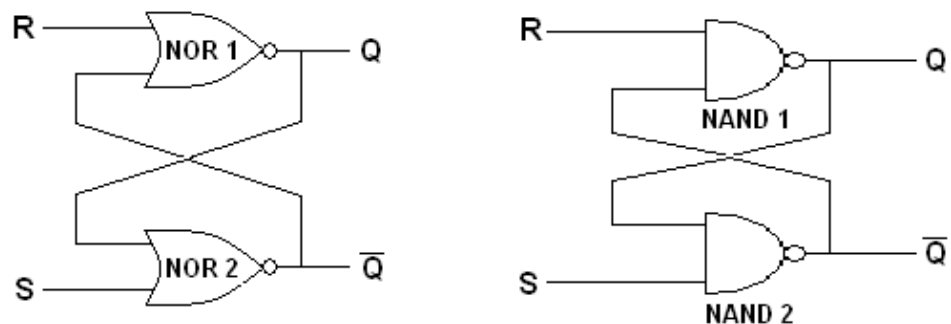
Flip-flop also serves as a fundamental component of another important digital circuit called counter. This device can keep track of events that occur in a digital circuit such as the number of pulses that a circuit can generate in a given time period. It can also divide frequency of a clock pulse which can lead to a digital clock circuit.

Flip-flops are constructed in a number of way, each type having specific characteristics necessary for a particular application. Here we discuss the following types of flip-flops :

- RS Flip-flop
- D Flip-Flop
- JK Flip-Flop
- MS Flip-Flop

### 5.3.1 RS Flip-Flop

RS Flip-Flop is also called Set-Reset Flip-Flop. An RS flip-flop can be built using many different circuits. Here we have shown two RS flip-flop circuits using NOR and NAND gate. To study its working principle, we can use any one of them. In the following section, let us consider the RS flip-flop using NOR gate. Note that the NOR gates are cross-coupled, that is one of the inputs of each gate is the output of the other gate.



**Fig. 5.1 : Basic circuit of RS Flip-Flop using NOR and NAND gate**

RS flip-flop has two inputs—Set(S) and Reset(R). It has two outputs, Q and  $\bar{Q}$ . It should be noted  $\bar{Q}$  is always the complement of Q. Though both outputs are not used in all the applications, the availability of both makes the flip-flop more versatile. Various combinations of inputs and their corresponding outputs are listed in the truth table below :

R	S	Q	Action
0	0	last value	No change
0	1	1	set
1	0	0	Reset
1	1	?	Forbidden

**Truth Table of RS Flip-Flop**

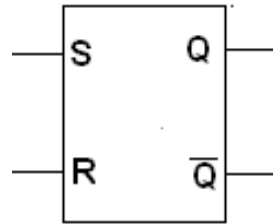
To understand the operation of the flip-flop let us assume that the flip-flop is initially *SET*, that means  $Q = 1$  and  $\bar{Q} = 0$ . Now the first input condition in the table i.e.  $R = 0, S = 0$ , will not change the state of the flip-flop. We can verify that by observing the fact that the input to the NOR gate 2 are 0 and 1, which keeps  $\bar{Q}$  at 0. Since  $\bar{Q}$  is 0, the input to the NOR gate 1 are both 0, making  $Q = 1$ . So we can say  $R = S = 0$  has no effect on its output, the flip-flop retains its previous state. Hence Q remains unchanged.

The second input condition  $R = 0, S = 1$  makes no change. The output of gate 2 remains low when both inputs are made high. Thus a 1 at the S input and 0 at R input will SET the flip-flop and Q will be equal to 1.

The third input condition  $R = 1, S = 0$  will reset the flip-flop. The high input to gate 1 causes its output Q to change from 1 to 0. That 0 is coupled to gate 2, which now has two low inputs. Therefore its output changes from 0 to 1. that 1 is coupled to gate 1 and therefore Q is held at 0. Hence, when  $R = 1, S = 0$ , then  $Q = 0, \bar{Q} = 1$ . Thus the flip-flop is RESET.

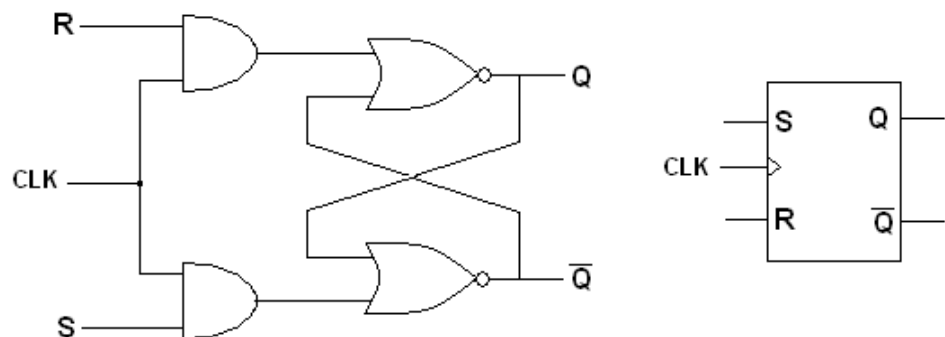
The last input condition in the table  $R = 1, S = 1$  is forbidden since it forces both the NOR gates to the low state, means both  $Q = 0$ , and  $\bar{Q} = 0$  at the same time, which violets the basic definition of

flip-flop that requires Q to be the complement of. Hence this input condition is forbidden and its output is unpredictable.



**Fig. 5.2 : Symbol of RS Flip-Flop**

**CLOCK INPUT :** For synchronization of operation of multiple flip-flop, an additional signal is added to all types of flip-flop which is called clock signal, generally abbreviated as CLK. Addition of CLK signal ensures that, what ever may be the input to the flip-flop, it effects the output only when CLK signal is given. Fig 5.3 shows a clocked RS flip-flop.



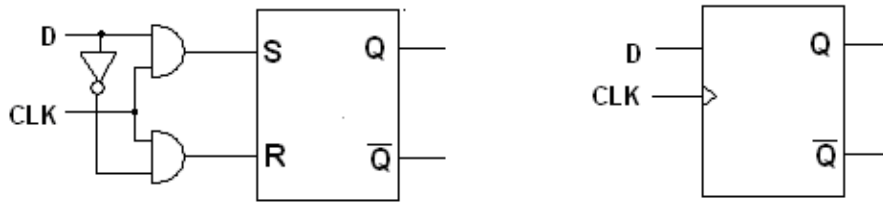
**Fig. 5.3 : Clocked RS Flip-Flop      Symbol of Clocked RS Flip-Flop**

**SET-UP TIME :** When the clock is low, the changes in R and S inputs does not effect the state of a flip-flop. To effect the output, the R and S input must remain constant for a minimum time before a clock pulse is applied. This minimum time is called *set-up time*. Its value is given in manufacturers' specifications.

### 5.3.2 D FLIP-FLOP

D flip-flop is a modification of RS flip-flop. In RS flip-flop when both the inputs are high i.e.  $R = 1$ ,  $S = 1$ , the output becomes unpredictable and this input combination is termed as forbidden. To avoid this situation, the RS flip-flop is modified so that both the

inputs can not be same at a time. The modified flip-flop is called D flip-flop. Fig 5.4 shows a clocked D flip-flop.



**Fig. 5.4 : (a) Clocked D Flip-Flop      (b) Symbol of D Flip-Flop**

In D flip-flop both inputs of RS flip-flop are combined together to make it one by a NOT gate so that inputs can not be same at a time. Hence in D flip-flop there is only one input. The truth table is:

CLK	D	Q
0	X	Last state
1	0	0
1	1	1

**Truth Table of D Flip-Flop**

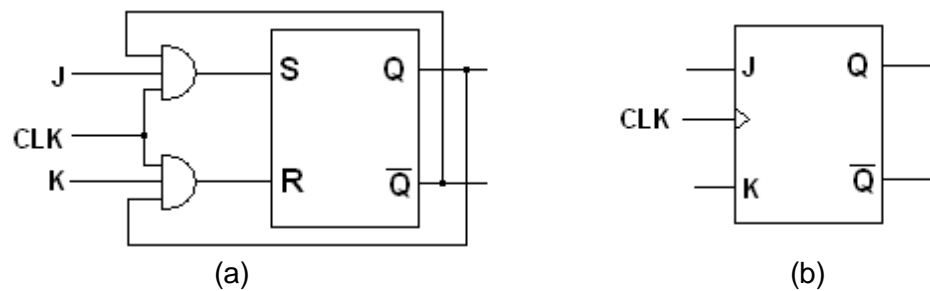
In a clocked D flip-flop the value of D cannot reach the output Q when the clock pulse is low. During a low clock, both AND gates are disabled, therefore, D can change value without affecting the value of Q. On the other hand, when the clock is high, both AND gates are enabled. In this situation, Q is forced to be equal to the value of D. In another way we can say that in the D flip-flop above, Q follows the value of D while the clock is high. This kind of D flip-flop is often called a D latch.

---

### 5.3.3 JK FLIP-FLOP

---

In RS flip-flop, the input  $R = S = 1$  is called forbidden as it causes an unpredictable output. In JK flip-flop this condition is used by changing the RS flip-flop in some way. In JK flip-flop both input can be high simultaneously and the corresponding toggle output makes the JK flip-flop a good choice to build counter- a circuit that counts the number of +ve or -ve clock edges. Fig 5.5 shows one way to build a JK flip-flop.



**Fig. 5.5 : (a) JK Flip-Flop, (b) Symbol of JK Flip-Flop**

CLK	J	K	Q
X	0	0	Last state
↑	0	1	0
↑	1	0	1
↑	1	1	Toggle

**Truth table for JK Flip-Flop**

The inputs J and K are called control inputs because their combinations decide what will be the output of JK flip-flop when a +ve clock pulse arrives. When J and K are both low, both the AND gates are disabled. Therefore the CLK pulse has no effect. The first input combination of the truth table shows this and under this case the output Q retains its last state.

When J is low, K is high, the upper AND gate is disabled while the lower AND gate is enabled. Hence the flip-flop cannot be set; instead it is reset, i.e.  $Q = 0$ . This is shown by the second entry in the truth table.

When J is high, K is low the upper AND gate is enabled while the lower one is disabled. So the flip-flop is set, thereby making  $Q = 1$ .

When J and K are both high, then the flip-flop is set or reset depending on the previous value of Q. If Q is high previously, the lower AND gate sends a RESET trigger to the flip-flop on the next clock pulse. Then Q becomes equal to 0. On the other hand if Q is low previously, the upper AND gate sends a SET trigger to the flip-flop making  $Q = 1$ .

So, when  $J = K = 1$ ,  $Q$  changes its value from 0 to 1 or 1 to 0 on the positive clock pulse. This changing of  $Q$  to  $\bar{Q}$  is called toggle. Toggle means to change to the opposite state.

Any flip-flop may be driven by +ve as well as -ve clock. As such JK flip-flop can also be driven by positive clock as well as negative clock. Fig 5.6 shows symbol of positive clocked and negative clocked JK flip-flop.

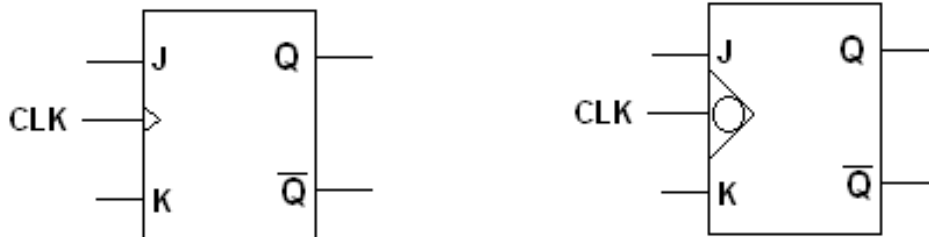


Fig. 5.6 : Positive clocked JK Flip-Flop

Negative clocked JK Flip-Flop

**RC Differentiator Circuit :** The clock pulse applied to a flip-flop is a square wave signal. Clock pulse is used to achieve synchronization of different flip-flops operation. The time span of the square wave is a longer time in comparison to the response time of flip-flops. Therefore the time span of the clock pulse is reduced by converting its shape to a narrow spike, so that the flip-flops can change their states at an instant rather than transiting their states one after another in a lethargic manner. It makes the synchronization more precise. The square wave pulse is modified to a narrow spike by using a RC differentiator circuit as shown in fig 5.7

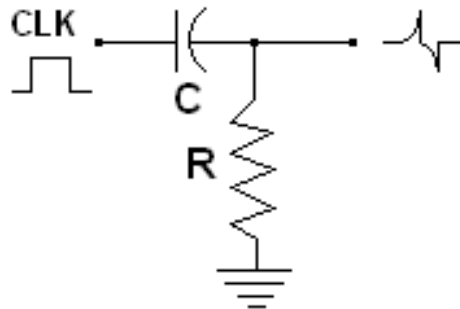


Fig. 5.7 : RC Differentiator Circuit

The upper tip of the differentiated pulse is called positive edge and the lower tip is called is negative edge. When a flip-flop is

triggered by this type of narrow spike, it is called edge triggered flip-flop. If the flip-flop is driven by +ve edge, it is called +ve edge triggered flip-flop. If it is driven by negative edge, it is called negative edge triggered flip-flop.

**Racing :** In a flip-flop if the output toggles more than ones during a clock edge than it is called racing. All flip-flop has a propagation delay, means the output changes its state after a certain time period from applying the input and the clock pulse. So when a flip-flop is edge triggered, then due to propagation delay the output cannot affect the input again, because by that time the edge of clock pulse has already passed away. If the propagation delay of a flip-flop is 20 ns and the width of the spike is less than 20 ns, then the returning  $Q$  and  $\bar{Q}$  arrive too late to cause false triggering.

---

#### 5.3.4 MS Flip-Flop

---

MS flip-flop is another way to avoid racing. Fig 5.8 shows one way to build MS flip-flop using two JK flip-flop, one of which is positive edge triggered and the other is negative edge triggered. The first JK flip-flop is master and the later is slave. The master responds to its J and K inputs at the positive edge and the slave responds in the negative clock edge. When  $J=1$ ,  $K=0$ , the master sets on the positive clock edge. The high Q output of the master drive the j input of the slave. So, at the negative clock edge, the slave also sets ,copying the action of the master.

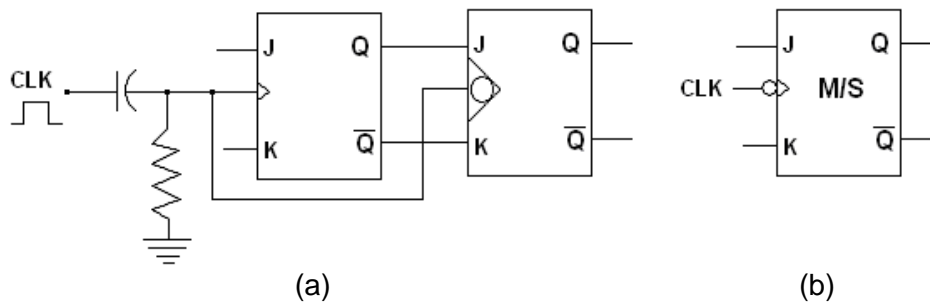
When  $J=0$ ,  $K=1$ , master resets at +ve clock edge and the slave resets at the –ve clock edge.

When  $J=K=1$  master toggles at +ve clock edge, and the slave toggles at the –ve clock edge.

Hence whatever master does, the slave copies it.

MS flip-flop is a very popular flip-flop in industry due to its inherent resistance to racing. Hence to build counter it is extensively used.





**Fig. 5.8 (a) Edge triggered JK MS Flip-Flop,  
(b) Symbol of JK MS Flip-Flop**



### CHECK YOUR PROGRESS

- Q.1. A flip-flop is basically a  
 (a) mono-stable multi-vibrator (b) astable multi-vibrator  
 (c) bi-stable multi-vibrator (d) none of these
- Q.2. JK flip-flop has the specialty in  
 (a) fast response time (b) toggle property  
 (c) spike shaped clock input (d) preset input
- Q.3. In MS flip-flop the master changes state  
 (a) after the slave (b) with the slave at the same time  
 (c) before the slave (d) never
- Q.4. In edge triggering the clock pulse used is  
 (a) Square wave (b) Triangular wave  
 (c) Narrow spike (d) None

## 5.4 COUNTER

A counter is one of the most useful sequential circuit in a digital system. A counter driven by a clock can be used to count the number of clock cycles. Since the clock pulse has a definite time period, the counter can be used to measure time, the time period or frequency.

A counter can also be used as a frequency divider circuit. In this application, a counter can reduce the frequency of a clock pulse.

There are basically two types of counter—Synchronous counter and asynchronous counter.

Counter are constructed by using flip-flops and other logic gates. If the flip-flops are connected serially then output of one flip-flop is applied as input to the next flip-flop. Therefore this type of counter has a cumulative settings time due to propagation delay. Counter of this type is called serial or asynchronous counter. These counters have speed limitation.

Speed can be increased by using parallel or synchronous counter. Here, flip-flops are triggered by a clock at a time and thus setting time is equal to the propagation delay of a single flip-flop. But this type of synchronous counters require more hardware and hence these are costly.

Combination of serial and parallel counter is also done to get an optimize solution of speed and hardware cost. If each clock pulse advances the count of the counter by one, it is called up counter. If the count of the counter goes down at each clock pulse, it is called down counter.

Before operation, some time it is required to reset all the flip-flops to zero. It is called “Clear”. Some time, it is required to set the flip-flops before hand. It is called preset. To do these, two extra inputs are there in every flip-flop called CLR and PR.

---

#### 5.4.1 Asynchronous Counter

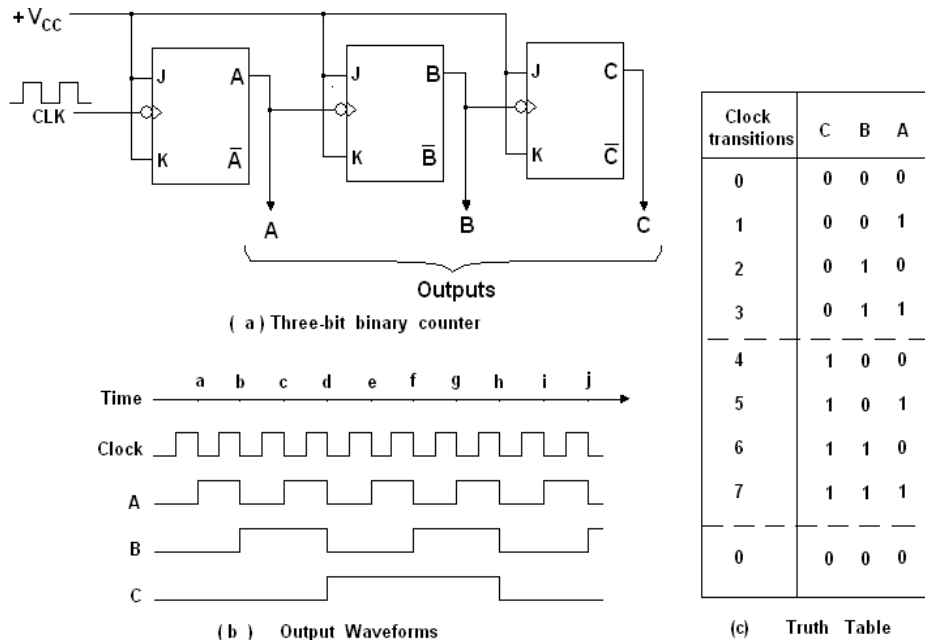
---

When the output of a flip-flop is used as the clock input for the next flip-flop it is called asynchronous counter.

Asynchronous counters are also called ripple counter because flip-flop transitions ripple through from one flip-flop to the next in sequence until all flip-flop reach a new state.

A binary ripple counter can be constructed by using clocked JK flip-flop. Fig 5.9(a) shows three MS JK flip-flops connected in series. The clock drives flip-flop A. The output of A drives B and the output of B drives C. J and K inputs of all the flip-flops are connected to positive to make them equal to 1. Under this condition each flip-flop will change state (toggle) with a negative transition at its clock point.

In the counter shown in Fig 5.9 (a), the flip-flop A changes its state at the negative edges of the clock pulses. Its output is applied to the B flip-flop as its clock input.



**Fig. 5.9 : Three Bit Binary Counter**

The output of B flip-flop toggles at the negative edges of the output of A flip-flop. Similarly output of B flip-flop is used as clock input to the C flip-flop and therefore C toggles at the negative edges of the output of B flip-flop. We can see that triggering pulses move through the flip-flops like a ripple in water.

The wave form of the ripple counter is shown in fig 5.9 ( b ). It shows the action of the counter as the clock runs. To understand the wave form let us assume that the counter is cleared before the operation. The A output is assumed the list significant bit(LSB) and C is the most-significant-bit (MSB). Hence at very beginning the contents of the counter is CBA=000.

Flip-flop A changes its state to 1 after the negative clock pulse transition. Thus at point 'a' on the time line , A goes high. At point 'b' it goes low, at 'c' it goes back to high and so on.

Now output of A acts as clock input of B. So, each time the output of A goes low, flip-flop B will toggles. Thus at point 'b' on the

time line, B goes high, at point 'd' it goes low, and toggles back high again at point 'f' and so on.

Since B acts as the clock input for C, each time the output of B goes low, the C flip-flop toggles. Thus C goes high at point 'd' on the time line, it goes back to low again at point 'h'.

We can see that the output wave form of A has half the frequency than the clock input wave. B has half the frequency than that of A and C has half the frequency than that of B.

We can further see that since the counter has 3 flip-flops cascaded together, it progresses through 000—001—010—011—100—101—110—111 as its CBA output. After CBA= 111, it starts the cycle again from CBA=000. One cycle from 000—111 takes 8 clock pulses, as it is evident from the wave form as well as from the truth table.

**MOD of a counter :** The number of input clock pulses that causes a counter to reset to its initial state is called the *modulus* of a counter. Therefore the modulus of a counter means the total number of distinct states including zero that a counter can store.

As for example the three stage counter shown in figure 5.9 has modulus 8 which can be seen from the wave diagram and the truth table. The modulus is called MOD in short.

When the modulus of a counter is equal to  $2^n$ , where n is the number of stages or flip-flops used, then the counter is called a natural counter. Hence a three stage counter has  $MOD = 2^3 = 8$ , four stage counter has  $MOD = 2^4 = 16$  etc and therefore these are natural counters.

A counter can be forcefully reset to its initial state before it completes its natural count by applying appropriate feedback. These type of counters have modulus lesser than its natural count. As such we can have MOD 6, MOD 7, MOD 10 counters etc.

---

### 5.4.2 Synchronous Counter

---

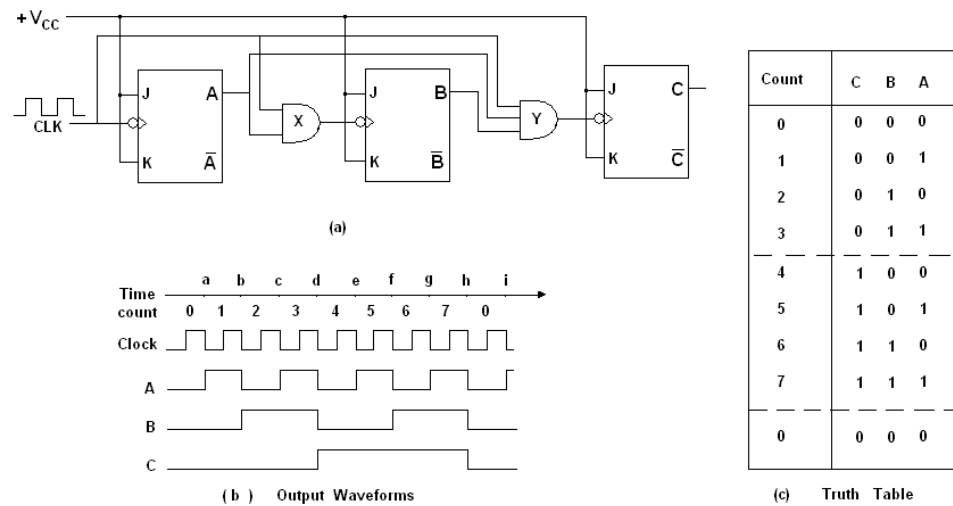
An asynchronous counter or ripple counter has limitation in its operating frequency. Each flip-flop has a delay time which is additive in asynchronous counter.

In synchronous counter the delay of asynchronous counter is overcome by the use of simultaneous applications of clock pulse to all the flip-flops. Hence in synchronous counter, the common clock pulse triggers all the flip-flops simultaneously and therefore the individual delay of flip-flop does not add together. This feature increase the speed of synchronous counter. The clock pulse applied can be counted by the output of the counter.

To build a synchronous counter, flip-flops and some additional logic gates are required. Fig 5.10 shows a three stage synchronous or parallel binary counter along with its output wave forms and truth table. Here the J and K inputs of each flip-flop is kept high and therefore the flip-flops toggle at the negative clock transition at its clock input. From figure we can see that the output of A is ANDed with CLK to drive the 2<sup>nd</sup> flip-flop and the outputs of A, B are ANDed with CLK to drive the third flip-flop. This logic configuration is often referred to as “steering logic” since the clock pulses are steered to each individual flip-flop.

In the figure , the clock pulse is directly applied to the first flip-flop. Its J and K are both high, so the first flip-flop toggles state at the negative transition of the input clock pulses. This can be seen at points a,b,c,d,e,f,g,h,i on the time line.

The AND gate X is enabled when A is high, and it allows a clock pulse to reach the 2<sup>nd</sup> flip-flop. So the 2<sup>nd</sup> flip-flop toggles with every other negative clock transition at points b, d, f and h on the time line.



**Fig. 5.10 : Parallel Binary Counter**

The AND gate Y is enabled only when both A and B are high and it transmits the clock pulses to the clock input of the 3<sup>rd</sup> flip-flop. The 3<sup>rd</sup> flip-flop toggles state with every fourth negative clock transition at d and h on the time line.

The wave form and the truth table shows that the synchronous counter progresses upward in a natural binary sequence from 000 to 111. The total count from 000 to 111 is 8 and hence this counter can also be called MOD-8 counter, in count up mode.



### CHECK YOUR PROGRESS

Q.5. State *true or false*

- Counters are non sequential digital circuits.
- Asynchronous counters are fast in operation than synchronous counters.
- The total number natural progression of a counter is called MODULUS.
- Counters can be used to build digital clock.

---

## 5.5 REGISTER

---

A register is a set flip-flops used to store binary data. The number to be stored is entered or shifted into the register and also taken out or shifted out as per necessity. Hence registers are also known as shift register.

Registers are used to store data temporarily. Registers can be used to perform some important arithmetic operations like complementation, multiplication, division etc. It can be connected to form counters, to convert serial data to parallel and parallel to serial data.

Types of registers: According to shifting of binary number different types of registers are:

- Serial In—Serial Out(SISO)
- Serial In –Parallel Out (SIPO)
- Parallel In –Serial Out(PISO)
- Parallel In –Parallel Out(PIPO)

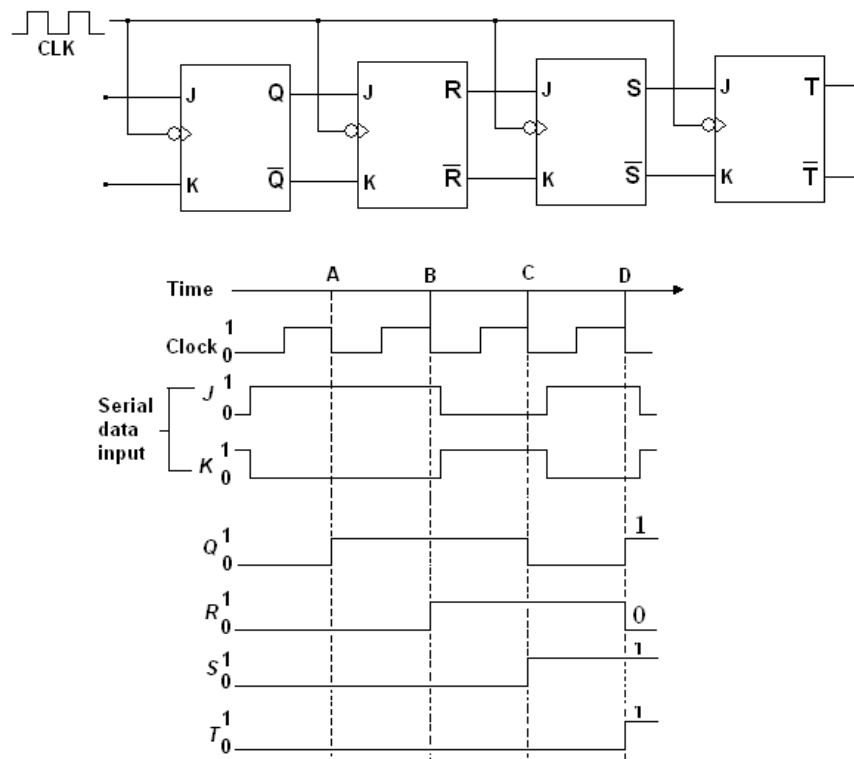
---

### 5.5.1 Serial In – Serial Out Register

---

Fig 5.11 shows a typical 4 bit Serial In – Serial Out register using flip-flops. Here the content of the register is named as QRST. Let us consider that all flip-flops are initially reset. Hence at the beginning QRST = 0000. Let us consider a binary number 1011 we want to store in the SISO register.

**At time A :** A 1 is applied at the J input and 0 at the K input of the first flip-flop. At the negative edge of the CLK pulse, this 1 is shifted into Q. The O of Q is shifted into R, O of R is shifted into S and O of S is shifted into T. The output of the register just after time A is QRST = 1000.



**Fig. 5.11 : 5 Bit Serial In – Serial Out Shift Register**

(During the input/output operation of the register, whatever we give to the J input, we must give complement of it to the K input.)

**At time B :** Another 1 and 0 is applied in the J & K input of the first flip-flop. So at the negative CLK edge , the 1 is shifted to Q. The 1 of Q is shifted in R., 0 of R is shifted in S, 0 of S is shifted into T. So, at the end of time B the output of all the flip-flops are QRST=1100.

**At time C :** A 0 (zero) is applied in the J input and 1 at the K input of the 1<sup>st</sup> first flip-flop. At the negative CLK edge , the 0 shifts to Q. The 1 of Q shifts into R., 1 of R shifts into S, 0 of S shifts into T. Hence the output becomes QRST=0110.

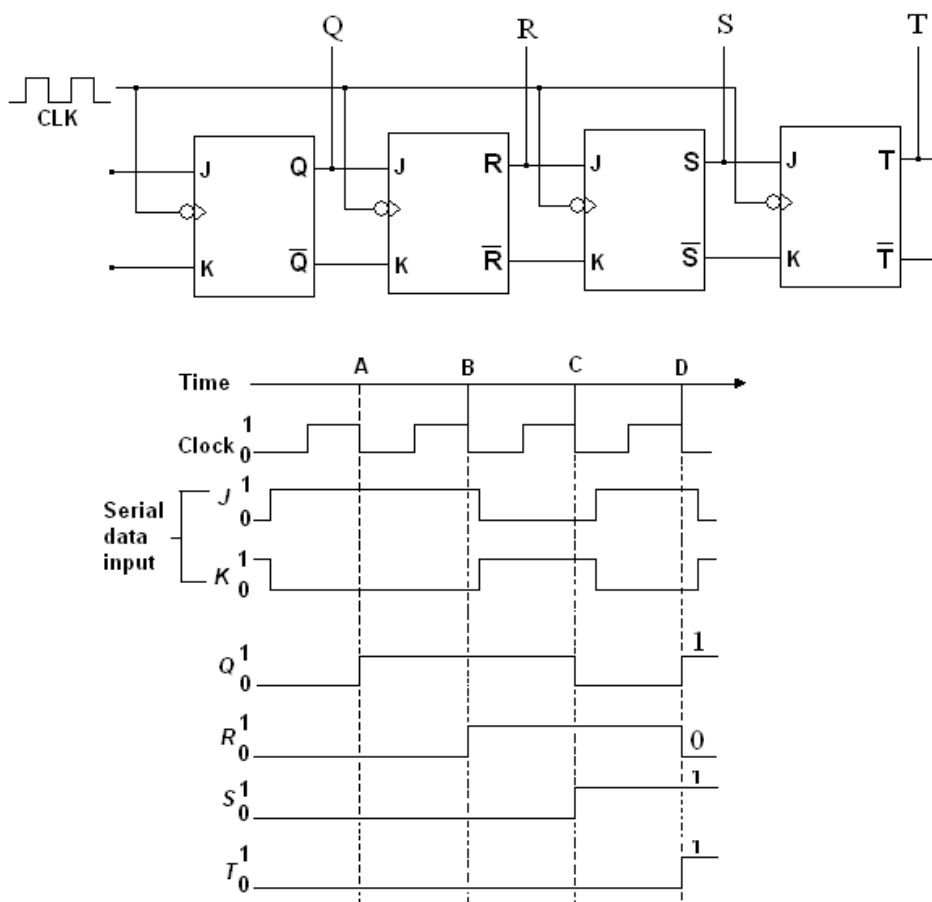
**At time D :** 1 is applied to J input and 0 at the K input of the first flip-flop. So after the negative clock pulse the 1 shifts into Q. The previous 0 of Q shifts into R, the 1 of R shifts into S, the 1 of S shifts into T. Hence at the end of time D, the registers contains QRST=1011.



In the above steps, using 4 CLK pulses, we have shifted a 4 bit binary number 1011 in the register in a serial fashion.

To take out this binary number serially, we need another 4 CLK pulses and four 0 inputs into J pin and four 1 into K pin of the first flip-flop. The binary number leave the register serially through the T pin of the last flip-flop.

### 5.5.2 Serial In—Parallel Out (SIPO)



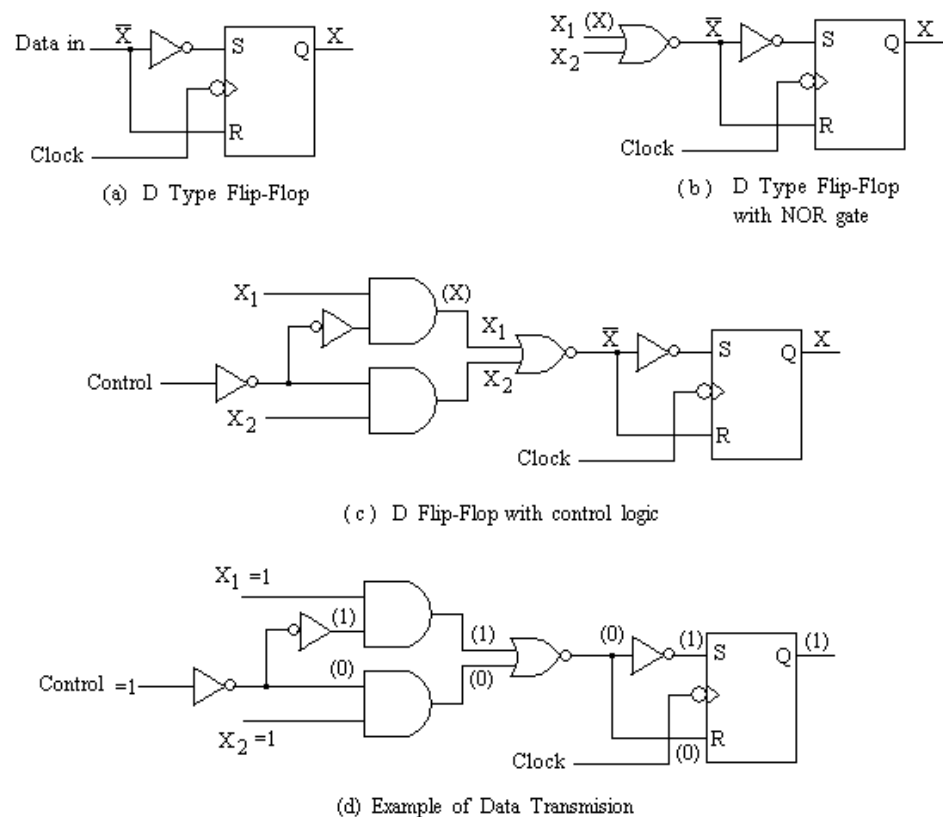
**Fig. 5.12 : 4 Bit Serial In – Parallel Out Shift Register**

In this type of shifts register, data entered serially into the register and once data entry is completed it can be taken out parallelly. To take the data parallelly, it is simply required to have the output of each flip-flop to an output pin. All other constructional features are same as Serial In—Serial Out (SISO) register.

The shifting of data into SIPO is same as SISO registers. In the SIPO of Fig 5.12 , a binary number say, 1011 would be shifted just like the manner as described in the previous section. It would take 4 CLK pulses to complete the shifting. As soon as shifting is completed, the stored binary number becomes available in the output pins QRST. SIPO register is useful to convert serial data into parallel data.

### 5.5.3 Parallel In-Serial Out Register

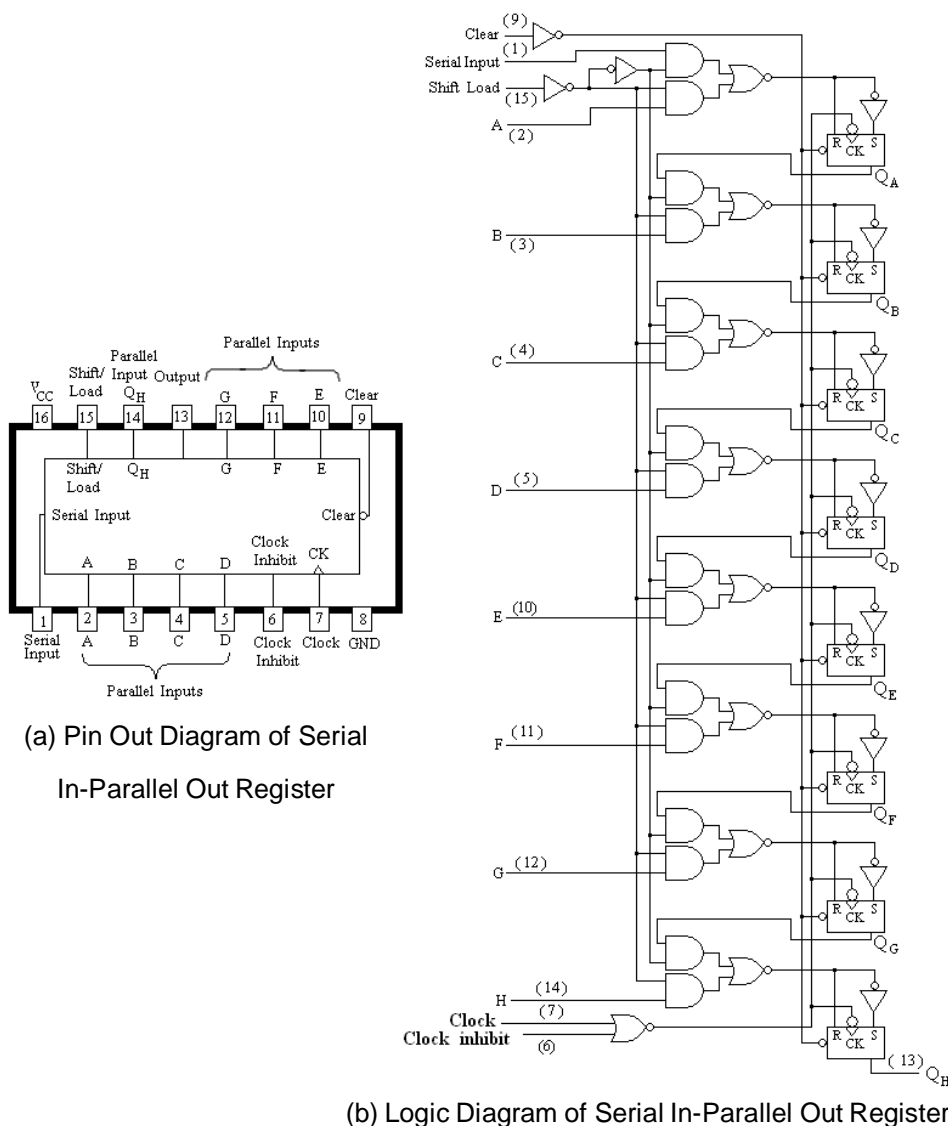
PISO registers takes data parallelly and shifts data serially. Commercially available TTL IC for PISO is 54/74166. To understand the functional block diagram of 54/74166 we should first understand the following :



**Fig. 5.13 : Building Block of Parallel In- Serial Out Register**

Fig 5.13 (a) is a clocked RS flip-flop, which is converted to D flip-flop by a NOT gate. The output of the flip-flop is 1 if Data IN ( $\overline{X}$ ) is 0. Next add a NOR gate as in Fig 5.13(b). Here, if  $X_2$  is at ground level,  $X_1$  will be inverted by the NOR gate. As for example, if  $X_1 = 1$ , then output of the NOR gate will be  $=0$ , thereby a 1 will be clocked into the flip-flop. This NOR gate allows entering data from two sources, either from  $X_1$  or  $X_2$ . To shift into the flip-flop,  $X_1$  is kept at ground level and to shift into the flip-flop,  $X_2$  is kept at ground level.

Now in Fig 5.13 (c) two AND gates and two NOT gates are added. These will allow the selection of data or data  $\overline{X}$ . If the control



**Fig. 5.14 : Circuit of 54/74166**

line is high, the upper AND gate is enabled and the lower AND gate is disabled. Thus the data will enter at the upper leg of the NOR gate and at the same time the lower leg of the NOR gate is kept at ground. Opposite to this, if the control is low, the upper AND gate is disabled and the lower AND gate is enabled. So will appear at the lower leg of the NOR gate and during this time the upper leg of the NOR leg gate is kept at ground level.

If we study the Fig 5.14 of PISO we see that circuit of Fig 5.13 ( c ) is repeated 8 times to form the 54/74166 shift register. These 8 circuits are connected in such a style that it allows two operations: (1) The parallel data entry and (2) shifting of data serially through the flip-flop from  $Q_A$  toward  $Q_B$

If Fig 5.14 the  $X_2$  input of Fig 5.13( c ) is taken out from each flip-flop to form 8 inputs named as ABCDEFGH to enter 8 bit data parallelly to the register. The control is named here as SHIFT/LOAD which is kept low to load 8 bit data into the flip-flops with a single clock pulse parallelly. If the SHIFT/LOAD is kept high it will enable the upper AND gate for each flip-flop. If any input is given to this upper AND gate then a clock pulse will shift a data bit from one flip-flop to the next flip-flop. That means data will be shifted serially.

---

#### 5.5.4 Parallel In –Parallel Out Register (PIPO)

---

The register of Fig 5.14 can be converted to PIPO register simply by adding an output line from each flip-flop.

The 54/74198 is an 8 bits such PIPO and 54/7495A is a 4 bit PIPO register. Here the basic circuit is same as Fig 5.13(c). The parallel data outputs are simply taken out from the Q sides of each flip-flop. In Fig 5.15 the internal structure of 54/7495A is shown.

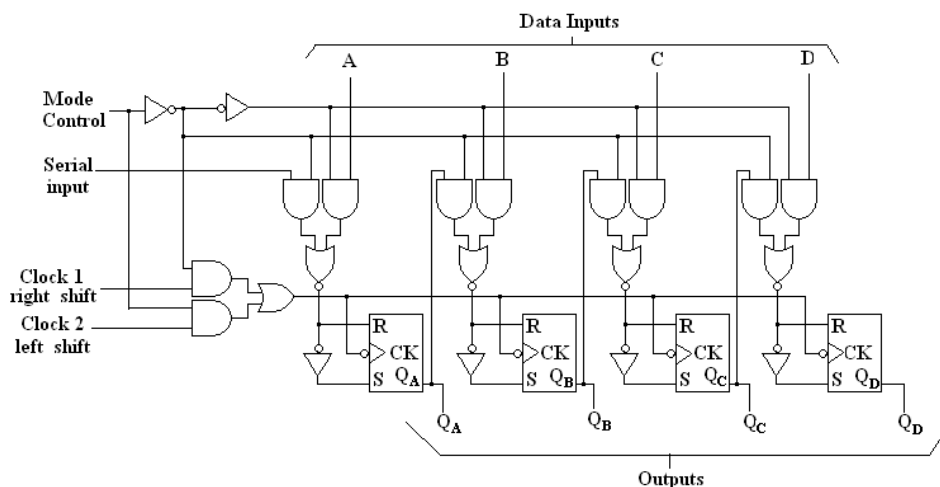
When the MODE CONTROL line is high, the data bits ABCD will be loaded into the register parallelly at the negative clock pulse. At the same time the output is available at  $Q_A Q_B Q_C Q_D$ . When the MODE CONTROL is low, then the left AND gate of the NOR gate is

enabled. Under this situation, data can be entered to the register serially through SERIAL INPUT. In each negative transition, a data bit shifted serially from  $Q_A$  to  $Q_B$ , from  $Q_B$  to  $Q_C$  and so on. This operation is called right-shift operation.


With a little modification of the connection, the same circuit can be used for shift-left operation. To operate in shift-left mode, the input data is to be entered through D input pin. It is also necessary to connect  $Q_D$  to C,  $Q_C$  to B,  $Q_B$  to A. MODE CONTROL line is high to enter data through the D input pin and each stored data bits of flip-flops will be shifted to left flip-flop on each negative clock transition. This is serial data and left shift operation.

Two clock inputs—clock1 and clock2 is used here to perform shift right and shift left operation..

Hence 54/7495A can be used as Parallel In –Parallel Out shift register as well as shift right and shift left register.



**Fig. 5.15 : Parallel In- Parallel Out Shift Register**



### CHECK YOUR PROGRESS

Q.6. Shift registers are

(a) basically a sequential circuit
(b) a combinational circuit

(c) permanent memory
(d) none of these

Q.7. In SIPO

- (a) data enters parallelly and leaves serially
- (b) data enters serially and leaves serially
- (c) data enters serially and leaves parallelly
- (d) data enters parallelly and leaves parallelly

---

## 5.6 LET US SUM UP

---

- Digital circuits are of two categories - combinational and sequential
- A sequential circuit's output depends on past output and present inputs.
- A flip-flop is basically a single cell of memory which can store either 1 or 0.
- Sequential circuits use flip-flop as their building block.
- There are many types of flip-flop viz RS, D, JK, MS flip-flop.
- A counter is a sequential circuit that can count square waves given as clock input. There are two types of counters- asynchronous and synchronous counter.
- Shift registers are also sequential circuit which are used to store binary bits. They are of four different types - Serial In- Serial Out, Serial In- Parallel Out, Parallel In- Parallel Out and Parallel In- Serial Out register.



---

## 5.7 FURTHER READING

---

- Mano, M. M., *Digital Logic and Computer Design*, PHI
- Mano, M. M., *Computer System Architecture*, PHI
- Malvino, Albert Paul & Leach, Donald P., *Digital Principles and Applications*, Mcgraw-Hill International
- Lee, Samuel C, *Digital Circuits and Logic Design*, PHI
- Theodore F. Bogart Jr., *Introduction to Digital Circuits*, Macmillan.
- Talukdar, Dr. Pranhari, *Digital Techniques*, N. L. Publications.



---

## 5.8 ANSWERS TO CHECK YOUR PROGRESS

---

**Ans. to Q. No. 1 :** (c)

**Ans. to Q. No. 1 :** (b)

**Ans. to Q. No. 1 :** (c)

**Ans. to Q. No. 1 :** (c)

**Ans. to Q. No. 1 :** (a) False, (b) False, (c) True, (d) True

**Ans. to Q. No. 1 :** (a)

**Ans. to Q. No. 1 :** (c)



---

## 5.9 MODEL QUESTIONS

---

- Q.1. Distinguish between combinational circuit and sequential circuit.
- Q.2. Define flip-flop. Draw an RS flip-flop using NOR gate and explain its operating principle with truth table.
- Q.3. What modification is done in RS flip-flop to construct D flip-flop? What is achieved after the modification ?
- Q.4. Explain the operation of MS flip-flop with its symbol and truth table.
- Q.5. What is the advantage of JK flip-flop over an RS flip-flop ? Write the working principle of a JK flip-flop.
- Q.6. Why a square wave clock pulse is converted to a narrow spike ? How it is done ?
- Q.7. What are the differences between asynchronous and synchronous counter? Draw a MOD-8 counter and explain its working principle.
- q.8. What do you mean by modulus of a counter ? What is a forced counter ?
- Q.9. What is called racing ? To get rid of racing what techniques are used ?
- Q.10. Draw logic diagram with output wave form of a 4-bit Serial In-Parallel Out shift register for an input of 1101. Explain its operation.

---

## UNIT 6 : MEMORY ORGANISATION

---

### UNIT STRUCTURE

- 6.1 Learning Objectives
- 6.2 Introduction
- 6.3 Random Access Memory
- 6.4 Types of RAM
  - 6.4.1 Static RAM
  - 6.4.2 DRAM
- 6.5 Organization of RAM
  - 6.5.1 2D organization
  - 6.5.2 3D organization
- 6.6 ROM
- 6.7 Types of ROM
- 6.8 Organization of simple ROM cell
- 6.9 Let Us Sum up
- 6.10 Answer to Check Your Progress
- 6.11 Further Reading
- 6.12 Model Questions

---

### 6.1 LEARNING OBJECTIVES

---

After going through this unit you will be able to

- define Random Access Memory
- define and describe the structure of RAM and ROM
- define their types and organization
- distinguish 2D and 3D organization of RAM

---

### 6.2 INTRODUCTION

---

In all the other units of this block we discussed various circuits and different types of logic gates. In this unit, we are going to discuss the internal organization of storage device of computer system where these circuits

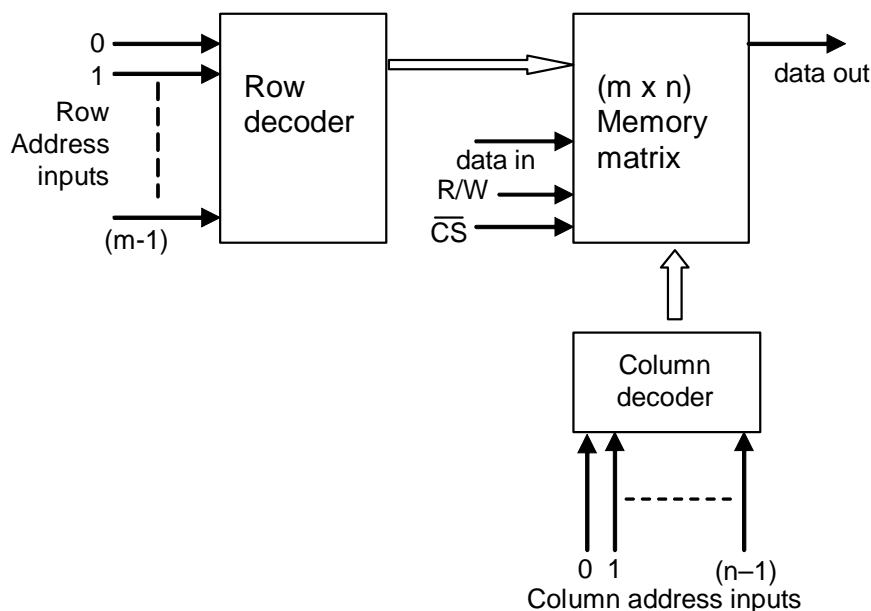


are associated to transfer data and information time to time. This is the memory unit, which is a collection of storage cells and circuits. A computer system uses varieties of storage devices for its different operations and accordingly the main memory is divided into two main categories: primary and secondary memory. Here we will discuss mainly about the primary memory which is accessed directly by the processor. It is mainly based on integrated circuits.

We will also discuss various types of primary memory, their classification and use with their organizational structures.

### 6.3 RANDOM ACCESS MEMORY (RAM)

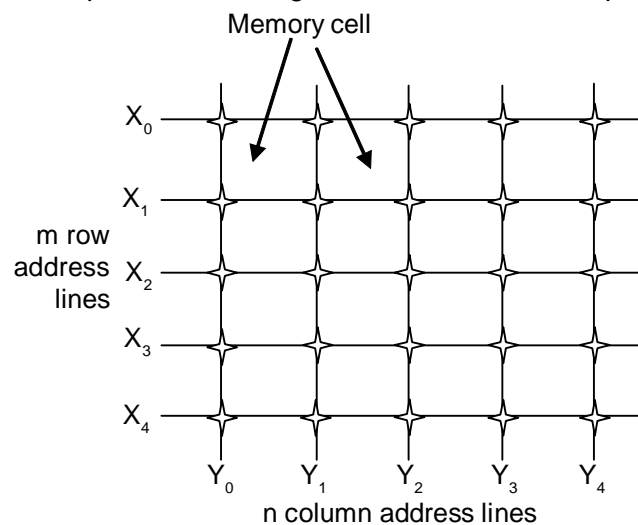
Each storage element of memory is directly or randomly accessible and can be examined and modified without affecting other cells and hence it is called random access memory. It can be both written into and read from. RAM gives a temporary place in computer to process electronic data. So RAM chip can store information only as long as computer has electronic power. It means that the data will be lost from RAM as and when the computer is shut off. So RAM is volatile in nature. Following figure represents the block diagram of typical RAM.



**Fig. 6.1 : Block diagram of a RAM chip**

The mode of access of a memory system is determined by the type of components it used. In a RAM, the word locations may be thought of as being separated in space, with each word occupying one particular location. The *access time* of a memory is the time required to select a word and either read or write it. In RAM, the access time is always the same regardless of the particular location of the word.

A semiconductor RAM is arranged in the form of a matrix of “m” words and each word is “n” bits wide. The intersections of rows and columns of such matrix are called *memory elements* or *cells*. Each cell has unique address representation. Figure shows the matrix representation of a RAM.

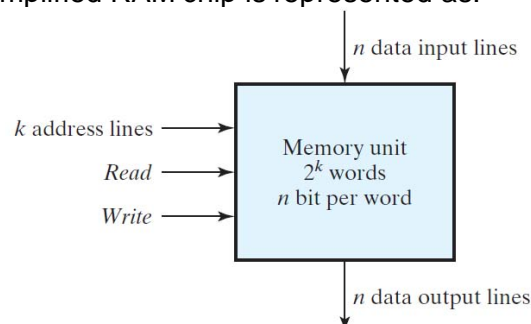


**Fig. 6.2 : RAM memory matrix representation**

The address and data stored in a specific location of RAM are binary in form. Data are typically organized into words and each word has n bits. For k bit address there will be  $2^k$  memory locations.

**k-bit address  $\rightarrow 2^k$  memory locations**

The simplified RAM chip is represented as:



**Fig. 6.3 : Simplified RAM chip**

## 6.2 TYPES OF RAM

Here in this section we are going to discuss two important types of RAM: Static RAM and Dynamic RAM. They are categorized according to their operating mode.

### 6.4.1 Static RAM

It is also called SRAM. It is one type of semiconductor memory that consist internal flip-flops which store binary information. The name static indicates that it does not need to be periodically refreshed. It is also volatile in nature as data is eventually lost when the memory is not powered.

**Static RAM cell and its structure :** Basically each bit in an SRAM is stored on four transistors that form two cross-coupled inverters. This storage cell has two stable states: 0 and 1. Two additional access transistors are used to control the read and write access operations. So, all total six transistors are required for a single SRAM memory cell. The Write and Read operations are performed by executing a sequence of actions that are controlled by the outside circuit.

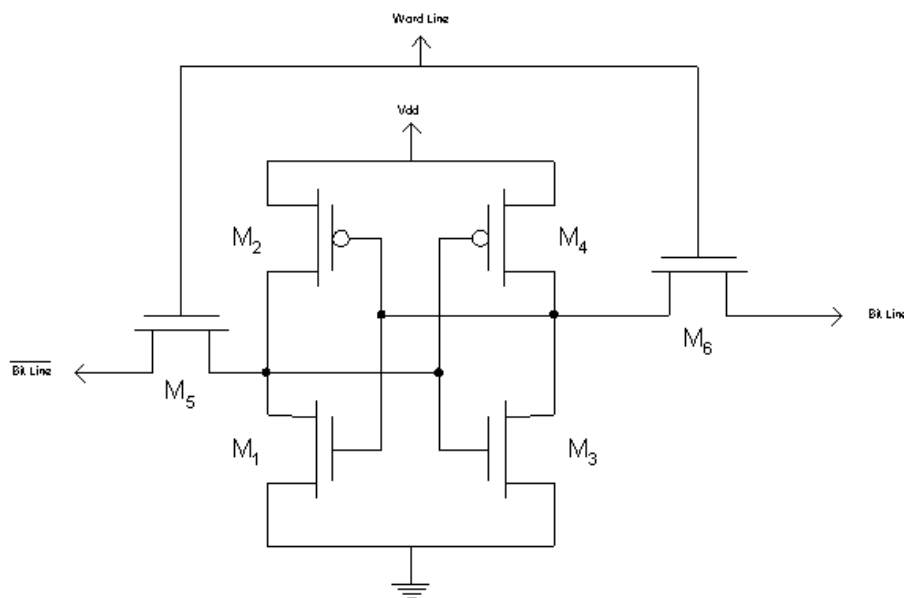


Fig. 6.4 : 6-transistor SRAM Memory Cell

Cell is accessed through the word line. It controls two access transistors  $M_5$  and  $M_6$ , to see whether the cell is connected to the bit lines (BL and BL in figure). They are used to transfer data for both read and write operations. But it is not strictly necessary to have two bit lines.

A Read operation is initiated by pre-charging both bit lines to logic 1. During read access inverters of SRAM are perform high and low operation. This improves SRAM bandwidth as compared to DRAMs. A Write operation is performed by first charging the Bit lines with values that are desired to be stored in the memory cell. Setting the Word Line high performs the actual write operation and the new data is latched into the circuit.

The size of an SRAM with  $m$  address lines and  $n$  data lines is  $2^m$  words, or  $2^m \times n$  bits.

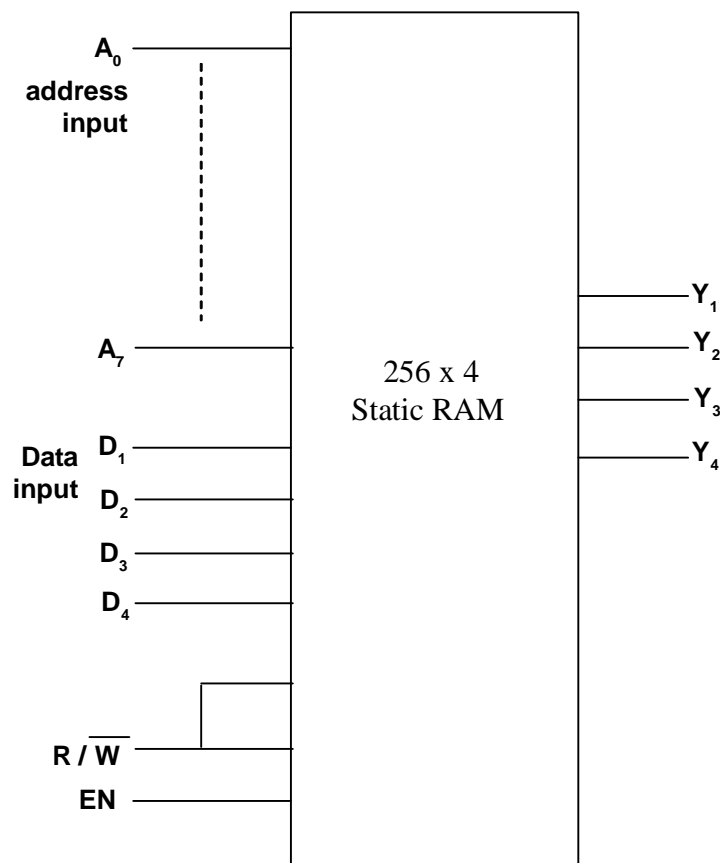


Fig. 6.5 : Logic diagram for 256 x 4 static RAM

**SRAM operation :** An SRAM cell has three different states and they perform differently as follows:

- a) **Standby states :** If the word line is not asserted, the access transistors  $M_5$  and  $M_6$  disconnect the cell from the bit lines. The two cross coupled inverters formed by  $M_1 - M_4$  can continue support each other as long as they are connected to the supply.
- b) **Reading states :** First of all let us assume that content of memory is initially 1 and stored at Q. As we know that the read cycle starts with the pre charging bit line values as 1, so it assert the word line which enables both access transistors. In the next step values of Q transferred to bit lines by leaving BL at its pre charges values and it discharge BL through  $M_1$  to  $M_5$  with the logic 0. On the BL side, the transistors  $M_4$  and  $M_6$  pull the bit line towards VDD with logic 1. The same will happen as opposite if the memory value is 0.
- c) **Writing states :** The write cycle begins by applying the values to the bit lines. If we wish to write 0, we would have to apply 0 to the bit lines. A 1 is written by applying the values of the bit lines. Word Line is then asserted and the value that is to be stored is latched in.

**Characteristics of SRAM :**

- a) More expensive, but faster and significantly less power is used than DRAM.
- b) Easier to control and generally more truly random access.
- c) Cannot be used in high-capacity, low-cost applications, such as the main memory in PCs.
- d) SRAM are used in many industrial and scientific subsystems. It is embedded in practically all modern appliances, toys etc.

**Types of SRAM :**

- a) **Non-volatile SRAM :** This is a special type of SRAM which can retain their data even the power supply is lost. They are used in a wide range of situations where the preservation of data is critical like networking, aerospace and medical.

- b) **Asynchronous SRAM** : This is fast access memory and is used in main memory. They are available from 4 Kb to 32 Mb.
- c) **By transistor type** : As according to the transistor type we can divide SRAM again in two categories like:
  - **Bipolar junction transistor** – This is used in TTL and ECL. They are very fast but consume a lot of power.
  - **MOSFET** – This is used in CMOS. They have low power and very common today.
- d) **By Function**: By function again SRAM can divide into two categories as:
  - **Asynchronous** – These are independent of clock frequency. Here data in and data out are controlled by address transition.
  - **Synchronous** – Here all timings are initiated by the clock edges. Address, data in and other control signals are associated with the clock signals.
- e) **By Feature** : By feature we can divide SRAM into few separate categories like Zero Bus Turnaround (ZBT), syncBurst SRAM, DDR SRAM etc.

**Usage of SRAM** : Other parts of the computer, such as cache memories and data buffers in hard disks, normally use static RAM (SRAM).

---

#### 6.4.2 Dynamic RAM

---

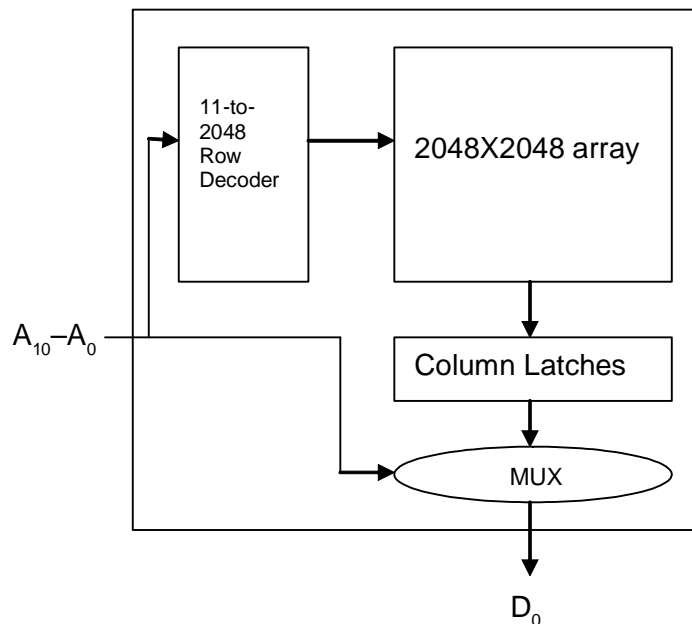
Dynamic RAM (DRAM) is the most common memory for personal computers and workstations. The main memory in PCs is DRAM. They store each bit of data in a separate capacitor within an integrated circuit. The capacitor can be either charged or discharged. These two states are represented by two values of a bit 0 and 1. Since capacitor leak charge, the information eventually fades unless the capacitor charge is refreshed periodically. Because of this refresh requirement, it is a dynamic memory.

The main advantage of DRAM is its structural simplicity. In DRAM only one transistor and a capacitor are required per bit, compared

to six transistors in SRAM. They are extremely small. So, hundreds of billions can fit on a single memory chip. This allows DRAM to reach very high densities. It is volatile in nature, since it loses its data quickly when power is removed.

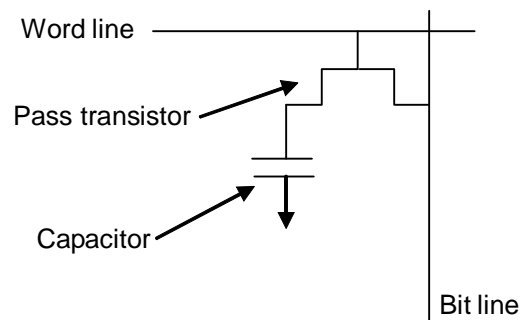
The first DRAM with multiplexed row and column address lines was the Mostek MK4096 (4096x1) designed by Robert Proebsting and introduced in 1973. DRAM is usually arranged in a square array of one capacitor and transistor per data bit storage cell. In the structure, there are two level decoding. They are:

- Row access – goes into decoder to select row
- Column access – controls the MUX.



**Fig. 6.6 : Basic DRAM structure**

**DRAM memory cell :** A typical DRAM cell consists of a single MOSFET and a capacitor. An array of such cells is used practically in a DRAM cell. Here the transistor acts as a switch. For READ and WRITE operations also it uses MOSFETs.



**Fig. 6.7 : A dynamic MOS RAM cell**

The steps of reading and writing in a memory cell are :

### **Writing to DRAM cell**

#### **Steps**

- Assert word line (turn the pass transistor “on”)
- Write value to bit line
- If the value is 1, the capacitor will be charged
- If the value is 0, the capacitor will be discharged

### **Reading DRAM cell**

#### **Steps**

- Bit line charged 1/2 between low and high voltage
- Assert word line
- Value in capacitor is “read out” onto the bit line
- Bit line swings slightly to low or high
- Sense amp detects swing and indicates a 0 or 1
- B/C charge used in detection, rewrite after read

### **Features of DRAM**

- Very dense
- Slow compared to SRAM (5-10X slower)

**Usage :** For economic reasons, the large (main) memories found in personal computers, workstations, and non-handheld game-consoles (such as PlayStation and Xbox) normally consist of dynamic RAM.

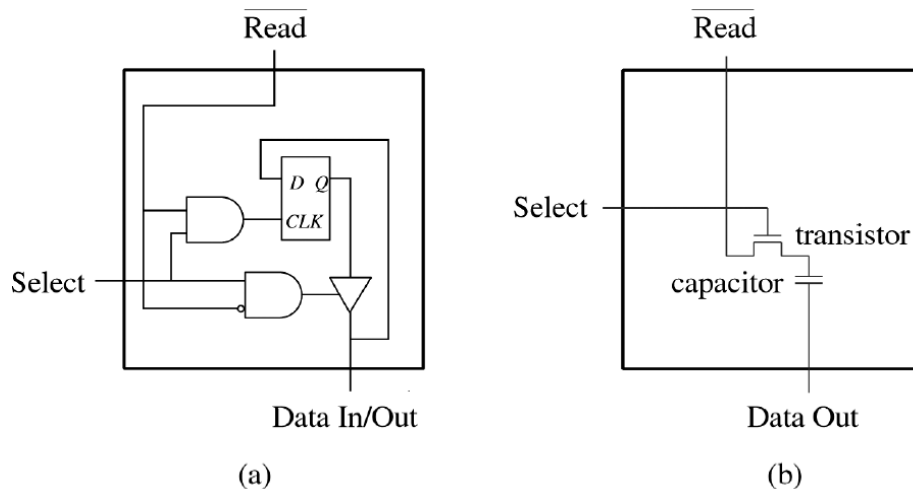
### **Comparisons between static and dynamic RAM**

- a) No refreshing in static RAM as it uses bistable latching circuitry to store each bit, but in DRAM, refreshed periodically.



- b) 6 to 8 MOS transistors are required to form one memory cell in SRAM, whereas in DRAM only 3 to 4 transistors are used to form one memory cell.
- c) Information stored in SRAM as voltage level in a flip flop, whereas in DRAM, information is stored as a charge in the gate to substrate capacitance.
- d) SRAM is more reliable, but 4 times expensive than DRAM
- e) An SRAM cell takes a lot more space on a chip than a DRAM.
- f) SRAM is used to create the CPU's speed sensitive cache, while DRAM forms the larger system RAM space.

The difference we can get from their functional behavior which is shown below:



**Fig. 6.8: (a) Static RAM cell and (b) dynamic RAM cell**



### CHECK YOUR PROGRESS

Q.1. Give classification of RAMs.

.....  
 .....

Q.2. Differentiate SRAM and DRAM.

.....  
 .....

Q.3. Fill in the blanks:

- a) RAM is \_\_\_\_\_ memory device.
- b) In personal computer \_\_\_\_\_ memory is used.
- c) In DRAM, if the value is 1, the capacitor is \_\_\_\_\_.
- d) A typical DRAM cell consists of a single \_\_\_\_\_ and a capacitor.
- e) In SRAM \_\_\_\_\_ transistors are used.
- f) \_\_\_\_\_ are more expensive
- g) For n bit address there will be \_\_\_\_\_ memory locations.
- h) The most common and low powered RAM is \_\_\_\_\_ used in CMOS.
- i) \_\_\_\_\_ is used to create the CPU's speed sensitive cache.
- j) The first DRAM with multiplexed row and column address lines was introduced in \_\_\_\_\_.

---

## 6.5 ORGANIZATION OF RAM

---

A RAM memory cell is organized in a number of different ways. A (m x n) memory has m-number of words, each word having n number of bits. In a RAM each memory location has a unique address. Depending on access mechanism, RAM is organized into different dimensions most commonly like 2D, 3D, 2.5D etc.

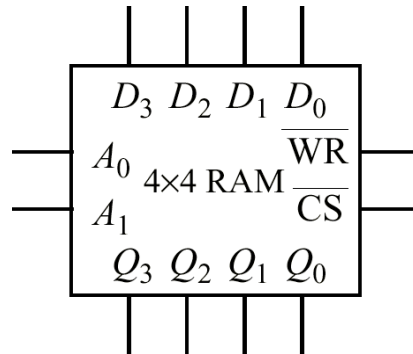
---

### 6.5.1 2D Organization

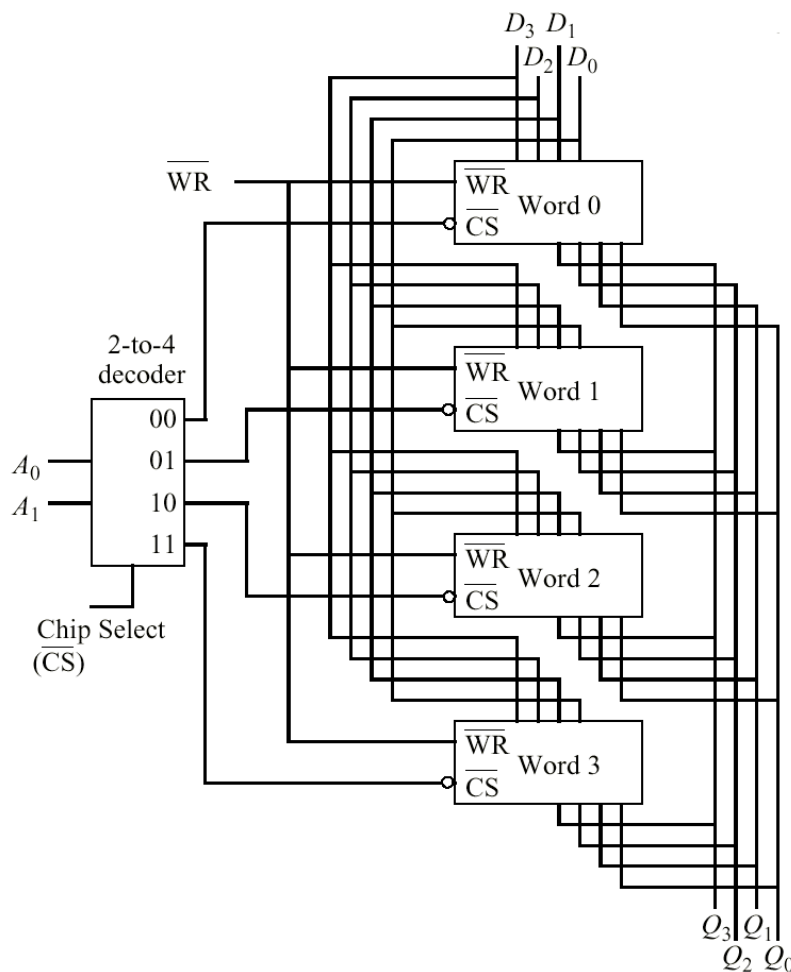
---

In this organization the memory on a chip is considered to be a list of words in which any word can be accessed randomly e.g., the memory of PCs used to have 16 bit words and normally in a chip it have 64KB memory = 32K words. The cells are organized as a 2-dimensional array with rows and columns. Each row refers to a word line. For n-bits per word, n-number of cells is interconnected

to a word line. Each column represents a bit line. The simplified representation of this organization is:



**Fig. 6.9 : Simplified 2D organization with 4X4 RAM chip**



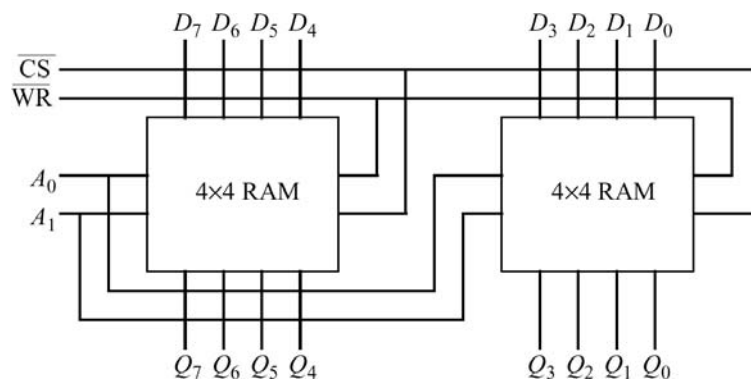
**Fig. 6.10 : A 4-word memory with 4-bit per word 2D organization**

In this array organization bit line is connected to the input and sense (or output) terminal of each cell in its respective column. Each decoded line of decoder drives a word line. A complete word

can be input or output from the memory simultaneously. In write operation, the address decoder selects the required word and the bit lines are activated for a value 0 or 1 according to the data line values. Thus completes the write operation. On read operation, the value of each bit line is passed through a sense amplifier and passed on the data lines. Thus enabling and completes the read operation. The word line identifies the word that has been selected for reading or writing. Usually ROMs and read mostly memories use 2D chip organization.

**Example 6.1 :** Design a 4-word by 8-bit RAM chip by connecting two 4 X 4 RAM chips.

**Solution :** The 4 x 8 RAM chip by using two 4 x 4 RAM chip is as follows :



**Fig. 6.11 : A 4 X 8 RAM chip**

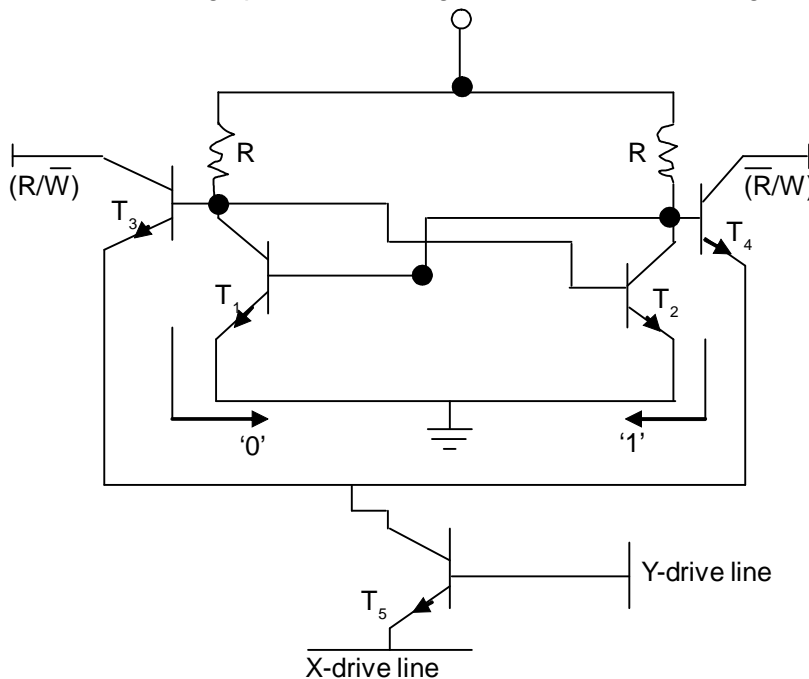
### 6.5.2 3D Organization

In this organization of memory, each cell needs three terminals:

- X drive lines for rows
- Y drive lines for columns
- A bit/sense pair

In comparison to 2D organization, 3D is more economical. The content of memory address register (MAR) has two parts having number of bits. Suppose they are x and y. For an n-bit word there are n-bit planes. For each bit plane, a sense amplifier is attached. Any cell in a bit plane is selected through X and Y drive lines. This

organization makes the circuit more complicated and also gives a low switching speed. Following is the 3 terminal 3D organization:



**Fig. 6.12 : 3 terminal semiconductor memory cell**



### CHECK YOUR PROGRESS

Q.4. Write TRUE or FALSE

- In a RAM each memory location has a unique address.
- In 2D organization of RAM, word cannot be accessed randomly.
- In 2D organization, each column represents a bit line.
- 2D organization is more economical than 3D organization of RAM.
- Each row refers to a word line in 2D organization.

## 6.6 ROM

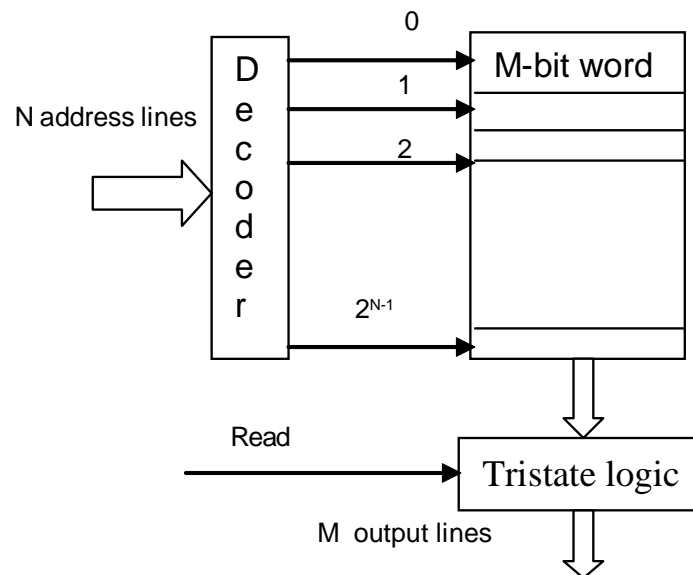
Read-only memory (ROM) is a class of storage device used in computer as well as in other electronic devices. Data stored in ROM cannot



**ROM:** A semiconductor memory where can store information permanently.

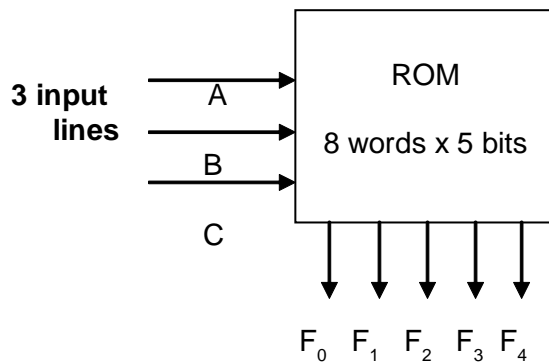
be modified, or can be modified only slowly or with difficulty. So, it is used mainly to distributed system. Its internal organization is similar to SRAM. It is used in computer systems for initialization because it does not lose storage value when power is removed. In other sense we can say that ROM is used to *boot* the operating system of the computer. It is programmed for particular purpose during the manufacturing time and user cannot alter its function.

ROM is a combinational logic circuit. Multiple single-bit functions embedded in a single ROM. It uses decoders to minimize the number of address lines. ROMs are effective at implementing truth tables. Any logic function can be implemented using ROMs. So, ROM can use to implement complex combinational circuits within one IC package or as permanent storage package for binary information that given by the designer.



**Fig. 6.13 : Block diagram of ROM**

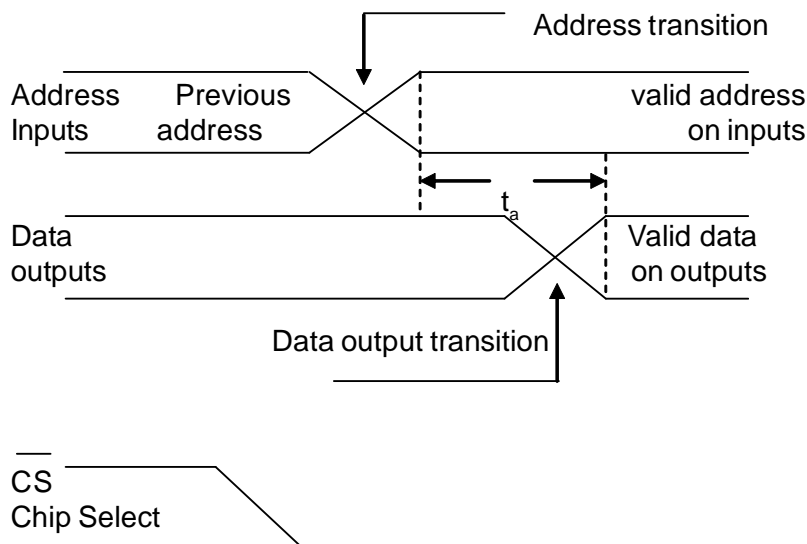
Here  $N$  is the address lines and  $M$  is the output lines. Each bit combination of the address variable is called an address and each bit combination that comes out from the output lines is a data word. An output word can be selected by a unique address. Since there are  $N$  input bits, ROM will form  $2^N$  words by  $M$  outputs. So, word available on the output lines at any instant depends on the address value applied to the input lines. For example, a ROM with 8 words by 5 bits is:



**Fig. 6.14 : 8 words X 5 bit ROM cell**

Since, here there are 3 input lines, the words will form by this ROM are  $2^3 = 8$  words.

**ROM access time :** Rom access time is the time from the application of a valid address code on the address inputs until the appearance of valid output data. It can also be measured from the activation of **chip select** to the occurrence of valid output data when a valid address is already on the inputs. Following figure shows the ROM access time.



**Fig. 6.15 : ROM Access Time**

**Example 6.2 :** Draw a 8 X 4 ROM

**Solution :** Since has to draw 8-word ROM, and  $2^3=8$ , so there will be 3 input lines.

Say, N = 3 input lines,

M = 4 output lines

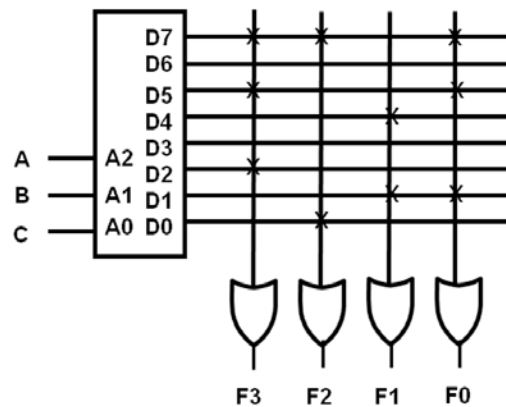


Fig. 6.16 : A 8 X 4 ROM

In the above ROM chip, the fixed “AND” array is a “decoder” with 3-inputs and 8-outputs that implementing minterms. The programmable “OR” array uses a single line to represent all inputs to an OR gate. An “X” in the array corresponds to attaching the minterm to the OR. In read mode for input  $(A_2, A_1, A_0) = 011$ , the output is  $(F_3, F_2, F_1, F_0) = 0011$ .

## 6.5 TYPES OF ROM

ROMs are classified according to how information are written or programmed into the memory storage locations. It is categorized as:

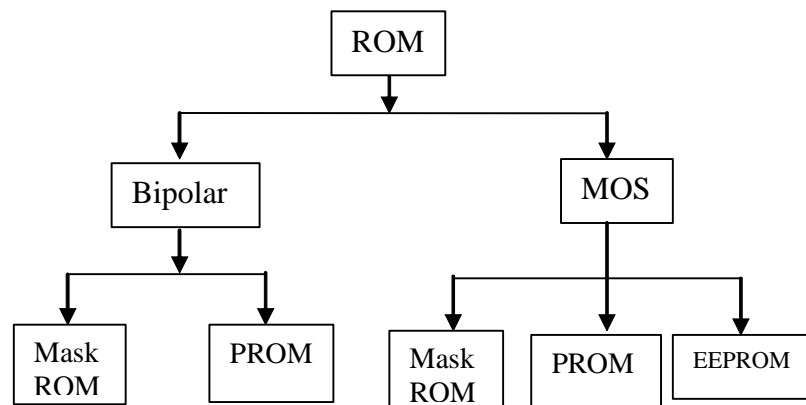


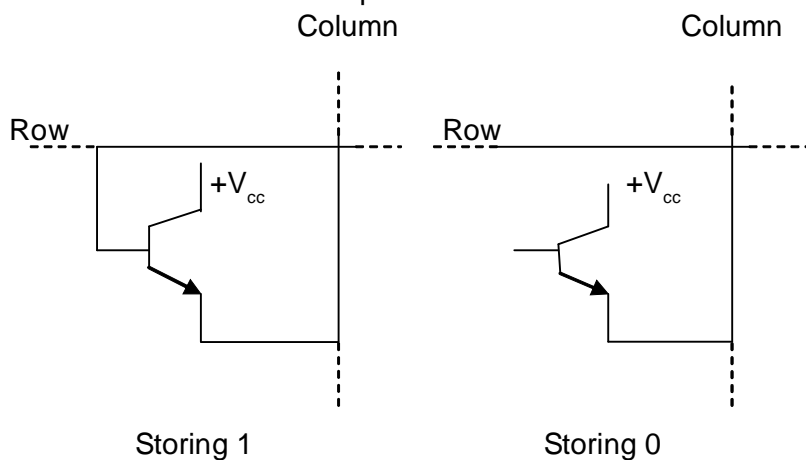
Fig. 6.17 : The ROM family

Bipolar ROM uses bipolar transistors within an integrated circuit. They can be programmed. On the other hand MOS ROM is constructed using MOSFETs. In a MOS cell, the silicon gate is completely insulated from the n-type substrate. The MOS can also be programmed. ROMs are programmed into two different ways: first is called *mask programming* and



is done by manufacturer during the last fabrication process of the unit; the second type is *programmable read only memory* (PROM).

**Mask ROM** : It is programmed at the time of manufacturing according to customer's requirements. Once the memory is programmed, it cannot be changed. Most IC ROMs utilize the presence or absence of a transistor connection at ROW or COLUMN junction to represent 1 or 0. The 1s and 0s are obtained by providing a mask in the last fabrication step. A photographic negative called a mask is used to control the electrical interconnections on the chip.

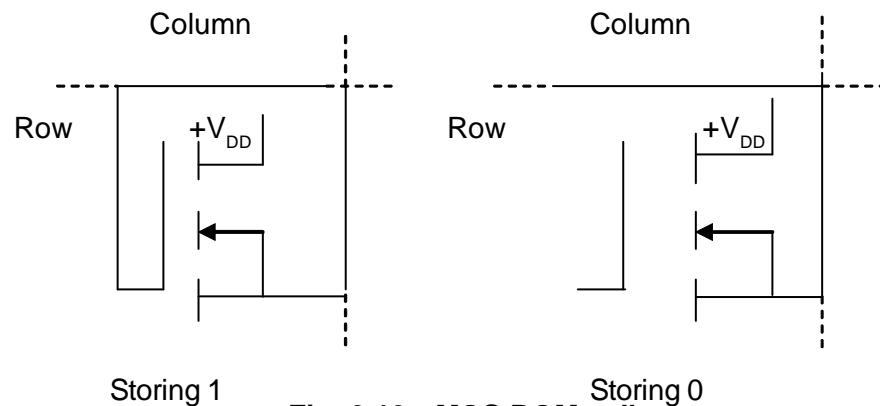


**Fig. 6.18 : Bipolar ROM cell**

In bipolar ROM cell, when Row line is taken high, all transistors with base connection to the Row line turn ON and connect the HIGH to the associated column lines. So, the presence of a connection from a Row line to the base of a transistor represents a '1' at that location. When there are no base connections at row or column junctions, the column lines remain LOW and represents as '0'.

In MOS ROM cell the presence or absence of a gate of MOSFET connection at junction stores 1 or 0 permanently. Manufacturer makes the corresponding mask for the path of bipolar or for MOS ROM cell to produce the 1s or 0s according to specified customer's truth table. This process is called **custom or mask programming**. Mask programmed ROM are economical only if large quantities of the same ROM configuration are to be manufactured.

The following figure illustrates the function of MOS ROM cell.



**Fig. 6.19 : MOS ROM cell**

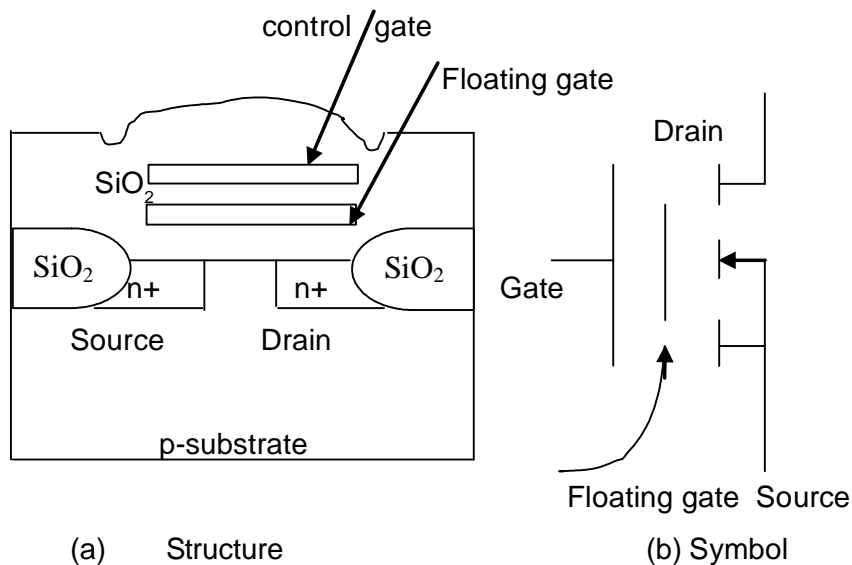
In MOS ROM cell the presence or absence of a gate of MOSFET connection at junction stores 1 or 0 permanently. Manufacturer makes the corresponding mask for the path of bipolar or for MOS ROM cell to produce the 1s or 0s according to specified customer's truth table. This process is called **custom or mask programming**. Mask programmed ROM are economical only if large quantities of the same ROM configuration are to be manufactured.

A major disadvantage of this type is the fact that it cannot be reprogrammed in the event of the design change or cannot do any modification of the stored program. Vendors are trying to overcome this problem by developing several program on this connection.

**Programmable Read-only Memory (PROM) :** In order to provide flexibility in some ROM applications, programmable ROM has been introduced. Blank PROM chips can be bought inexpensively and coded by anyone with a special tool called a **programmer**. But it cannot be reprogrammed. It is manufactured as a generalized integrated circuit with all matrix intersections and every intersection has a **fuse** connecting them. The higher voltage breaks the connection between the column and row by **burning** out the fuse and can thereby program the PROM according to the required truth table. This process is known as **burning the PROM**. Once a PROM is programmed, it cannot be changed and therefore it has to be done carefully and correctly in the first time itself. Hence, the fusing process is irreversible. PROMs are widely used in the control of electrical equipment such as washing machines and electric oven.

PROMs are available in both bipolar and MOS technologies. They have 4-bit or 8-bit output word formats with capacities ranging in excess of 2,50,000 bits.

**Erasable Programmable ROM (EPROM) :** A PROM device which can be erased and reprogrammed is called EPROM device. EPROM chips can be rewritten many times. Erasing an EPROM requires a special tool that emits a certain frequency of ultraviolet (UV) light. Once again we have a grid of columns and rows. In an EPROM, the cell at each intersection has two transistors. One of the transistors is known as the **floating gate** and the other as the **control gate**. To erase it, you must supply a level of energy strong enough to break through the negative electrons blocking the floating gate. In a standard EPROM, this is best accomplished with UV light at a frequency of 253.7. The EPROM must be very close to the eraser's light source, within an inch or two, to work properly.



**Fig. 6.20 : EPROM cell**

Here an additional floating gate is formed within the silicon dioxide (SiO<sub>2</sub>) layer. The floating gate is left encountered while the normal control gate is connected to the row decoder output of EPROM. The data bits are represented by the presence or absence of a stored charge. The initial values of un-programmed EPROM cells may be all 0s or all 1s.

#### **Disadvantages :**

- a) Changes in the selected memory locations cannot be made in the reprogramming. The entire memory should be erased before reprogramming.
- b) The process of reprogramming cannot take place with the IC in the circuit. The EPROM IC must be removed from the circuit and the stored program can be erased by exposing the memory cells to ultraviolet light through a 'window' on the IC package. This process takes about half an hour.

**Electrically Erasable PROM (EEPROM)** : It is also known as Electrically Alterable PROM (EAPROM). It removes the biggest drawbacks of EPROMs. The entire chip does not have to be completely erased to change a specific portion of it. It can be erased and programmed by the application of controlled electric pulses to the IC in the circuit, and thereby changes can be made in the selected memory locations without disturbing the correct data in other memory locations. Changing the contents does not require additional dedicated equipment. They are non-volatile like EPROM but do not require ultraviolet light to erase.

EEPROM is a rugged, low power semiconductor device and it occupies less space. It has the advantage of program flexibility, small size and semiconductor memory ruggedness. With EEPROM, the programs can be altered remotely, possibly by telephone. EEPROMs are changed 1 byte at a time.

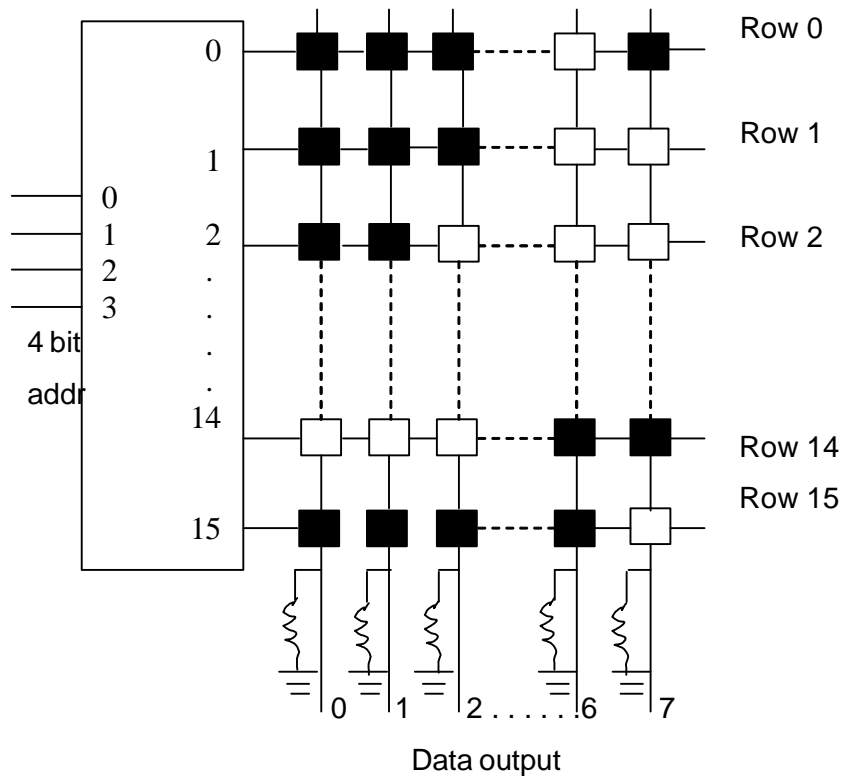
**Flash memory** : These memories are basically EEPROMs except that erasure occurs at the block level in order to speed up the write process. They work much faster than traditional EEPROMs because it writes data in chunks, usually 512 bytes in size, instead of 1 byte at a time. One of the most common uses of Flash memory is for the **basic input/output system** of computer. Advantages are that they are non-volatile in nature and densities are higher than both SRAM and DRAM. Disadvantages are that they are slow and data cell must be erased before writing data to it.

---

## 6.5 ORGANIZATION OF SIMPLE ROM CELL

---

A ROM is sometimes specified by the total number of bits it contains, which is  $2^n \times m$ . We already specified that a ROM can be organized by its address and data bits. Figure shows a 16 x 8 bit simplified ROM array.



**Fig. 6.21 : A 16 x 8-bit ROM array**

Here ROM is organized into 16 addresses, each of which stores 8 data bits. The total capacity of this ROM is 128 bits. The dark squares represent that ROM stored 1s by base connected transistor or gate-connected MOSFET and light squares represent 0s. When a 4 bit binary address is applied to the address inputs, the corresponding row line becomes HIGH. This HIGH is connected to the column line through transistors at each junction where 1 is stored. The column line stays LOW at each cell where 0 is stored because of the terminating resistor. The column lines form the data output. Thus, the eight data bits stored in the selected ROW appear on the output lines.

**CHECK YOUR PROGRESS**

Q.5. Fill in the blanks

- a) ROM is used to \_\_\_\_\_ the operating system of the computer.
- b) ROM is used mainly to \_\_\_\_\_.
- c) The internal organization of ROM is similar to \_\_\_\_\_.
- d) ROM uses \_\_\_\_\_ to minimize the number of address lines.
- e) \_\_\_\_\_ is done by manufacturer during the last fabrication process of the unit.
- f) Mask is used to control the \_\_\_\_\_ on the ROM chip.
- g) In Mask ROM cell, if column lines are LOW, it represents as \_\_\_\_\_.
- h) \_\_\_\_\_ are available in both bipolar and MOS technologies.
- i) In an EPROM, the cell at each intersection has \_\_\_\_\_ transistors.
- j) In EPROM, the program can remove for reprogramming is done by \_\_\_\_\_.

Q.6. Write TRUE or FALSE

- a) Two transistors used in EPROM are floating gate and control gate.
- b) ROM is a volatile memory.
- c) ROM access time is measured from the chip select activation to the occurrence of valid output data.
- d) Flash memory is used to speed up the read process of ROM chip.
- e) EEPROM is a low power semiconductor device and it occupies less memory space.

---

## 6.5 LET US SUM UP

---

- Data used in a program as well as instructions for executing program are stored in memory device. So, digital systems require memory facilities for temporary as well as for permanent storage of data to perform their functions.
- Based on the method of access, memory devices can be classified into different categories.
- In RAM any location can be accessed in a random manner and thus has equal access time for all memory locations.
- RAM chip can store information only as long as computer has electronic power.
- A semiconductor RAM is arranged in the form of a matrix of “m” words and each word is “n” bits wide and makes *cells*. Each cell has unique address representation.
- The address and data stored in a specific location of RAM are binary in form.
- An SRAM cell has three different states and they perform differently.
- DRAM use capacitance of a transistor as the storage device.
- Depending on access mechanism, RAM is organized into different dimensions most commonly like 2D, 3D, 2.5D etc.
- In comparison to 2D organization, 3D is more economical.
- In order to provide flexibility in some ROM applications, different programmable ROMs are introduced as PROM, EPROM and EEPROM.
- PROMs are widely used in the control of electrical equipment such as washing machine and electric oven.
- DRAM stores its binary information in the form of electric charges on capacitors.
- The bipolar ROMs can be subdivided into Mask ROMs and PROMs.
- Flash memories are basically EEPROMs except that erasure occurs at the block level in order to speed up the write process.



---

## 6.10 ANSWER TO CHECK YOUR PROGRESS

---

**Ans. to Q. No. 1 :** a

**Ans. to Q. No. 2 :** a

**Ans. to Q. No. 3 :** a) Volatile, b) DRAM, c) charged, d) MOSFET,  
e) 6 (six), f) SRAM, g)  $2^n$ , h) MOSFET , i) SRAM,  
j) 1973

**Ans. to Q. No. 4 :** a) T, b) F, c) T, d) F, e) T

**Ans. to Q. No. 5 :** a) Boot, b) distributed system, c) SRAM, d) Decoders,  
e) mask programming, f) electrical interconnections,  
g) 0 (zero), h) PROMs, i) two, j) UV light.

**Ans. to Q. No. 6 :** a) T, b) F, c) T, d) F, e) T



---

## 6.11 FURTHER READING

---

- “Digital Techniques” by Dr. Pranhari Talukdar, N.L. Publication.
- “Digital Design” by M. Morris Mano, PHI Publication.
- “Microprocessor and Peripherals” by S. P. Chowdhury and Sunetra Chowdhury, SCITECH.
- “Computer Fundamentals and C Programming” by Dr. Amiya Kumar Rath, Alok Kumar Jagdev and Santosh Kumar Swain, SCITECH.
- “Operating System” by P. Balakrishna Prasad, SCITECH.
- “Computer Fundamentals, Architecture and Organisation” by B. Ram, New Age International (P) Ltd., Publishers.



---

## 6.12 MODEL QUESTIONS

---

Q.1. What is RAM? Give an example.

Q.2. What is access time? How to find ROM access time in a computer?



- Q.3. Explain ROM family. How it is classified?
- Q.4. Discuss the applications of ROM.
- Q.5. Give two examples of non-volatile memory devices.
- Q.6. Give the classification of semiconductor RAMs.
- Q.7. What is the purpose of employing address multiplexing in a DRAM?
- Q.8. Differentiate SRAM with DRAM.
- Q.9. Describe the ROM internal structure.
- Q.10. Draw an internal logic of 32 x 8 word ROM memory circuit by using OR output gates.
- Q.11. Discuss the 3D structure of RAM with their internal organizational structure.
- Q.12. List the pins required by a (64 x 4) bit RAM memory chip.
- Q.13. Draw the block diagram of a typical (2048 x 16) bits ROM and describes its working principles.
- Q.14. What is flash memory? Where it is used?
- Q.15. A memory chip is organized as (1024 x 4) bits RAM. Find the number of such chips required to obtain:
  - a) (2048 x 8) RAM
  - b) 4k bytes of RAM