UG CA S2-01
(DSC)

FOUR YEAR BACHELOR'S DEGREE COURSE

(CBCS in Accordance with NEP 2020)

SEMESTER - II

DIGITAL TECHNIQUES

COMPUTER SCIENCE

Hiranya Chandra Bhuyan School of Science and Technology

KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Shikhar Kumar Sarma, Gauhati University
Prof. Hiren Kumar Deva Sarma, Gauhati University
Dr. Sanjib Kr. Kalita, Gauhati University

Course Co-ordinator : Dr. Nabankur Pathak
Dr. Ridip Dev Choudhury
Dr. Tapashi Kashyap Das
Dr. Sruti Sruba Bharali

SLM Preparation Team

UNITS	CONTRIBUTORS
1, 2, 3, 4 & 6	Dr. Sangeeta Kakoty, KKHSOU
5, 15	Dr. Nabankur Pathak, KKHSOU
7, 8	Prof. Kandarpa Kr. Sarma, Gauhati University
9	Dr. Kashyap Mahanta, KKHSOU
10 ,13, 14	Dr. Chakradhar Das, Lakhimpur Polytechnic
11, 12	Mr. Swapnanil Gogoi, IDOL, Gauhati University

Editors

Content	**:**	Dr.Jyotiprokash Goswami, Assam Engineering College (Unit 1-6, 9,11,12) Dr. Sruti Sruba Bharali, KKHSOU (Unit 15) Dr. Sanjib Kr. Kalita (Unit 7,8,10,13,14)
Language	**:**	Prof. (Retd.) Robin Goswami, Cotton College (Formerly KKHOU)
Structure, Format & Graphics	**:**	Dr Nabankur Pathak

April, 2024

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State Open University is made available under a Creative Commons Attribution-Non Commercial-Share Alike 4.0 License (international): http://creativecommons.org/licenses/by-nc-sa/4.0/

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

Headquarters: Patgaon, Rani Gate, Guwahati-781017

City Office: Resham Nagar, Khanapara, Guwahati-781022; Web: www.kkhsou.in

The University acknowledges with thanks the financial support provided by the Distance Education Bureau, UGC for preparation of this material.

CONTENTS

		Pages
Unit 1:	**Introduction to Number Systems** Decimal, Binary, Hexadecimal and Octal number system, Number system conversions.	**7-21**
Unit 2:	**Binary Arithmetic** Complement: r's and (r-1)'s complement, Binary addition, Binary subtraction, Binary Multiplication, Binary division.	**22-35**
Unit 3:	**Data Representation** Fixed Point representation and Floating-point representation.	**36-45**
Unit 4:	**Code Conversion** Gray code, BCD, BCD to Excess-3 code conversion technique, ASCII, EBCDIC.	**46-55**
Unit 5:	**Boolean Algebra** Introduction, Properties, De-Morgan's Theorem, Boolean Identities.	**56-61**
Unit 6:	**Logic Gates** Logic Gates: AND, OR, NOT, NAND, NOR, XOR; Conversion of the logic gates.	**62-76**
Unit 7:	**Floating Point Number Representation** Floating point number, Normalization of floating point, overflow and underflow, detection of overflow, IEEE floating point standard.	**77-95**
Unit 8:	**Logic Families** Introduction, Resistor Transistor Logic(RTL), Integrated Injection Logic(IIL), Diode- Transistor Logic (DTL), Emitter-Coupled Logic (ECL), Transistor-Transistor Logic (TTL), TTL-NAND, Tri State Logic, MOS devices, Logic gates with MOSFET's.	**96-111**
Unit 9:	**Reduction Techniques** Boolean Expression, SOP, POS, Conversion of SOP to POS and vice-versa, Simplification of Boolean expression by Algebraic method and Karnaugh Map, Don't care condition.	**112-130**

Unit 10: Binary Arithmetic Circuits **131-140**

Binary Half adder, Full adder, Binary Half subtractor, Full subtractor, 1's complement of n bit adder/subtractor.

Unit 11: Combinational Circuits **141-152**

Multiplexer, 4-to-1 multiplexer, Demultiplexer, Encoder, Priority encoder, Decoder,3-to-8 Decoder.

Unit 12: Sequential Circuits **153-168**

Synchronous and Asynchronous Sequential circuit, Flip-Flop and its different types: RS, JK; Master-Slave JK Flip-Flop.

Unit 13: Registers **169-177**

Shift register, SISO, PISO, SIPO, PIPO shift register, Application of Shift Register.

Unit 14: Counters **178-185**

Synchronous and Asynchronous Counter, Limitations of asynchronous counter, Frequency division counter.

Unit 15: Memory Organization **186-204**

Memory types, Random access memory, Access time, RAM package, Static RSM, CMOS memory cell, Dynamic RAM, Read Only Memory, Memory expansion.

COURSE INTRODUCTION

Welcome to the world of Digital Techniques, where we delve into the essential building blocks of electronic systems, from your computer to your cell phone. At the core of these systems lies the language of zeroes and ones, known as binary code. With this foundational knowledge, we unlock the secrets of crafting electronic circuits that transmit and process information, thanks to logic gates like AND, OR, and NOT.

Throughout this course, you'll embark on a journey through fifteen comprehensive units, each designed to provide you with a panoramic view of the digital underpinnings of computer systems. We kick things off by introducing you to number systems, binary arithmetic, and data representation – the very essence of digital communication.

But we don't stop there; we dive deep into the world of Boolean algebra and logic gates, where you'll discover how these powerful tools shape the digital landscape. You'll also explore reduction techniques, understanding how to simplify complex Boolean expressions, and gain insights into both combinational and sequential circuits – the building blocks of digital functionality.

To add further depth to your understanding, we dedicate substantial sections of this course to registers and counters, illuminating their significance in the world of digital electronics.

In a nutshell, this course offers a comprehensive exploration of digital techniques, from the binary basics to advanced topics like Boolean algebra and circuit design. By the end, you'll have the knowledge and skills to navigate the digital realm with confidence and expertise.

While going through a unit, you will notice some along-side boxes, which have been included to help you know some of the difficult, unseen terms. Again, we have included some relevant concepts in "LET US KNOW" along with the text. And, at the end of each section, you will get "CHECK YOUR PROGRESS" questions. These have been designed to self-check your progress of study. It will be better 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.

UNIT 1: INTRODUCTION TO NUMBER SYSTEMS

UNIT STRUCTURE

1.1 Learning Objectives

1.2 Introduction

1.3 Number System

1.4 Categories of Number System

- 1.4.1 Decimal Number System
- 1.4.2 Binary Number System
- 1.4.3 Octal Number System
- 1.4.4 Hexadecimal Number System

1.5 Number System Conversion

- 1.5.1 Binary to Decimal Conversion
- 1.5.2 Decimal to Binary Conversion
- 1.5.3 Octal to Decimal Conversion
- 1.5.4 Decimal to Octal Conversion
- 1.5.5 Octal to Binary Conversion
- 1.5.6 Binary to Octal Conversion
- 1.5.7 Hexadecimal to Decimal Conversion
- 1.5.8 Decimal to Hexadecimal Conversion
- 1.5.9 Hexadecimal to Binary Conversion
- 1.5.10 Binary to Hexadecimal Conversion

1.6 Let Us Sum Up

1.7 Answers to Check Your Progress

1.8 Further Reading

1.9 Model Questions

1.1 LEARNING OBJECTIVES

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

- define and describe the number system used in computer
- identify different types of number system from their bases
- describe how data is represented in computers.

1.2 INTRODUCTION

Computer system uses different number systems to represent data. In this unit you will be able to understand how data are represented in computers and how they are used.

In this unit, we will introduce the different number systems. In addition to this, we shall discuss the conversion procedure. After completion of this unit we will be able to understand how numbers are converted from one number system to another.

1.3 NUMBER SYSTEM

A number system is a system for expressing numbers, or a mathematical notation for representing numbers of a given set. It uses digits or other symbols. In short, we can say, a number system will:

- represent a useful definite set of numbers
- give every number represented a unique representation
- reflect the algebric and arithmetic structure of the numbers.

There are different ways of representing numbers in computer. In our day-to-day life we use decimal number system which is represented with the combination of ten separate digits from 0 to 9. The base of this number system is 10. But digital computers can store, understand and manipulate information composed of only zeros and ones (0 and 1). So, computers use basically binary number system for which the base is 2. On the other hand, a programmer or a user who works on a computer is allowed to use all digits (0 – 9), letters (in upper and lower case), special symbols present in the key board. So, each decimal digit, letters, symbols etc. written by the programmer/user are converted to binary codes in the form of 0's and 1's within the computer as per the convenience. Coding systems such as ASCII, EBCDIC etc. are universally accepted for representing the numbers as well as the other symbols. On the other hand, we need numbers for calculation or performing different operations in computers. There are different number systems used in computer systems, which are explained in the following section.

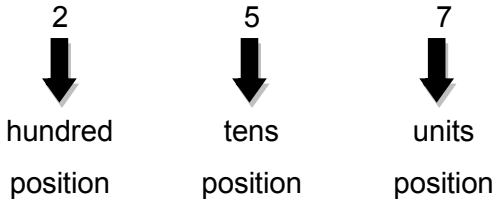
1.4 CATEGORIES OF NUMBER SYSTEM

The categories are according to the base (or radix) of the system. If a number system has base r, then the system has r distinct symbols for r digits. The knowledge of the number system is essential to understand the operation of a computer. In computer, we have four different categories to represent our number system. They are **Decimal, Binary, Octal** and **Hexadecimal** number systems.

1.4.1 Decimal Number System

Decimal number system has ten unique or distinct 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 257, then the digit representations are:



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

Example 1.1: Find the weight of $(6472)_{10}$

The weight of each digit of the decimal no. 6472 is:

$6472 = 6000 + 400 + 70 + 2$

$= 6 \times 10^3 \times 4 \times 10^2 + 7 \times 10^1 + 2 \times 10^0$

The weight of digits from right hand side are-

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

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

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

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

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

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

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

The number 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.

1.4.2 Binary Number System

Only two digits 0 and 1 are used to represent a binary number system. So the base or radix of binary system 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:

<– 64 <– 32 <– 16 <– 8 <– 4 <– 2 <– 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: Find the weight of $(10110)_2$

The weight of bits of the binary number 10110 is:

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

$= 16 + 0 + 4 + 2 + 0$

$= 22$ (decimal number)

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

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

Weight of 2nd bit = 2nd bit $\times 2^1$

.....

.....

and so on.

The weight of the n^{th} bit of the number from right hand side:

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

$= n^{th}$ bit $\times$ (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

1.4.3 Octal Number System

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

$8^3 \leftarrow 8^2 \leftarrow 8^1 \leftarrow 8^0 = 512 \leftarrow 64 \leftarrow 8 \leftarrow 1$

The decimal value 8 is represented in octal as 10, 9 as 11, 10 as 12 and so on. A.s $8 = 2^3$, 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: Octal Number System and their Binary Representations

Decimal Number	Octal Number	Binary Coded Octal No.
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111
8	10	001 000
9	11	001 001
10	12	001 010

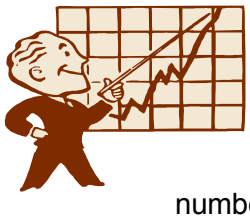
1.4.4 Hexadecimal Number System

The hexadecimal number system is now extensively used in computer industry. Its base (or radix) is 16, and the numbers are represented as 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 1.3 shows the binary equivalent of a decimal number and its hexadecimal representation.

Table 1.3: Hexadecimal Number System and their Binary Representation

Decimal No.	Hexadecimal No.	Binary Coded Hex. No.
0	0	0000
1	1	0001
2	2	0010
:	:	:
:	:	:
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111



CHECK YOUR PROGRESS

Q.1: Fill in the blanks:

a) Number system is used for expressing numbers as well as ..... of a given set.

b) In binary number system, ... represents on and ..... represent off

c) Octal number system uses total ..... numbers.

d) The radix of hexadecimal number system is .....

e) Hexadecimal number is represented by a group of ..... binary bits.

1.5 NUMBER SYSTEM CONVERSION

We now know that, there are different ways of representing numbers in computer by using different number system.

As the computer uses different number systems, there is a provision for converting one number system to others. The processes of converting generally used decimal number systems to other number systems and vice-versa are mentioned in the next few sections.

1.5.1 Binary to Decimal Conversion

To convert a binary number to its decimal equivalent we use the following expression. The weight of the n^{th} bit of the number from right hand side is : = n^{th} bit × 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 1.3: Convert binary $(110100)_2$ to its decimal equivalent.

Solution:

$$(110100)_2 = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$$
$$= 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^2$$
$$= 32 + 16 + 4$$
$$= (52)_{10}$$

Therefore, $(110100)_2 = (52)_{10}$

Example 1.4: Converting binary fraction $(111011.101)_2$ to its equivalent decimal fraction.

Solution:

$$(111011.101)_2 = (1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) + (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3})$$
$$= (32 + 16 + 8 + 0 + 2 + 1) + (0.5 + 0 + 0.125)$$
$$= (59.625)_{10}$$

Therefore, $(111011.101)_2 = (59.625)_{10}$

1.5.2 Decimal to Binary Conversion

There are different methods used to convert decimal number to binary number. The most common method is the repeated-division method. In this method, the number is successively divided by 2 and its remainders whether 0's or 1's are recorded. The final binary result is obtained by assembling the remainders in reverse order to obtain the binary equivalent of the decimal number. In this case, the last remainder will be the most significant bit (MSB).

Example 1.5: Convert $(75)_{10}$ to its binary equivalent.

Solution:

2 |75	Remainder	
2|37	1	↑
2|18	1	
2|9	0	Read in this
2|4	1	order
2|2	0	
2|1	0	
0	1	

So, $(75)_{10} = (1001011)_2$

The method to convert the fraction part of a decimal number to its binary equivalent is that, it should repeatedly multiply the fraction part by 2 to get an integer part and a fraction part. The new fraction part again is to be multiplied by 2 to get a new integer and fraction part. This process is to be repeated until the fraction part becomes zero or we obtain the required number of digits. Integers, so obtained are to be placed next to the binary point in the order in which they are produced i.e. the first integer appears first and so on.

Example 1.6: Convert decimal fraction $(12.75)_{10}$ to its equivalent binary fraction.

Solution:

2|12	Remainder	MSB	.75	
2|6	0	↑	× 2	
2|3	0	|	1.50	Read
2|1	1	↓	× 2	the MSB
0	1		1.00	bits.

So, $(12)_{10} = (1100)_2$ and $(.75)_{10} = (.11)_2$

Now, by combining both we will get $(12.75)_{10} = (1100.11)_2$

1.5.3 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 1.7: Find decimal equivalent of octal number $(153)_8$.

Solution: $1 \times 8^2 + 5 \times 8^1 + 3 \times 8^0$

$= 64 + 40 + 3$

$= 107$

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

Example 1.8: Find decimal equivalent of octal number $(123.21)_8$.

Solution: $(1 \times 8^2 + 2 \times 8^1 + 3X8^0) + (2 \times 8^{-1} + 1 \times 8^{-2})$

$= (64 + 16 + 3) + (0.25 + 0.0156) = 83.2656$

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

1.5.4 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 1.9: Find the octal equivalent of decimal $(3229)_{10}$.

Solution:

	Remainders	
8 | 3229		↑
8 | 403	5	read in
8 | 50	3	reverse
8 | 6	2	order
0	6	

So, $(3229)_{10} = (6235)_8$

Example 1.10: Find the octal equivalent of $(.123)_{10}$.

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

$8 \times 0.123 = 0.984$	0	↓
$8 \times 0.984 = 7.872$	7	read in
$8 \times 0.872 = 6.976$	6	forward order
$8 \times 0.976 = 7.808$	7	

We 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 $(0.123)_{10} = (0.0767)_8$

1.5.5 Octal to Binary Conversion

To convert an octal number to its binary equivalent, we just have to replace each bit separately by its 3-bit binary equivalent. For example, process to convert octal number 15 to its binary equivalent is to replace binary equivalent of octal 1 and 5 separately. 3-bit Binary equivalent of 1 and 5 are 001 and 101 respectively. Therefore, the binary equivalent of octal 15 is 001101.

Example 1.11: Find the Binary equivalent of $(124.73)_8$.

Solution: Given octal number is 1 2 4 . 7 3

Converting each to binary is 001 010 100 . 111 011

Therefore, the result is $(001010100.111011)_2$

1.5.6 Binary to Octal conversion

This is the opposite of the above conversation. To convert a binary number to its equivalent octal, make a group of 3 bits each starting from right to the last for the numbers before decimal point and left to right for the numbers after the decimal point. Then replace each group by its equivalent octal number.

Example 1.12: Find the Octal equivalent of $(1110011.1111)_2$

Solution: Binary number after making group of 3 is:

1 110 011 111 1

The equivalent octal number is: 1 6 3 7 1

Therefore, the result is $(163.71)_8$

1.5.7 Hexadecimal to Decimal Conversion

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

Example 1.13: Find the decimal equivalent of $(4A83)_{16}$

Solution: $(4A83)_{16} = (4 \times 16^3) + (10 \times 16^2) + (8 \times 16^1) + (3 \times 16^0)$

$= 16384 + 2560 + 128+3$

$= (19075)_{10}$

$(4A83)_{16} = (19075)_{10}$

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

Solution: $(53A.0B4)_{16} = (5 \times 16^2) + (3 \times 16^1) + (10 \times 16^0) + (0 \times 16^{-1}) + (11 \times 16^{-2}) + (4 \times 16^{-3})$

$= 1280 + 48 + 10 + 0 + 0.04927 + 0.0009765$

$= (1338.0439)_{10}$

$(53A.0B4)_{16} = (1338.0439)_{10}$

1.5.8 Decimal to Hexadecimal Conversion

To convert a decimal integer number to hexadecimal, successively divide the given decimal number by 16 till the quotient is zero. The last remainder is the MSB (Most Significant Bit). The remainders read from bottom to top give the equivalent hexadecimal integer.

To convert a decimal fraction to hexadecimal, successively multiply the given decimal fraction by 16, till the product is zero or till the required accuracy is obtained, and collect all the integers to the left of decimal point. The first integer is the MSB and the integers read from top to bottom give the hexadecimal fraction.

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

Solution: Let us first consider $(1234)_{10}$

Remainder

	Decimal		Hexadecimal
16|1234	2		2
		read in	
16|77	13	reverse	D
		order	
16|4	4	4	4

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

Now considering the decimal part $(0.675)_{10}$.

Hexadecimal conversion of fractional part of a decimal number is as follows:

	Decimal	Hexadecimal
0.675 × 16 = 10.8	10	A
0.800 × 16 = 12.8	12	C
0.800 × 16 = 12.8	12	C
0.800 × 16 = 12.8	12	C

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

Hence, $(1234.675)_{10} = (4D2.ACCC)_{16}$.

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 hexadecimal first, and then convert the hexadecimal to binary.

1.5.9 Hexadecimal to Binary Conversion

To convert a hexadecimal number to its binary equivalent, we just have to replace each bit separately by its 4-bit binary equivalent. For example, the process to convert hexadecimal number B5 to its binary equivalent is to replace binary equivalent of hexadecimal B and 5 separately. 4-bit Binary equivalent of B and 5 are 1011 and 0101 respectively. Therefore, the binary equivalent of hexadecimal B5 is 10110101.

Example 1.16: Find the binary equivalent of $(12A.D3)_{16}$.

Solution:

The given hexadecimal number is 1 2 A . D 3

Converting each to binary is 0001 0010 1010 . 1101 0011

Therefore, the result is $(000100101010.11010011)_2$

1.5.10 Binary to Hexadecimal Conversion

To convert a binary number to its equivalent hexadecimal, make a group of 4 bits each starting from right to the last for the numbers before decimal point and from left to the last for the numbers after decimal point and replace each group by its equivalent hexadecimal number.

Example 1.17: Find the hexadecimal equivalent of $(11100111.1111)_2$.

Solution: Given binary number after making group of 4 is:

1110 0111 . 1111

The equivalent hexadecimal number is: E 7 . F

Therefore, the result is $(E7.F)_8$.



CHECK YOUR PROGRESS

Q.2: Convert the following:

a) $(65)_{10}$ to its equivalent binary number.

b) $(256)_8$ to its decimal equivalent.

c) $(13B)_{16}$ to its decimal equivalent.

d) $(1010.110)_2$ to its decimal equivalent.



1.6 LET US SUM UP

- In this unit, we have discussed the number system used in computer and its different categories.
- We are familiar with decimal number system, but computer cannot understand our decimal system Computers can understand only their number system. These are binary, octal and hexadecimal number system.
- As computer uses three other number systems, decimal number system should be converted to one these number systems. Therefore, conversion from one number system to another is important in computer.
- In short,
- radix or base of decimal number system is 10, since it consists of 10 (ten) distinct/unique symbols or digits viz., 0 to 9.
- radix or base of binary number system is 2, since it consists of only 2 distinct/unique digits viz., 0 and 1.
- radix or base of octal number system is 8, with only 8 distinct unique digits, viz. 0 to 7.

- radix or base of hexadecimal number system is 16, where number of distinct/unique symbols is 16 and they are 0 to 9,A,B,C,D,E and F.



1.7 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) mathematical notation; b) 1, 0; c) 8; d) 6; e) 4

Ans. to Q. No. 2: a) $(1000001)_2$; b) $(174)_{10}$; c) $(316)_{10}$; d) $(10.75)_{10}$



1.8 FURTHER READING

1) ITL Education Solutions Limited, & Sargunar, J. (2011): *Introduction to Computer Science*; Pearson Education India.

2) Morris, M. M. (1987): *Digital Logic and Computer Design*.

3) Ram, B. (2000): *Computer Fundamentals: Architecture and Organization*; New Age International.

4) Sinha, P. K., & Sinha, P. (2010): *Computer Fundamentals* (Vol. 4); BPB publications.



1.9 MODEL QUESTIONS

Q.1: Define number system? What are the different number systems used in computer? Explain.

Q.2: What is 'Base' or 'radix' of a number system? Write the radix of binary, decimal, octal and hexadecimal number system.

Q.3: How to convert octal number to its binary equivalent?

Q.4: Convert the following:

a) $(156.4)_8$ to its binary equivalent

b) $(A5F.B7)_{16}$ to its binary equivalent

c) $(269.25)_{10}$ to its binary equivalent

d) $(1010110.001)_2$ to its decimal equivalent

e) $(110011.110)_2$ to its octal equivalent

f) $(111001100.1010)_2$ to its hexadecimal equivalent

*** ***** ***

UNIT 2: BINARY ARITHMETIC

UNIT STRUCTURE

2.1 Learning Objectives

2.2 Introduction

2.3 Sign Number Representation

2.4 Sign Magnitude

2.5 Complement of Numbers

- 2.5.1 (r-1)'s Complement
- 2.5.2 r's Complement
- 2.5.3 1's Complement
- 2.5.4 2's Complement

2.6 Binary Arithmetic

- 2.6.1 Addition
- 2.6.2 Subtraction
- 2.6.3 Multiplication
- 2.6.4 Division

2.7 Let Us Sum Up

2.8 Answers to Check Your Progress

2.9 Further Reading

2.10 Model Questions

2.1 LEARNING OBJECTIVES

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

- define the sign magnitude of binary numbers and their representation
- define the complement of a number
- find out the complement form of a given number
- do arithmetic operations on binary numbers.

2.2 INTRODUCTION

In our previous unit we have learnt about the different number systems used in computers We have also leant how one number system

can be converted to another. But we have not discussed how a negative number is represented in computer. Therefore, in this unit, we will discuss various representations of negative numbers in computer systems.

We will also discuss about the complements of a number which is used in the arithmetic operations. In addition this unit describes the methods to perform arithmetic operations like addition, subtraction, multiplication and division.

2.3 SIGN NUMBER REPRESENTATION

In computing, representation of negative number is important because digital systems are not able to understand "–" sign which we use as prefix to represent negative sign. Therefore, signed number representations are required to encode negative numbers in binary number systems.

The preferred methods to represent sign bit in binary number system are sign magnitude method and complements of a number. There is no definitive criterion by which one can say which one of the representations is universally superior. The representation used in most current computing devices is two's complement. PDP-11, VAX, MIPS, SPARC, ARM, Itanium, PA-RISC, DEC Alpha processors all use two's complement. But CDC 3000 and 6000 series, and the Unisys ClearPath Dorado series mainframes, use one's complement.

2.4 SIGN MAGNITUDE

In this approach, the most significant bit i.e. the left most bit is allocated for the sign bit separately. For the negative sign, the value of this bit is 1, and for positive number it is 0. The remaining bits in the number indicate the magnitude or absolute value. Hence, in a byte, only 7 bits, ranging from 0000000 (0) to 1111111 (127), are used to represent the number. Thus numbers ranging from -127_{10} to $+127_{10}$ can be represented once the sign bit (the 8th bit) is added.

It is directly comparable to the common way of showing a sign (placing a "+" or "–" before the number's magnitude). Some early binary computers

(e.g., IBM 7090) used this representation. Signed magnitude is the most common way of representing negative values.

2.5 COMPLEMENT OF NUMBERS

Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. Once are find out the complement of the number to be subtracted (subtrahend), it is added with the other number (minuend) to obtain the result of subtraction operation. 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.

2.5.1 (r-1)'s Complement

Given a number N with 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 37 is (99 – 37) = 62

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

2.5.2 r's Complement

The r's complement of an n-digit number N in base r (r > 1) is defined as r^n – N for all N i.e. not equal to zero. Comparing with the (r–1)'s complement, here 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 37 is (99–37) + 1 = 62 + 1 = 63

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

Or, without help of 9's complement also we can directly find out 10's complement with the formula r^n–N

For example, for the same value, 10's complement of 37 is:

$N = (37)_{10}$

here r = 10

n = 2

Hence, we can write the 10's complement of this number as:

10^2 - 37 = 63.

Hence, we can say that 10's complement of 37 is 63.

2.5.3 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 the binary number from $(1111)_2$.

However, the subtraction of a binary digit 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 101010 is 010101.

2.5.4 2's Complement

It is obtained by adding 1 to 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 10101 is 01010 + 1 = 01011.



CHECK YOUR PROGRESS

Q.1: Find 9's complement and 10's complement of decimal numbers 54 and 172.

Q.2: Find 1's complement and 2's complement of binary numbers 101100 and 0000.

2.6 BINARY ARITHMETIC

There are four basic arithmetic operations in binary arithmetic. They are explained in the next few sections.

2.6.1 Addition

In the binary number system, 1 + 0 = 1

and

0 + 1 = 1.

When 1 is added to 1, the sum is 0 with a carry 1. If the sum is written in 2 bits it is equal to 10 (2 decimal). The table is shown below:

BIT 1	BIT 2	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The complexity may arise when adding combination of positive and negative binary number. In this case the arithmetic addition is

dependent on the representation of the number from any one of the following:

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

Let us discuss this with the help of following example:-

Example 2.1: Add 25 and –30 in binary using 7 bit register in:

signed magnitude representation
signed 1's complement representation
signed 2's complement representation

Solution: In Signed magnitude representation: To do the arithmetic addition with a negative number in signed magnitude representation we have to follow the rules of ordinary arithmetic. First, we have to check the magnitude of the numbers. The number having smaller magnitude is subtracted from the bigger number and the sign of bigger number is selected.

For example, (+25) + (–30)
= –(30 -25)
= –5

This process requires the comparison of the sign and the magnitude of the number and then taking decision, whether to perform addition or subtraction operation.

In signed 2's complement representation:

We get that +30 is 0 011110
–30 is 1 011110

Now, 2's complement of –30 (not including sign bit) 1 100010
+25 is 0 011001

Now, addition of these two is,

+25	0	011	001	
–30	1	100	010	
–05	1	111	011	(Just add the numbers)

The result shows the sign bit as 1. It means that the value is negative, which is true in general subtraction also.

The result for negative number is stored in signed 2's complement form and hence the above result is in signed 2's complement format. We have to again take 2's complement of the above result to interpret the result in decimal system. Therefore, 2's complement of 111011 is,

1 000 100 + 1 = 1 000 101 which is –05 in decimal system



Sign bit

From the above example it can be noticed that, signed 2's complement representation is simpler than signed magnitude representation. This representation puts additional constraint that the negative numbers should be stored in signed 2's complement form in the register. We get the signed 2's complement by complementing the positive number bit by bit and then incrementing the resultant by 1.

In signed 1's complement representation: In this method we add the two numbers including the sign bit. If there is a carry of the most significant bit or sign bit, then we add it to the sum to obtain the result. In case of negative results, the numbers will be in 1's complement representation. We have to take 1's complement again for interpreting the result.

+25 = 0 011 001

–30 = 1 100 001 (1's complement of –30)

–5 = 1 111 010

Note: CARRY: When we add two binary numbers of n bits each and the result shows a "1" occupies the (n+1)th position of the result then we say a carry occurs or there is a carry.

The result is stored 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.

Let us consider another example: –25 + 30

25	1	100	110	(1's complement of 25)
+30	0	011	110	
+5	1 0	000	100	

CARRY

In the above result, there is a carry, so we add it to the resultant sum.

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

Now, let us try another example: –25 – 30

–25	1	100	110	(1's complement of 25)
–30	1	100	001	(1's complement of 30)
–55	1 1	000	111	

There is a carry in the above result. So we add it to sum to obtain the result.

So, sum = 1 001 000 which is in 1's complement form. Now taking 1's complement of this we get –55 in decimal form.

1 110111 which is –55 in decimal form

Sign bit

Representation of 0 (zero): The interesting feature is the representation of 0 in signed magnitude and 1's complement. There are two representations for zero and they 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 representation of zero.

+0 000000

–0 in 2's complement is +0 = 1 111111

$$\begin{array}{r} 1 \\ \hline 1 \quad 0000000 \end{array}$$

Now, discarding the carry, it is 0 000 000.

Thus, both +0 and –0 are same in 2's complement notation. This is an added advantage in favor 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 $-(2^7-1)$

$= 128 - 1$ to $-(128-1)$

$= 127$ to -127

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

2.6.2 Subtraction

Though there are many methods of performing subtraction, here we will consider the method of subtraction known as complementary subtraction where, subtraction operation is realized in terms of addition operation. This is a more efficient method of subtraction while using electronic circuits. The following are the ways of subtracting binary numbers in 1's and 2's complement methods.

1's complement method:

Steps: 1) Find the 1's complement of the number to be subtracted (subtrahend).

2) Add the other number (minuend) with the complement value obtained from the previous step (step 1).

3) If there is a carry of 1 after addition, add the carry with the result obtained.

Else, take complement of the result again and attach a negative sign with the result.

Example 2.2: Subtract 3 -5 by 1's complement method.

Solution: 3 011

5 101

Step 1: 1's complement of 5 is 010

Step 2: Adding 010 with 011 gives us result:

```
  010
  011
  101
(No carry)
```

Step 3 : As there is no carry in the result, we again take complement of the result and the final result is now 010. After attaching the negative sign with the final result, the obtained result is now –10 (in binary) which is –2 in decimal form.

2's complement method: It is almost same as 1's complement method. Steps 1 & 2 are same as 1's complement method; instead we have to take 2's complement here. Step 3 is different from 1's complement.

Following are the Steps:

1) Find the 2's complement of the number (subtrahend) which is subtracting.
2) Add the other number (minuend) with the complement value obtained from Step 1.
3) If there is a carry, in the result obtained from step 2, ignore it. Else, take again 2's complement of the result and attach a negative sign with the result.

Example 2.3: Subtract 3 –5 by 2's complement method.

Solution: Binary form of 3 is: 011

Binary form of 5 is: 101

Step 1: The 2's complement of 5 is 011 (1's complement 010+1)

Step 2: Adding 011 with 011 will give result:

```
  011
  011
  110
(No carry)
```

Step 3: As there is no carry with the result, again have to take 2's complement of 110 which is 001 + 1 = 010 and attaching a negative sign the required result is –10 (in binary) which is –2 in decimal form.

Overflow: An overflow occurrs when the sum of two 'n' digits number occupies 'n + i' digits. This definition is valid for both binary as well as decimal digits. Every computer employs a fixed limit for representation of numbers. For example, in our examples we have been using 8 bit registers for calculating the sum results. But what will happen, if the sum of the two numbers is a combination of 9 bits? Where will we store the 9th bit? This problem can be clearly understood with the help of the following example.

Example 2.4: Add the numbers 65 and 75 in 8 bit register.

Solution:

Number	sign bit	binary form
65	0	1000001
75	0	1001011
140	1	0001100

The result indicates a negative number as the sign bit of the result is 1 and the number has a value equal to –16 which is a wrong result. This has error in result has occurred because of overflow.

Now the question is how does the computer know that overflow has occurred? If the carry is equal 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 = 0		
Sign bit = 1			Sign bit = 1		
Over flow			No over flow		

2.6.3 Multiplication

Multiplication in binary follows the same operations that are followed in the decimal system. But we use only two digits 0 and 1 here. The table to be remembered for multiplication is:

0 × 0 = 0

0 × 1 = 0

1 × 0 = 0

1 × 1 = 1

For example: 10101 × 11001

```
  10101
× 11001
-------
  10101
 00000
00000
10101
10101
----------
1000001101
```

2.6.4 Division

The complete table for binary division is: 0/1 = 0

1/1 = 1

The steps for binary division are:

1) Start from the left of the number to be divided.
2) Perform subtraction in which the divisor is subtracted from the dividend.
   a) If subtraction is possible then put a 1 in the quotient and subtract the divisor from the corresponding digits of the dividend else 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.

For example, 100001/110

Then,

```
                 101     (Quotient)
(Divisor)  110 | 100001      (Dividend)
                 110         --Step 1
                ----
                1001         --Step 2b
                 110         --Step 2a
                ----
                  11         (Remainder)
```



CHECK YOUR PROGRESS

Q.3: Add 35 and –30 in binary using 7 bit register in 2' complement representation.

Q.4: Subtract $(1010101)_2 - (1001100)_2$ by 2's complement method.

Q.5: Multiply $(1100)_2$ by $(1001)_2$.

Q.6: Divide $(101101)_2$ by $(10)_2$.



2.7 LET US SUM UP

- The methods to represent sign bit in binary number system are sign magnitude method and 2's complements of a number.
- In signed magnitude representation, the most significant bit is allocated for the sign bit separately.
- The complement of a binary number is obtained by inverting all the bits.
- Given a number N with base r having n digits, the (r–1)'s complement of N is defined as $(r^n–1) – N$.
- The r's complement of an n-digit number N in base r (r > 1) is defined as $r^n – N$ for N where N is not equal to zero.



2.8 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: 9's complement of 54 is 45 and 172 is 827
10's complement of 54 is 46 and 172 is 828.

Ans. to Q. No. 2: 1's complement of 101100 is 010011 and 0000 is 1111
2's complement of 101100 is 010100 and 0000 is 10000.

Ans. to Q. No. 3: $(000110)_2$

Ans. to Q. No. 4: $(0001001)_2$

Ans. to Q. No. 5: $(1101100)_2$

Ans. to Q. No. 6: $(10110)_2$



2.9 FURTHER READING

1) Morris, M. M. (1987): *Digital Logic and Computer Design*.

2) Ram, B. (2000): *Computer Fundamentals: Architecture and Organization*; New Age International.

3) Sinha, P. K., & Sinha, P. (2010): *Computer Fundamentals* (Vol. 4); BPB publications.



2.10 MODEL QUESTIONS

Q.1: Define overflow.

Q.2: Explain sign magnitude representation.

Q.3: Perform the addition of the following using 7 bit register in sign 2's complement method.

a) +25 and +30

b) +25 and –30

c) –25 and –30

d) –25 and +30

Q.4: Perform the above operations in 1's complement method.

Q.5: Subtract 65 from 70 by using 1's complement and 2's complement method.

Q.6: Divide $(11001.101)_2$ by $(101)_2$

Q.7: Multiply the following:

a) $(1101.11 \times 101.11)_2$

b) $(10101.1 \times 101.101)_2$

*** ***** ***

UNIT 3: DATA REPRESENTATION

UNIT STRUCTURE

3.1 Learning Objectives

3.2 Introduction

3.3 Data Representation

- 3.3.1 Fixed Point Representation
- 3.3.2 Floating Point Representation

3.4 IEEE 746 Standards

3.5 Let Us Sum Up

3.6 Answers to Check Your Progress

3.7 Further Reading

3.8 Model Questions

3.1 LEARNING OBJECTIVES

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

- define fixed point representation
- define floating point representation
- define overflow and underflow of floating point numbers
- describe IEEE 746 standards.

3.2 INTRODUCTION

In the previous two units we have learnt about different data representation methods including sign bit. In this unit, we will learn about fixed point and floating point representation. In addition to this, the unit also covers the explanation of IEEE 746 standards.

3.3 DATA REPRESENTATION

Data are usually formed by using the alphabets A to Z, 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 also 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*** in 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, etc. and non-arithmetic operations. In this unit, we will consider only numeric data representations.

A register must have n flip-flops to store a binary number of n bits. A flip flop is a binary cell which can store a bit of information.This n bit binary number represents the magnitude of the number but does not tell us about its sign or the position of the binary point. The position of the binary point is needed to represent integers, fractions or mixed integer-fraction numbers. We have already discussed in the previous units how to use the left most bit for the purpose of indicating sign of the number. The representation of the binary point is complicated by the fact that it is characterized by a position between two flip flops in the register, i.e. no bit of the register is reserved for the binary point.

There are two ways of specifying the position of the binary point in a register, viz., fixed point representation and floating point representation.

3.3.1 Fixed Point Representation

In this method, it is assumed that the binary point is always fixed in one position.

Two most widely used positions of the binary point are:

a) in the extreme left of the register, making the stored number a fraction and
b) in the extreme right of the register to make the stored number a integer.

The binary point is not physically visible in both the cases, instead it is assumed from the fact that the number stored in the register is treated as a fraction or as an integer. Sign of a positive

fixed point binary number is represented by 0 and the rest of the bits are representing the magnitude of the positive number. In case of a negative number the sign bit is 1 and the rest of the number may be represented in any one of the following ways.

1) Sign magnitude
2) Sign–1's complement
3) Sign–2's complement

1) Sign-magnitude representation: Here, the left most bit is used for indicating the sign bit and the other bits are used for magnitude, which is represented by a positive binary number. For example, the binary equivalent of the number 8 is written in sign-magnitude representation as: (using 7 bit register)

+8		–8	
0	001000	1	001000
↑		↑	
Sign bit		Sign bit	

2) Sign–1's complement representation: In sign 1's complement representation, all the bits of the positive number, including the sign bit are complemented to obtain the negative number as shown below.

+8		–8	
0	001000	1	110111
↑		↑	
Sign bit		Sign bit	

3) Sign–2's complement representation: In this representation, the negative number is obtained by taking the 2's complement of the positive number, including the sign bit.

+8		–8	
0	001000	1	111000
↑		↑	
Sign bit		Sign bit	

Though the signed magnitude system is easier to interpret, it is not always efficient. The circuits for handling numbers are

simplified if 1's or 2's complement systems are used and as a result mostly, one of these systems is almost always used.

Note 1: In 1's and 2's complements, all positive integers are represented as in sign magnitude 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.

3.3.2 Floating Point Representation

A number which has both an integer part and 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. To represent a number in floating point representation, we need two registers. The first represents a signed-fixed point number and the second, the position of the decimal/binary/radix point.

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 the decimal number +15.37, the floating point notation is given below:

$15.37 = 0.1537 \times 10^2$ or 1537×10^{-2}

Now, the floating point representation of 0.1537×10^2 is:

Sign		Sign	
0	.1537	0	02
Mantissa (Fraction)		Exponent	

The floating point representation of 1537×10^{-2} is:

Sign		Sign	
0	1537	1	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:

Sign		Sign	
0	.10111010	0	000100
Mantissa (Fraction)		Exponent	

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. They 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, 0.00038695×10^5 and 0.0589×10^{-4} are not normalized numbers. But 0.38695×10^2 and 0.589×10^{-5} are normalized numbers. Similarly, the binary numbers 0.0011001×2^8 and 0.0001011×2^{-5} are not non-normalized binary numbers. But 0.11001×2^6 and 0.1011×2^{-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 longer time for execution and also requires

complex hardware. But floating point representation is frequently used in scientific calculations.

Overflow and Underflow: When the result is too small or too large to be represented by the computer, an overflow or underflow condition occurs. When two floating-point numbers of the same sign are added, a carry may be generated. 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 may be lost in the process of alignment. This is referred to as *mantissa underflow.* To perform the multiplication of two floating point numbers, the exponents are added. In certain cases the sum of the exponents may be too large and 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 cannot be adjusted and hence, an error indication has to be displayed on the computer screen.



CHECK YOUR PROGRESS

Q.1: Represent 10 by the following representation methods:

a) Signed magnitude representation.

b) Signed 2's complement representation.

Q.2: Represent (1010.1010)2 with floating point representation method.

Q.3: Fill in the blanks:

a) When all the bits of the computer word are used to represent the number, it is called ..... of the number.

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

c) In general, ... of a number is used to represent sign bit in signed magnitude representation.

d) ..... takes longer time for execution and requires complex hardware.

e) An ..... occurs when the result is too small to be represented by the computer.

3.4 IEEE 746 STANDARDS

Each digital component has its own unique graphic symbol which is commercially recognized and from which a user can recognize its function. To give the recognition globally, a standard is needed and so the IEEE standards came into existence. It has various standards for different purposes.

The IEEE 746 standard describes the method for measuring performance level of uniformly coded analog-to-digital (A/D) and digital-to-analog (D/A) converters for pulse code modulation (PCM) television video signals. It was established in 1984. The services of this standard are mainly for:

- monitoring and tracking
- access to product
- service and patent catalog
- advertise

One advanced version of this standard, IEEE 754, is the standard used to do floating point computation. It was established in 1985 by the Institute of Electrical and Electronic Engineers (IEEE). The standards addressed the problems arising in floating point representation.

The standard defines:

- **arithmetic formats:** sets of binary and decimal floating-point data, which consist of finite numbers, infinities, and all special symbols.
- **interchange formats:** used to exchange floating-point data in an efficient and compact form
- **rounding rules:** properties to be satisfied when rounding numbers during arithmetic operations and conversions
- **operations:** all arithmetic and other operations
- **exception handling:** handling of exceptional conditions like division by zero, overflow, etc.

The standard is reviewed and revised from time to time to incorporate new ideas and methods. The last revision was done in the September 2015 to incorporate clarifications.

The standard also includes functions like advanced exception handling and additional operations like trigonometric functions, expression evaluation.



CHECK YOUR PROGRESS

Q.4: Fill in the blanks:

a) ..... is a technically recognized standard.

b) IEEE 746 standard is used in ..... television video signals.

c) IEEE 754 is the standard used to do .....

d) ..... measures the performance level of converter.

e) Monitoring and tracking is the service of ..... standards.



3.5 LET US SUM UP

- 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.

- For fixed point we have discussed three different methods, they are:
- Signed magnitude representation method
- Signed 1's complement representation method
- Signed 2's complement representation method
- **Signed magnitude representation:** It uses one bit, usually the leftmost bit, to indicate the sign where '0' indicates the positive integer and the '1' indicates the negative integer. The rest of the bits are used as magnitude.
- A number which has both an integer part and a fractional part is called real number or floating point number. A floating point number can be either positive or negative.
- 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.***
- 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.
- 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 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.
- The IEEE 746 standard describes the method for measuring performance level of uniformly coded analog-to-digital (A/D) and digital-to-analog (D/A) converters for pulse code modulation (PCM) television video signals.



3.6 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) 1 001010; b) 1 110110

Ans. to Q. No. 2: Mantissa is 10101010, fraction is 000100 (in 16 bit

representation)

Ans. to Q. No. 3: a) unsigned representation; b) zero; c) leftmost bit; d) Arithmetic operation; e) overflow or underflow

Ans. to Q. No. 4: a) IEEE; b) pulse code modulation (PCM); c) floating point computation; d) IEEE 746; e) IEEE 746



3.7 FURTHER READING

1) Morris, M. M. (1987): *Digital Logic and Computer Design*.

2) Ram, B. (2000): *Computer Fundamentals: Architecture and Organization*; New Age International.

3) Sinha, P. K., & Sinha, P. (2010): *Computer Fundamentals* (Vol. 4); BPB publications.



3.8 MODEL QUESTIONS

Q.1: How can data be represented in computer? Explain the fixed point representation method with example.

Q.2: How can decimal number be represented?

Q.3: What is mantissa and exponent of decimal representation?

Q.4: How can a floating point number be normalized for computerized representation?

Q.5: What do you mean by overflow and underflow?

Q.6: Explain IEEE 746 standard.

Q.7: Write the functions of IEEE 746 standard.

*** ***** ***

UNIT 4: CODE COVERSION

UNIT STRUCTURE

4.1 Learning Objectives

4.2 Introduction

4.3 BCD

4.4 ASCII

4.5 EBCDIC

4.6 Gray Code

- 4.6.1 Conversion from Binary to Gray Code
- 4.6.2 Conversion from Gray to Binary Code

4.7 Let Us Sum Up

4.8 Answers to Check Your Progress

4.9 Further Reading

4.10 Model Questions

4.1 LEARNING OBJECTIVES

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

- define standards of coding
- describe different coding techniques
- describe conversion from one coding form to another.

4.2 INTRODUCTION

Coding of characters has been standardized 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 +, -, *, /, ;, ., (,), =, 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. Some of the popular coding systems are discussed in the following sections.

4.3 BCD

In computing and electronic systems, **Binary-Coded Decimal (BCD)** is a way to express each of the decimal digits with its binary equivalent. It is a four digit binary code for representing 0 to 9 in decimal form. It allows easy conversion to decimal digits for printing or display and helps in faster decimal calculations. Decimal numbers with their BCD equivalent are given in the following table:

Table 4.1: Decimal Number with BCD Equivalents

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 is shown below:

Example 4.1: Convert the decimal numbers 35 and 170 to BCD.

Solution: 3 = 0011 and 5=1010

35 = 00111010

Similarly, 170 = 000101110000

Example 4.2: Convert each of the BCD code 10000111 to decimal.

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

1000 0111

8 7

Thus, 10000111(in BCD) = 87 (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 an example to show 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 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 invalied states and return the code to BCD form. The six invalied states are 1010, 1011, 1100, 1101, 1110, 1111. If a carry again results when 6(0110) is added, we simply add the carry to the next 4-bit group.

Example 4.3: Add the following BCD numbers:

a) 0001 + 0100

b) 0000111 + 01010011

Solution: The decimal number addition is shown for comparison:

a) 0001 1

+0100 +4

0101 5

0001 + 0100 = 0101

b) 1000 0111 87

+0101 0011 +53

1101 1010 Both groups are invalid (>9) +140

Now, adding 6 (i.e. 0110) to both groups, we are getting as:

	1101	1010
	+0110	+0110
0001	0100	0000

It is now valid BCD number which is 140 in decimal.

4.4 ASCII

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

However, there is no such standard way to represent letters and other non-numeric characters with 0's and 1's. Therefore, in order to do that, computers use *ASCII tables,* which are tables or lists that contain all the letters used 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 can be easily represented using 0's and 1's in binary: 65 when 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.

4.5 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.



CHECK YOUR PROGRESS

Q.1: Fill in the blanks:

a) BCD is the code of ..... bit representation.

b) EBCDIC uses ..... bit per character.

c) The ASCII is a standard of ..... bits code.

d) The full form of ASCII is .....

e) ..... is the common code for microcomputer equipment.

f) ..... code is used in IBM mainframe models.

Q.2: Add the following BCD numbers:

i) 00100011 + 00010101

ii) 1001 + 0100

4.6 GRAY CODE

The gray code is an un-weighted 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 4.2: Decimal, Binary and Gray Codes

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

4.6.1 Conversion from Binary to Gray code

The following rules explain how to convert from binary number to gray code:

a) The most significant bit (left most) in the gray code is the same as the corresponding most significant bit in the binary code.

b) Going from left to right, add each pair of adjacent bits of binary code to get the next gray code bit. Discard the carry.

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

Binary code 1 1 0 0

As per the rule, the MSB of Gray code will be same as MSB of Binary code, which is 1 in this example.

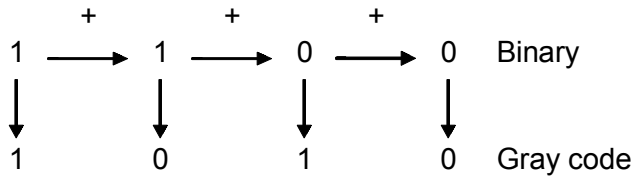
Now, by adding each pair of adjacent bits of Binary code:

1 + 1 = 10 = 0, discarding the carry 1

1 + 0 = 1, no carry

0 + 0 = 0, no carry

Therefore, 1100 in gray code is 1010. The process is graphically explained below:



4.6.2 Conversion from Gray to Binary Code

The following rules apply for the conversion from gray code to binary:

a) The most significant bit (left most) in the binary code is the same as the corresponding most significant bit in the gray code.

b) Add each binary code bit generated by gray code, to the next adjacent position of the gray code. Discard the carry.

It will be able to understand it more clearly from the following example.

Let the gray code be 10011 where MSB is 1. So the MSB for the resultant binary code will also be 1. For the remaining bits:

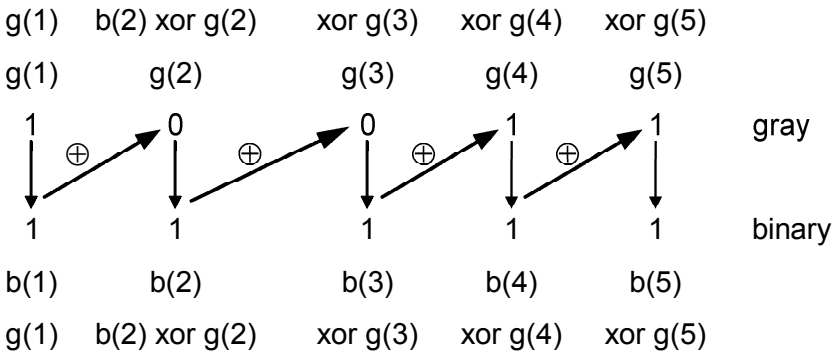
Now: (MSB) 1 + 0 (2^{nd} bit position of gray code) = 1 (2^{nd} bit position of resultant binary code). In the same way,

1 (2^{nd} bit position of binary code) + 0 (3^{rd} bit position of gray code) = 1 (3rd bit position of resultant binary code)

1 (3^{rd} bit position of binary code) + 1 (4^{th} bit position of gray code) = 10. After discarding the carry 1, it is 0 (4^{th} bit position of resultant binary code)

0 (4^{th} bit position of binary code) + 1 (5^{th} bit position of gray code) = 1 (5^{th} bit position of resultant binary code)

Therefore the resultant binary code is 11101. The process is graphically explained below:





CHECK YOUR PROGRESS

Q.3: Convert the following number from binary to gray code no.

i) 1111 ii) 1010

Q.4: Convert the following numbers from gray codes to binary.

i) 11011 ii) 100111



4.7 LET US SUM UP

- **Binary-Coded Decimal (BCD)** code, a way to express each of the decimal digits with its binary equivalent. It is a four digit binary code for representing from 0 to 9 in decimal form.
- ASCII is the only standard that is used to convert all characters and special symbols. ASCII has 128 character codes (from 0 to 127) and symbols represented by a 7-bit binary code.
- Another code conversion standard is the EBCDIC **(Extended Binary-Coded Decimal Interchange Code).** 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. Here, the first 4 bits are known as zone bits and remaining 4 bits represent digit values.

- The un-weighted code called **gray code** is not used in arithmetic operations. It is basically used in conversion procedure like analog to digital convertion.
- Gray code is sometimes referred to as reflected binary, because the first eight values compare with those of the last 8 values, but in reverse order.



4.8 ANSWER TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: a) 4; b) 8; c) 7; d) American Standard Code for Information Interchange; e) ASCII; f) EBCDIC

Ans. to Q. No. 2: i) Solution:

0010 0011		23
+00010101		+15
00111000		38

ii) Solution:

1001		9
+0100		+4
1001	Invalid BCD number (>9)	+13
+0110	Add 6 (i.e., 0110)	
0001 0011	A valid BCD number which is 13 in decimal	

Ans. to Q. No. 3: i) 1000; ii) 1111

Ans. to Q. No. 4: i) 10010; ii) 111010



4.9 FURTHER READING

1) Morris, M. M. (1987): *Digital Logic and Computer Design*.
2) Ram, B. (2000): *Computer Fundamentals: Architecture and Organization*; New Age International.
3) Sinha, P. K., & Sinha, P. (2010): *Computer Fundamentals* (Vol. 4); BPB publications.



4.10 MODEL QUESTIONS

Q.1: Why is code conversion needed? Describe any two code conversion methods.

Q.2: Add the following BCD numbers:

a) 100100 + 1100101

b) 1000011 + 0101011

c) 101010 + 110011

d) 11000 + 1011000

e) 101001 + 11001

Q.3: What is ASCII? Why is it used?

Q.4: Write the full form of BCD, ASCII and EBCDIC.

Q.5: What is Gray Code? Where is it used?

Q.6: Explain the following conversion methods:

a) Gray code to Binary equivalent

b) Binary to Gray code

*** ***** ***

UNIT 5: BOOLEAN ALGEBRA

UNIT STRUCTURE

5.1 Learning Objectives

5.2 Introduction

5.3 Boolean Algebra

5.4 Properties of Boolean Algebra

5.5 De-Morgan's Theorem

5.6 Let Us Sum Up

5.7 Answers to Check Your Progress

5.8 Further Reading

5.9 Model Questions

5.1 LEARNING OBJECTIVES

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

- describe boolean algebra
- describe the properties of boolean algebra
- analyze De-Morgan's theorem
- list the boolean identities.

5.2 INTRODUCTION

In the previous units, we have come across different number systems, their arithmetic, different types of data representation and code coding techniques.

Now, in this unit you will be introduced to the most important part of digital electronics, i.e. boolean algebra. In this unit, at first, you will be introduced to boolean algebra and its properties. Finally, the unit will end with the discussion on De-Morgan's theorem.

5.3 BOOLEAN ALGEBRA

Boolean algebra can be considered fundamental in the development of digital electronics. It is a branch of algebra in which it deals with only two

variables, 1 and 0, which can also be termed as TRUE or FALSE respectively. It was first developed by George Boole in 1954. There are a few rules related to boolean algebra. These are as follows:

- Variables can have only one value, either 1 (HIGH or TRUE) or 0 (LOW of FALSE).
- If we consider a variable Y then its complement is represented by a over-bar (–). Thus, compliment of Y can be represented as $\overline{Y}$. Thus, if Y = 0 then $\overline{Y}$ =1 and vice versa.
- When OR operation is done between the variables then it is represented by '+' (plus) sign.
- When AND operation is done between the variables then it is represented by '.' (dot) sign.

5.4 PROPERTIES OF BOOLEAN ALGEBRA

The basic properties or laws of boolean algebra include commutative law, associative law and distributive law. The laws are same as in basic algebra. For illustration purpose two or three variables are considered, but it can be extended up to any numbers. Let us now start with the commutative law-

Commutative Law: For two arbitrary logic variables A and B, the commutative law for addition states that: A + B = B + A

Again, the commutative law for multiplication states that: AB = BA.

Associative Law: For any three arbitrary logic variables A, B and C the associative law for addition states that: A + (B+C) = (A+B) + C

For multiplication it can be stated as: A.(BC)=(AB).C

Distributive Law: For any three arbitrary logic variables A, B and C it can be stated as: A(B+C) = AB + AC

In addition to the above three laws, there are a few rules that that are important in terms of simplification of different boolean expressions. They are listed below-

Rule 1: A + 0 = A

If one input is fixed (in this case it is 0), and the other input is arbitrary (that means 0 or 1, it is represented as A); the output value of the

OR gate will depend on the arbitrary input. If A = 0, then output will be zero and if A = 1, output will be 1. So we can write in general that A + 0 = A (where A = 0 or 1)

Rule 2: A + 1 = 1

Similar to the rule 1 explanation if one input is fixed as 1 and other one is arbitrary (i.e. A = 0 or 1), the output will be 1.

Rule 3: A.0 = 0

If one input is 0, then whatever be the other input value, the output is zero. So, whether A = 0 or 1, the output remains zero.

Rule 4: A.1 = A

Here in this case one input value is fixed as 1, so the output will depend upon A. If A = 1, then output will be 1 and if A = 0 then output will be 0, it implies that, whatever be the value of A, the output will be same.

Rule 5: A + A = A

If we take A = 0, than 0 + 0 will be 0 and if A = 1, 1 + 1 = 1.This explains the rule.

Rule 6: $A + \overline{A} = 1$

Here in this case, $\overline{A}$ has the value compliment or inverted to the value assigned to A, so if A = 0 then $\overline{A}$ will be 1 and vice versa. Now 0 + 1 = 1 and 1 + 0 = 1.

Rule 7: A.A = A

Similar to rule 5, if A = 0, 0.0 = 0 and if A = 1, 1.1 = 1

Rule 8: $A.\ \overline{A} = 0$

Similar to rule 6, it is clear that 1.0 = 0 and 0.1 = 0, hence it explains the rule.

Rule 9: $\overline{\overline{A}} = A$

If we invert a variable twice, we again get the original value of the variable. For example, if A = 0 then $\overline{A} = 1$; and again if we invert $\overline{A}$, then we get $\overline{\overline{A}} = 1$, which is equal to A.

Rule 10: A + AB = A

To explain this law, let start from the left hand side:

A + AB = A.1 + AB (using rule 4)

= A(1+B) (distributive law)

= A.1 (as 1 + B = 1, using rule 2)

= A (using rule 4)

Rule 11: $A+\overline{A}B = A+B$

Here, also to prove it, we start from the left hand side-

$A+\overline{A}B$ = (A+AB) + $\overline{A}B$ (using rule 10)

= (A.A + A.B) + $\overline{A}B$ (using rule 7)

= A.A + A.B + 0 + $\overline{A}B$ (here we added one zero, which do not effect the value of the expression)

= A.A + A.B + A.$\overline{A}$ + $\overline{A}B$ (using rule 8 as A.$\overline{A}$ =0)

= (A+$\overline{A}$)(A+B) (factorisation process)

= 1.(A+B) (using rule 6)

= A + B (rule 4)

Using all these properties of boolean algebra, we can expand or reduce any boolean expression according to our need and requirements.



CHECK YOUR PROGRESS

Q.1: The commutative law of boolean addition states that A + B = A × B. (True/False)?

Q.2: Determine the values of A, B, C, and D that make the product term $\overline{A}B\overline{C}D$ equal to 1.

5.5 DE-MORGAN'S THEOREM

Augustus De Morgan, a British mathematician, proposed two theorems related to boolean algebra. These two theorems are extremely useful in boolean algebra.The theorems are discussed below-

Theorem 1: $\overline{(A+B)} = \overline{A}.\overline{B}$

Theorem 2: $\overline{(A.B)} = \overline{A}+\overline{B}$

To prove this let us look at the truth table below:

Input		Output			
A	B	$\overline{(A+B)}$	$\overline{A}.\overline{B}$	$\overline{(A.B)}$	$\overline{A}+\overline{B}$
0	0	1	1	1	1
0	1	0	0	1	1
1	0	0	0	1	1
1	1	0	0	0	0

If you look at the table, there are four output columns. If we look at the first column, the output is obtained after performing the operation $\overline{(A+B)}$. First if we perform '+' operation between A and B and invert the result, we get the value for $\overline{(A+B)}$. Similarly, if we invert both the inputs and perform AND operation we get the output for $\overline{A}.\overline{B}$, shown in the 2nd column of output. Similarly also we get the outputs for $\overline{(A.B)}$ and $\overline{A}+\overline{B}$. After getting outputs of all the columns of required truth table, now if we look at the output table, it clearly shows that: $\overline{(A+B)}$ = $\overline{A}.\overline{B}$ and $\overline{(A.B)}$ = $\overline{A}+\overline{B}$

Note: These two theorems can be extended to any number of variables/literals.



CHECK YOUR PROGRESS

Q.3: Which among the Boolean expressions stated below do not obey De-Morgan's theorem?

a) X + Y = X.Y b) X.Y = X + Y

c) X.Y = X + Y d) None of the above



5.6 LET US SUM UP

- Boolean algebra was first developed by George Boole in 1954.
- Boolean algebra can be considered fundamental in development of digital electronics. It is a branch of algebra in which it deals with only two variables, 1 and 0, which can also be termed as TRUE or FALSE.
- The basic properties or laws of boolean algebra include the commutative law, associative law and distributive law. They are same as in basic algebra.

- Augustus De Morgan, a British mathematician, proposed two theorems related to boolean algebra.
- Two De morgan's theorems which are extremely useful in boolean algebra.

  Theorem 1: $\overline{(A+B)} = \overline{A}.\overline{B}$

  Theorem 2: $\overline{(A.B)} = \overline{A}+\overline{B}$



5.7 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: False

Ans. to Q. No. 2: A = 0, B = 1, C = 0, D = 1

Ans. to Q. No. 3: (c) X.Y = X + Y



5.8 FURTHER READING

1) Mano, M. M. (2002): *Digital Design*; EBSCO Publishing, Inc.

2) Talukdar. P. H. (2004): *Digital Techniques*; N. L. Publications.



5.9 MODEL QUESTIONS

Q.1: Describe Boolean algebra and its properties.

Q.2: Prove that $\overline{A+B} = \overline{A}.\overline{B}$

Q.3: Define and describe De Morgan's Theorem.

Q.4: Describe the different rules of boolean algebra.

*** ***** ***

UNIT 6: LOGIC GATES

UNIT STRUCTURE

6.1 Learning Objectives

6.2 Introduction

6.3 Logic Gates

- 6.3.1 Truth Table
- 6.3.2 AND Gate
- 6.3.3 OR Gate
- 6.3.4 NOT Gate
- 6.3.5 NAND Gate
- 6.3.6 NOR Gate
- 6.3.7 XOR Gate
- 6.3.8 XNOR Gate

6.4 Conversion of Logic Gates

6.5 Let Us Sum Up

6.6 Answers to Check Your Progress

6.7 Further Reading

6.8 Model Questions

6.1 LEARNING OBJECTIVES

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

- describe logic gates
- draw differenet logic gates like AND, OR, NOT, NAND, NOR, XOR, XNOR
- represent the output of logic gates in truth table
- describe DeMorgan's theorem
- perform conversion of logic gates.

6.2 INTRODUCTION

In previous units, we have studied different number systems like Decimal, Binary, Hexadecimal and Octal and their conversion from one form to another.

This unit provides a description of logic gates and defines a few of the common logic gates found in simple digital circuits. Computers work on an electrical flow where a high voltage is considered as “1” and a low voltage is considered as “0”. Using these highs and lows, data are represented. Electronic circuits must be designed to manipulate these positive and negative pulses into meaningful logic.Logic gates are the building blocks of digital circuits.

Combinations of logic gates form circuits, designed with specific tasks in mind. For example, logic gates are combined to form circuits to add binary numbers (adders), set and reset bits of memory (flip-flops), multiplex multiple inputs, etc.

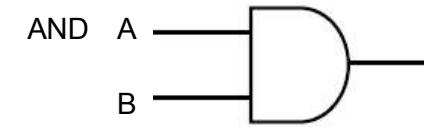
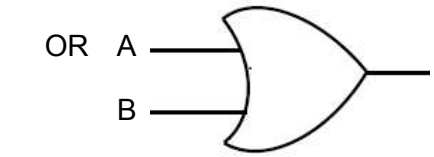
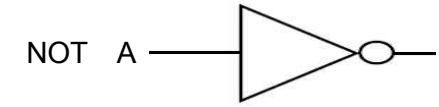
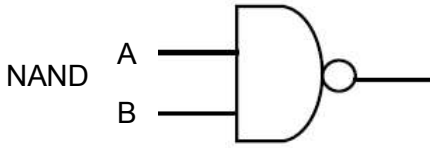
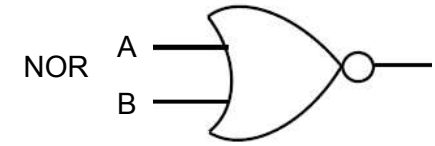
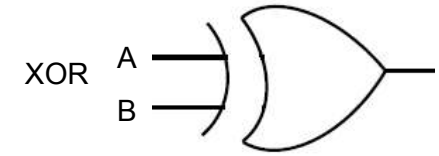
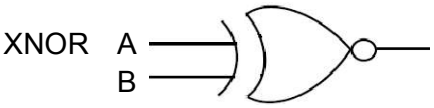
6.3 LOGIC GATES

A gate is an electronic circuit which produces a typical output signal depending on its input signal. The output signal of a gate is single boolean operation of its input signal. Gates are the basic logic elements. Any boolean function can be represented in the form of gates.

In general, we can represent gates in three ways. These are:

- Graphical symbols
- Algebraic notation
- Truth table

Different types of logic gates with their graphical symbol, algebraic notion and their truth tables are summarised below:

Gate Name	Symbol	Notation	Truth Table
AND		F = A.B or F = AB	A B A.B 0 0 0 0 1 0 1 0 0 1 1 1
OR		F = A + B	A B A+B 0 0 0 0 1 1 1 0 1 1 1 1
NOT		F = $\overline{A}$ or F = A′	A F 1 0 0 1
NAND		F = $\left(\overline{A.B}\right)$ or F = (AB)′	A B F 0 0 1 0 1 1 1 0 1 1 1 0
NOR		F = $\left(\overline{A+B}\right)$ or F = (A+B)’	A B F 0 0 1 0 1 0 1 0 0 1 1 0
XOR		F = A $\oplus$ B or F = A′B + AB′ or F = $\overline{A}$B + A$\overline{B}$	A B F 0 0 0 0 1 1 1 0 1 1 1 0
XNOR (Exclusive NOR)		F = A B or F = AB + A′B′	A B F 0 0 1 0 1 0 1 0 0 1 1 1

Note: Remember 1 OR 1 written as 1 + 1 = 1, but 1 + 1 (1 plus 1) in binary = 10

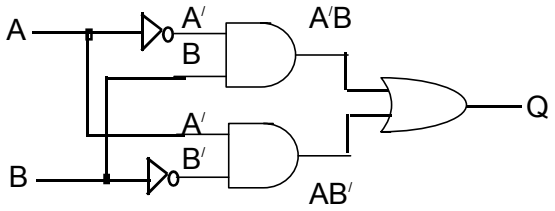
6.3.1 Truth Table

A truth table is used to show the behaviour of logic gates. It shows the output states for every possible combination of input states. The symbols 0 (false) and 1 (true) are usually used in truth tables. They show how the input(s) of a logic gate relate to its output(s).

A truth table for two input of an OR gate is shown below, but it can be extended to any number of inputs. The gate input(s) are shown in the left column(s) of the table with all the different possible input combinations. This is normally done by making the inputs count up in binary. The gate output(s) are shown in the right hand side column.

Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	1

Example: Draw the truth table for the following logic circuit and find the output expression Q.



Solution: The above logic circuit gives the following truth table and output expression ($A^{/}$.B)+(A.$B^{/}$).

A	B	$A^{/}$	$B^{/}$	$A^{/}$.B	A.$B^{/}$	$A^{/}$.B + A.$B^{/}$
0	0	1	1	0	0	0
0	1	1	0	1	0	1
1	0	0	1	0	1	1
1	1	0	0	0	0	0



EXCERCISE 6.1

Draw the truth table for a 3-input AND gate.

6.3.2 AND Gate

An AND gate has two or more inputs but it has only one output. An input signal applied to a gate has only two stable states, either 1

(HIGH) or 0 (LOW). There is no intermediate state. In case of a 2-input AND gate the output is 1 (high) only if both inputs are 1 (high), otherwise the output is 0 (low). The logical operation of an AND gate is called the AND operation(or AND function).

The logical AND operation can easily be explained with the help of two switches connected in series, Figure 6.2. The current will flow in the circuit only when both switches, A and B, are closed. A switch has two stable states, ON and OFF. The ON state is taken as logic 1, the OFF as logic 0. When both switches are ON, i.e., A and B both have value 1, the output is 1. When any one of the swithces is OFF, there is no output (no current in the circuit) and the output value is taken as 0.

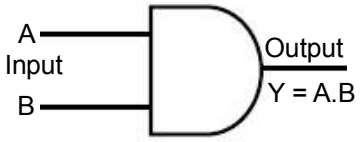


Fig. 6.1: AND Gate

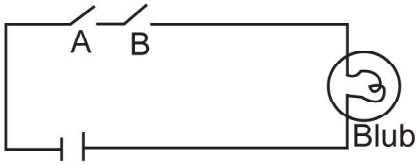


Fig. 6.2: A circuit containing two switches in series

Figure 6.1 shows standard graphical symbol for a two-input AND gate. It shows only two inputs, A and B. If there are more than two inputs, they will be shown on input side. Table 6.1 shows the truth table for a two-input AND gate. The output of an AND gate is 1 only when all the inputs are 1. If any one of the inputs is 0, the output will be 0. Therefore, the output of an AND gate is equal to the product of inputs i.e., the output Y = A.B. The symbol ^ is also used to denote AND operation. A ^ B denotes AND operations of A and B.

Table 6.1: Truth table for AND Gate

INPUTS		OUTPUT
A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

The output of an AND gate is Y = A.B. In this case A and B (inputs) are independent variables. Y (output) is a dependent variable. The table which shows the values of dependent variable for all possible values of independent variables is called truth table.

6.3.3 OR Gate

An OR gate is also known as INCLUSIVE-OR gate. It has two or more inputs, but only one output. The output of an OR gate will be high if atleast any one of the inputs is high. The output will be 0 only when all inputs are 0. This logical operation will be called OR operation (or OR function). Figure 6.3 shows a two input OR gate. Its truth table has been shown in Table 6.2. Its output is given by Y = A + B. The logical OR operation can easily be explained taking an example of two switches connected in parallel as shown in Figure 6.4. The current will flow in the circuit when either switch is in ON position (i.e., logic 1). The current will not flow at all when both switches are in OFF position (i.e. logic 0). The V (DOWN) symbol denotes OR operation. A V B denotes OR operation of A and B.

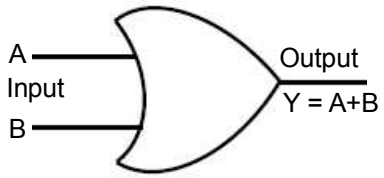


Fig. 6.3: Grapical Symbol OR Gate

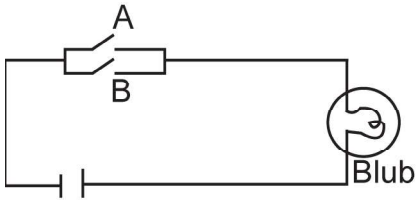


Fig. 6.4: A Circuit containing two switches in parallel

Table 6.2: Truth Table for OR Gate

INPUTS		OUTPUT
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

6.3.4 NOT Gate

A NOT gate has only one input, and only one output signal. It is also called INVERTER. Its output is the complement of the input signal. The output is 1 (HIGH) if the input is 0 (Low). The output is 0 when input is 1.

The output is given by $Y = A'$. NOT gate is used to invert the logic levels i.e. from LOW to HIGH or HIGH to LOW when required. Figure 6.5 shows standard symbol for a NOT gate.

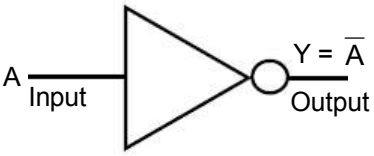


Fig. 6.5: Graphical Symbol NOR gate

A	Y = A
0	1
1	0

Table 6.3: Truth Table for NOT Gate



CHECK YOUR PROGRESS

Q.1: State whether the following statements are True (T) or False(F):

i) In case of a 2-input AND gate the output is 1 only if both inputs are 0 (low), otherwise the output is 0.

ii) The output of an OR gate will be high if atleast any one of the inputs is high.

iii) A NOT gate has only one input, and only one output signal.

iii) OR gate is also known as INCLUSIVE-OR gate.

Q.2: Represent AND, OR and NOT gate graphically.

Q.3: Draw the truth table for a 3-input OR gate.

6.3.5 NAND Gate

A NAND gate has two or more inputs but only one output Figure 6.6 shows a two-input NAND gate in two different ways. The NAND function is the complement of the AND function. An AND

gate can be combined with an INVERTER to form a NAND gate. The abbreviation NAND is the short form of NOT-AND. Its output is given by Y = $\overline{A.B}$. Table 6.4 shows the truth table for a 2-input NAND gate.

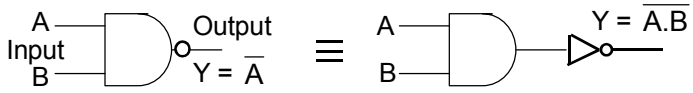


Fig. 6.6: Graphical Symbol NAND Gate

Table 6.4: Truth Table for NAND Gate

INPUTS		OUTPUT
A	B	Y = $\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

The NAND operations is sometimes called universal operations because other logic operation can be realized by using NAND gates. A universal operation is one that can be used to implement the three basic logic operations AND, OR and NOT. It is easier to fabricate NAND and NOR gates employing modern IC technology compared to AND and OR gates. NAND and NOR gates also consume less power. Therefore, combinational logic networks are realized by using only NAND gates. NAND gates are used as basic building blocks in fabricating a digital circuit.

6.3.6 NOR Gate

A NOR gate has two or more inputs but only one output. Figure 6.7 shows a two input NOR gate. The NOR function is the complement of OR function. An OR gate can be combined with an INVERTER to form a NOR gate as shown in the Figure 6.7. The abbreviation NOR is the short form of NOT-OR. Its output is given by Y = $\overline{A + B}$. Table 6.5 shows the truth table for a 2-input NOR gate.

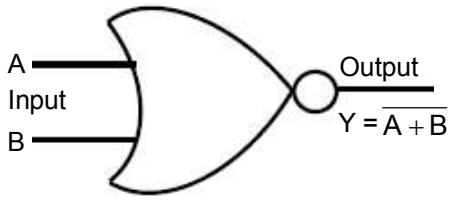


Fig. 6.7: NOR gate

The NOR operation is also a universal operation. Other logic gates such as AND, OR and NOT can be realized using NOR gates. Combinational logic network can be realized by using only NOR gates. NOR gates are also used as building blocks in fabricating a digital network.

Table 6.5: Truth Table for NOR Gate

INPUTS		OUTPUT
A	B	Y = A + B
0	0	1
0	1	0
1	0	0
1	1	0

6.3.7 XOR Gate

The output of Exclusive-OR (XOR) gate is high only when its inputs are different. Its output is low when both inputs A and B are same i.e., either both are high or both are low. Its output is given by

$Y = A \oplus B$

$= AB + AB$

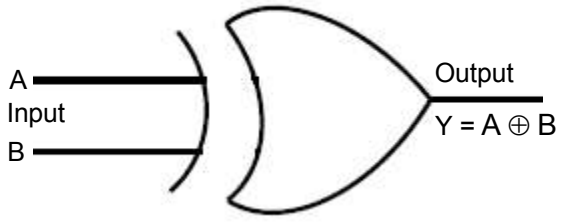


Fig. 6.8 (a): EXCLUSIVE-OR gate

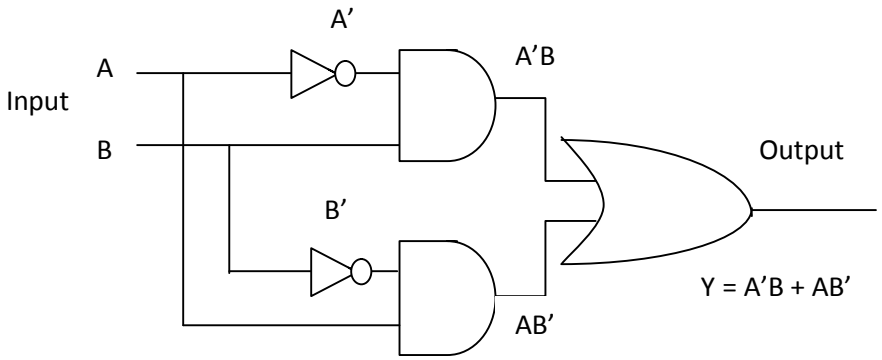


Fig. 6.8 (b): Logic diagram for a 2-input EXCLUSIVE-OR gate

The XOR operation is also called Modulo-2-Sum operation. Figure 6.8(a) and 6.8(b) shows the graphical symbol and logic diagram for a 2-input XOR gate. Table 6.6 shows its truth table.

Table 6.6: Truth table for XOR Gate

INPUTS		OUTPUT
A	B	$Y = A \oplus B = \overline{AB} + \overline{AB}$
0	0	0
0	1	1
1	0	1
1	1	0

An XOR operation can be distinguished from OR functions as follows: A OR B means A or B, or both, whereas A XOR B means A or B but not both.

6.3.8 XNOR Gate

The Exclusive-NOR (XNOR) operation is the complement of XOR operation. The output of XNOR is high only when the logic values of both inputs A and B are same i.e., either both are1 or both are 0. Its output is 0 when its inputs are different. Table 6.7 shows its truth table. The output is given by

$$Y = \overline{A \oplus B}$$
$$= AB + \overline{A}.\overline{B}$$

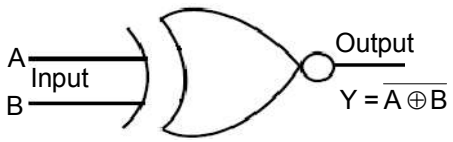


Fig. 6.9: Graphical symbol XNOR Gate

Figure 6.9 shows a two-input XNOR gate. The 74135 is an XOR/XNOR gate. It has three inputs A, B and C. If C input is low it operates as an XOR gate. If C input is high it opearates as XNOR gate.

Table 6.7: Truth Table XNOR Gate

INPUTS		OUTPUT
A	B	$Y = \overline{A \oplus B}$ $= AB + \overline{A}.\overline{B}$
0	0	1
0	1	0
1	0	0
1	1	1



CHECK YOUR PROGRESS

Q.4: Draw the logic diagram of the function:

a) D = A.(B+C) b) C = (A+B).(A.B)

Q.5: Choose the correct option:

i) Which operation is called Modulo-2-Sum operation:

a) AND b) OR c) XOR d) None of these

ii) For NAND operation if one input is 1 and the other input is 0, then the output will be:

a) 1 b) 0 c) low d) None of these

iii) If both inputs are HIGH in a 2-input NOR gate, then the output will be:

a) 1 b) low c) high d) None of these

iv) If both inputs are 0 in a 2-input NOR gate, then the output will be:

a) 0 b) 1 c) high d) None of these



EXCERCISE 6.2

Draw the truth table and graphic symbol for a 2-input XOR gate.

6.4 CONVERSION OF LOGIC GATES

A logic gate with more positive voltage (> = 4.5 volt) treating as '1' or HIGH and the other as the '0' level or LOW is called a positive logic gate. On the other hand when the more negative voltage represents the logic level '1' or HIGH and the more positive voltage represents the logic level '0' or LOW, the corresponding gate of logic system is called negative logic gate or negative logic system. Positive and negative logic gates are distinguished by attaching a small circle at the input of a gate for the negative logic. The rules for converting a positive logic to a corresponding negative logic are:

- to change AND graphic symbol to OR and vice-versa.
- to change NAND graphic symbol to NOR and vice-versa.
- to add circles at the input and output points whenever there are no circles.
- to remove circles at the input and output points whenever there are circles.



6.5 LET US SUM UP

- A logic gate is an electronic circuit that operates on one or more input signals to produce standard output signals. The logic gates are the building blocks of all electronic circuits in computer.
- To represent different logic gates, specific shapes are assigned. These are called graphical symbols.
- Logic gates are operated with binay numbers '0' and '1'. In analog terminology, '0' is termed as 'LOW' (ground) voltage and '1' is termed as 'HIGH' voltage (> = 4.5 volt.)
- A truth table is a good way to show the behaviour of logic gates. It shows the output states for every possible combination of input states.
- An AND gate is the physical realization of logical multiplication (AND) operation. It is an electronic circuit that generates an output signal of 1, only if all input signals are also 1.

- An OR gate is the physical realization of logical addition (OR) operation. It is an electronic circuit that generates an output signal of 1, if any of the input signals is also 1.
- An NOT gate is the physical realization of complementation operation. It is an electronic circuit that generates the reverse of the input signal as output signal. It is also known as an inverter because it inverts the input.
- A NAND gate is a complemented AND gate. That is, the output of NAND gate will be 1 if any one of the inputs is a 0, and will be 0 only when all inputs are 1.
- A NOR gate is a complemented OR gate. That is, the output of NOR gate will be 1only when all inputs are 0, and it will be 0 if any input is a 1.
- Exclusive-OR and Exclusive-NOR are two important logic components.
- DeMorgan's theorem states that the complement of a sum of variables is equal to the product of the complemented variables.

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

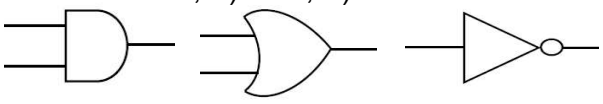
  and the complement of a product of variables is equal to the sum of the complemented variables.

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

6.6 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i)False; ii) True; iii) True; iii) True

Ans. to Q. No. 2:

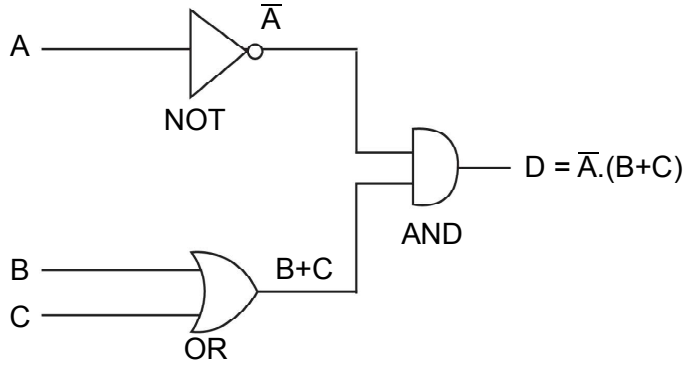


AND gate OR gate NOT gate

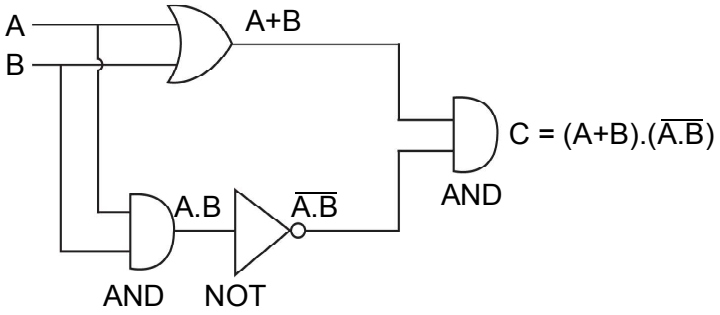
Ans. to Q. No. 3: Truth table for a 3-input OR gate is as follows:

INPUTS			OUTPUT
A	B	C	Y = A + B + C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Ans. to Q. No. 4: a)



b)



Ans. to Q. No. 5: i) (c) XOR; ii) (a) 1; iii) (b) low; iv) (b) 1



6.7 FURTHER READING

1) Jain, M., Pillai, V., & Jain, S. (2002): *Computer Organization and Architecture*; BPB Publications.

2) Morris, M. M. (1987): *Digital Logic and Computer Design*.

3) Ram, B. (2000): *Computer Fundamentals: Architecture and Organization*; New Age International.

4) Sinha, P. K., & Sinha, P. (2010): *Computer Fundamentals* (Vol. 4); BPB publications.



6.8 MODEL QUESTIONS

Q.1: What are logic gates? Draw the block diagram of a 3-input AND gate. Give its truth table.

Q.2: Draw the graphical symbol and truth table for a 2-input XOR gate.

Q.3: Draw the logic diagram and graphic symbol of a 2-input Exclusive-OR gate.

Q.4: Define and describe DeMorgan's Theorem.

*** ***** ***

UNIT 7 : FLOATING POINT NUMBER REPRESENTATION

UNIT STRUCTURE

7.1 Learning Objectives

7.2 Introduction

7.3 Floating Point Number

7.4 Normalization of Floating Point Number

7.5 Overflow and Underflow

7.6 IEEE standard for Floating Point Representation

7.7 Floating-Point Arithmetic

- 7.7.1 Addition and Subtraction
- 7.7.2 Multiplication
- 7.7.3 Division

7.8 Let Us Sum Up

7.9 Answers to Check Your Progress

7.10 Further Reading

7.11 Model Questions

7.1 LEARNING OBJECTIVES

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

- define floating point number
- describe the method to normalize floating point number
- understand overflow and underflow
- know the accepted standard of floating point presentation
- become familiar with floating point arithmetic.

7.2 INTRODUCTION

In the previous units, we have learnt about the different types of code conversion methods and logic gates. Different types of logic gates have been discussed in detail in the previous units. In this unit, we describe floating point number representation. We will also discuss how floating point

number is normalized. Further, we shall describe what are overflow and underflow. The IEEE standard for floating point number representation shall also be described. We will also discuss floating point arithmetic in this unit.

7.3 FLOATING POINT NUMBER

The floating point number system is used to represent large fraction of numbers for scientific and computational purposes using two segments namely mania and exponent. Numbers that are too large for standard integer representations or that has fractional components are usually represented in scientific notation using floating point numbers. Thus, 875,000,000,000,000 can be represented as 8.75×10^{14} and 0.0000000000000875 can be represented as 8.75×10^{-14}. Here the decimal point is shifted to a convenient location and the exponent of 10 is used to keep track of that decimal point. This allows a range of very large and very small numbers to be represented with only a few digits. This same approach can be used for binary numbers.

The floating point number system, based on scientific notation, is capable of representing very large (1.33×10^88) and very small (1.33×10^-88) numbers. A floating-point number can be represented in the form $\pm F \times B^{\pm E}$

This number can be stored in a binary word with three fields as shown below:

- Sign (plus or minus)
- Fraction F (also called mantissa or significand)
- Exponent E

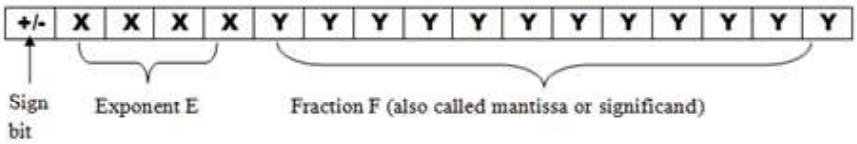


Figure 7.1: Floating point number representation

Figure 7.1 shows floating point number representation using mantissa and exponent. The base B is implicit and need not be stored because it is the same for all numbers. Decimal numbers use base 10 ($F\times10^E$) while binary numbers use base 2 ($F\times2^E$). The leftmost bit stores the sign of the number

('0' for positive numbers and '1' for negative numbers). The exponent value is stored in the next 'k' bits. The representation used is known as a biased representation. A fixed value, called bias, is subtracted from the exponent field to get the true exponent value. Typically, the bias is equals to, $(2^{k-1}-1)$ where 'k' is the number of bits in the binary exponent. The 8-bit exponent field yields the numbers from 0 to 255. With a bias of 127 i.e. (2^7-1), the true exponent values are in the range -127 to+128. The significand is composed of an implicit leading bit (before the radix point) and fractional bits (after the radix point). The leading bit for normalized numbers is 1 and for the denormalized numbers is 0. Modern computer architecture adopt IEEE 754 standard for representing floating-point numbers. The IEEE standard defines both 32-bit single -precision and 64-bit double-precision format with 8-bit and 11-bit exponents, respectively. The implied base is 2.

7.4 NORMALIZATION OF FLOATING POINT NUMBER

A normalized number is defined to have the most significant digit of the significand, a nonzero value. Any floating-point number can be expressed in many ways. For example, the following are equivalent, where the significand is expressed in binary form: 0.110×2^6, 110×2^3, 0.0110×2^7. To simplify operations on floating-point numbers, it is required that they be normalized.

For example,

- $+(1.23356789)_{10}\times10^1$: Normalized decimal floating point number
- $-(9.97865231)_{10}\times10^{12}$: Normalized decimal floating point number
- $(23.2)_{10}\times10^4$: Unnormalized decimal floating point number
- $(101.011)_2\times2^3$: Unnormalized binary floating point number
- $(1.01011)_2\times2^5$: Normalized binary floating point number

Therefore, for base 2 representation, a normalized number is one in which the most significant bit (MSB) of the significand is one. Thus, a normalized nonzero number is one in the form $\pm1.F\times2^{\pm E}$. Since the most significant bit is always one, it is not necessary to store this bit. This is an implicit bit hidden in the significand. Thus, the 23-bit significand field (in 32-

bit single-precision representation) is used to store a 24-bit value. Given a number that is not normalized, it may be normalized by shifting the radix point to the right of the most significant binary digit (bit 1) and adjusting the exponent accordingly.

Example 7.1: Represent each of the following decimal number using the 8-bit floating-point format with 3 bits for the mantissa and 4 bits for the exponent.

a) $(5.5)_{10}$

b) $-(96)_{10}$

Solution: a) $(5.5)_{10}$

Steps : 1.Convert the decimal number to binary: $(101.1)_2=(1.011)_2\times2^2$

2. Store the binary number in the following binary word with three fields:

Sign (1 bit)	**Exponent** (4 bits)	**Significand** (3 bits)

Sign bit is equals to 0 since the number is positive. The MSB of the number $(1.011)_2\times2^2$ will not occupy a bit position because it is always a 1. Therefore, the significand is the fractional number 011. Here, exponent field is of 4 bits. So, bias is equal to 7 (2^3-1). Add 7 to the exponent to get the true exponent value. True exponent value (biased exponent) is $2+7=(9)_{10}=(1001)_2$. Therefore, the complete floating point number is:

0	**1001**	**011**

b) $-(96)_{10}$

Steps: 1.Convert the decimal number to binary: $(-1100000)_2= (1.100000)_2\times2^6$

2. Store the binary number in the following binary word with three fields:

Sign (1 bit)	**Exponent** (4 bits)	**Significand** (3 bits)

Sign bit is equals to 1 since the number is negative. The MSB of the number $(1.100000)_2\times2^6$ will not occupy a bit position because it is always a 1. Therefore, the significand is the fractional number 100000. The required three significand bits are the first three bits i.e. 100. Here, significand needs to be rounded off to the nearest eight bits to fit in the allowed range of the

given representation. Here, exponent field is of 4 bits. So, bias is equal to 7 (2^3-1). Add 7 to the exponent to get the true exponent value. True exponent value (biased exponent) is 6+7=$(13)_{10}$=$(1101)_2$. Therefore, the complete floating point number is :

1	1101	100

Example 7. 2: Consider the 8-bit floating-point representation with 3 bits for the mantissa and 4 bits for the excess-7 (bias value is 7) exponent. What decimal number does the bit pattern 11101100 represent?

Solution:

1. The number is negative, since the sign bit is 1.
2. The exponent bits represent $(1101)_2 = (13)_{10}$. This is 7 more than the actual exponent, and so the actual exponent must be 13-7=6. Significand is 100. Thus, in binary scientific notation, we have - $(1.100)_2 \times 2^6$.
3. Convert this to binary number: - $(1.100)_2 \times 2^6$= - $(1100000)_2$.
4. Convert the binary number into decimal number: $(1100000)_2$= - $(96)_{10}$.
5. Therefore, the resultant decimal number is - $(96)_{10}$.

The 8-bit floating-point format can represent a wide range of both small numbers and large numbers. The smallest possible positive number that can be represented in 8-bit floating point format with 4 bit exponent and 3 bit significand is :

0 (sign bit)	0000 (Exponent bits)	000 (significand bits)

This gives 00000000 which represents,

$(1.000)_2 \times 2^0$-7= $(1.000)_2 \times 2^{-7}$. The decimal equivalent of this number is (0.0078)10.

The largest possible positive number that can be represented in this 8-bit floating point format is :

0 (sign bit)	1111 (Exponent bits)	111 (significand bits)

This gives 01111111 which represents,

$(1.111)_2 \times 2^{15-7}$= $(1.111)_2 \times 2^8$. The decimal equivalent of this number is $(480)_{10}$.

Thus, the 8-bit floating-point format (with 4 exponent bits and 3 significand

bits) can represent positive numbers from about $(0.0078)_{10}$ to $(480)_{10}$. In contrast, the 8-bit 2's-complement representation can only represent positive numbers in the range from 1 to 127. But the floating-point representation cannot represent all the numbers in this range, since 8 bits can represent only $2^8(256)$ distinct values, and there are infinitely many real numbers in the range to represent (even within a small range of 0.0 to 0.1). If the number of bits in the exponent is increased, the range of expressible numbers will get expanded. But because only a fixed number of different values can be expressed, the density of those numbers will be reduced and therefore the precision. The only way to increase both range and precision is to use more bits. Thus, most computers offer 32-bit single-precision numbers and 64 bit double-precision numbers.



CHECK YOUR PROGRESS

Q 1. The decimal numbers represented in the computer are called as floating point numbers, as the decimal point floats through the number.

(a) True (c) cannot say anything

(b) False (d) none of the above

Q 2. If the decimal point is placed to the right of the first significant digit, then the number is called as

(a) Orthogonal (c) Determinate

(b) Normalized (d) none of the above

Q 3._____ constitute the representation of the floating number.

(a) Sign (c) Significant digits

(b) Scale factor (d) All of the above

Q 4. The sign followed by the string of digits is called as _____

(a) Significant (c) Mantissa

(b) Determinant (d) Exponent

7.5 OVERFLOW AND UNDERFLOW

An AND gate 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. Figure 6.3 in unit 6 shows graphic symbols used to represent an AND gate.

Overflow occurs when an arithmetic operation results in a magnitude greater than the one that can be expressed in the given floating point notation. Underflow occurs if the number is not zero but too small to be represented in the given floating point representation. For example, Figure 7.2 shows the range of numbers that can be represented in a 32-bit word.

- Negative numbers can be represented between-$(2\text{-}2^{-23}) \times 2^{128}$ and -2^{-127}.
- Positive numbers can be represented between 2^{-127} and $(2\text{-}2^{-23}) \times 2^{128}$.

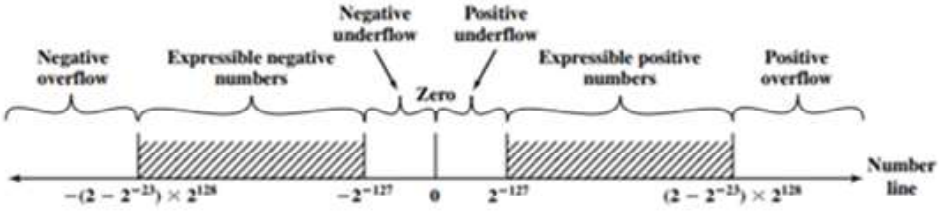


Figure 7.2: Expressible Numbers in Typical 32-Bit Formats

So, in this case overflow will occur if the magnitude of the result of an arithmetic operation is greater than the number that can be expressed with an exponent of 128 e.g. $2^{120} \times 2^{100} = 2^{220}$. Underflow will occur if the fractional magnitude is too small e.g. $2^{-120} \times 2^{-100} = 2^{-220}$.

7.6 IEEE STANDARD FOR FLOATING POINT REPRESENTATION

Floating point numbers are an important data type used in almost all computer hardware. The most important floating-point representation is defined in IEEE Standard 754, was adopted in 1985 by all computer manufacturers. This standard defines both a 32-bit single precision format and a 64-bit double precision format for representing a floating point number. The implied base is 2.

32-bits Double Precision IEEE 754 Standard

In 32-bit single-precision floating-point representation:

- The most significant bit is the sign bit (S), with 0 for positive numbers and 1 for negative numbers.

- The following 8 bits represent exponent (E).
- The remaining 23 bits represents fraction (F).

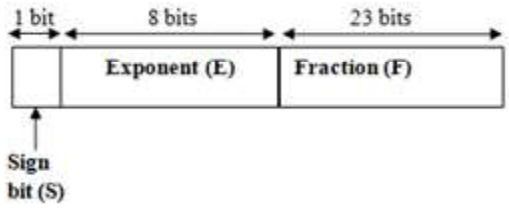


Figure 7.3: IEEE 754 representation of 32 bits floating point number

Figure 7.3 shows a typical 32-bit floating-point format. Exponent (E) can be both positive and negative numbers. So, instead of using a separate sign bit for the exponent, the standard uses a biased representation. Here, since the exponent field is of 8 bits, it can have numbers from 0 to 255. A bias value of 127 (2^7-1) is added to the true exponent to be stored in the exponent field. The exponent is biased, so that the range of exponents is from -126 to +127. So for exponent values in the range of 1 to 254, the value N = $\pm(1.F)\times 2^{E-127}$ or $(-1)^s \times (1.F)\times 2^{E-127}$ is the normalized nonzero floating-point number.

To increase the precision of the significand, the IEEE 754 standard uses a normalized significand. The MSB of the significand is always one and it is not necessary to store this bit. This is an implicit bit hidden in the significand. Thus in the single precision IEEE Standard, the significand field is 24 bits long of which 23 bits of the significand is stored in the memory and the MSB is an implied 1. The extra bit increases the number of significant digits in the significand.

For exponent (E) =0, the numbers are in the de-normalized form. Unlike normalized form in the de-normalized form an implicit leading 0 is used for the significand and the actual exponent is always -126. For E=0, the value N = $(-1)^s (0.F)\times 2^{-126}$. Hence, de-normalized form can be used to represent the number zero with E=0 and F=0. An exponent of all ones (i.e. E=255) together with a fraction (significand) of zero represents positive or negative infinity, depending on the sign bit. Magnitude of numbers that can be represented in this 32-bits single precision format is approximately

in the range of 1.8×10^{-38} to 3.40×10^{38}. An exponent of all ones (E=255) together with a nonzero fraction gives the value NaN, which means Not a Number. When an operation is performed by a computer on a pair of operands, the result may not be mathematically defined. For example, if zero is divided by zero, the result is indeterminate. Such a result is called Not a Number (NaN) in the IEEE Standard.

Example 7.3 : Convert the decimal number $(0.75)_{10}$ to a 32-bits single precision floating point binary number.

Solution: Given decimal number is $(0.75)_{10}$

Convert the decimal number to binary:

$(0.11)_2 = (1.1)_2 \times 2^{-1}$

Sign bit is equals to 0 since the number is positive. The MSB of the number $(1.1)_2 \times 2^{-1}$ will not occupy a bit position because it is always a 1 in normalized number. Therefore, the significand is the fractional 23-bits binary number 10000000000000000000000 and the biased exponent is:

$-1+127= (126)_{10} = (01111110)_2$

Therefore, the complete floating point number is

0	01111110	10000000000000000000000

Example 7.4 : Determine the decimal value of the following floating point binary number:

1 10010001 10001110001000000000000

Solution:

Step 1. The number is negative, since the sign bit is 1.

Step 2. The exponent bits represent $(10010001)_2 = (145)_{10}$. This is 127 more than the actual exponent, and so the actual exponent must be 145-127=18. Significand is 10001110001000000000000.

Step 3. The general approach to determine the value of a normalised floating point number is expressed by the formula:

$(-1)^S \times (1.F) \times 2^{E-127}$

$(-1)^1 \times (1.10001110001) \times 2^{145-127} = (-1)^1 \times (1.10001110001) \times 2^{18}$

Step 4. Convert this to binary number: $- (1.1001110001)_2 \times 2^{18} = -$

$(1100011100010000000)_2$.

Step. 5. Convert the binary number into decimal number: $-(1100011100010000000)_2 = -(407680)_{10}$.

64-bits Double Precision IEEE 754 Standard

In 64-bit double-precision floating-point representation:

- The most significant bit is the sign bit (S), with 0 for positive numbers and 1 for negative numbers.
- The following 11 bits represent exponent (E).
- The remaining 52 bits represents fraction (F).

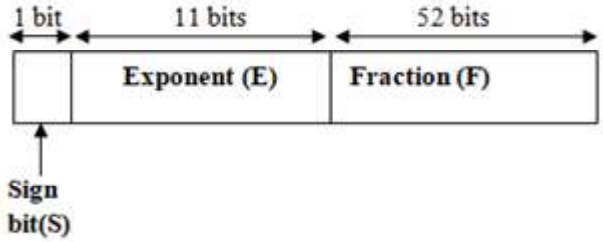


Figure 7.4: IEEE 754 representation of 64 bits floating point number

Figure 7.4 shows a typical 64-bit floating-point format. Similar to 32-bits representation, in this format also Exponent (E) can be both positive and negative numbers. So, instead of using a separate sign bit for the exponent, the standard uses a biased representation. Here, since the exponent field is of 11 bits, it can have numbers from 0 to 2048. A bias value of 1023 (2^{10}-1) is added to the true exponent to be stored in the exponent field. The exponent is biased, so that the range of exponents is from -1022 to +1023. So for exponent values in the range of 1 to 2046, the value N = $\pm(1.F)\times 2^{E-1023}$ or $(-1)^S\times(1.F)\times 2^{E-1023}$ is the normalized nonzero floating-point number. Here, the significand field is 53 bits long of which 52 bits of the significand is stored in the memory and the MSB is an implied 1. The extra bit increases the number of significant digits in the significand.

For exponent (E) =0, the numbers are in the de-normalized form. Unlike normalized form in the de-normalized form an implicit leading 0 is used for the fraction and the actual exponent is always -1022. For E=0, the

value N = $(-1)^S \times (0.F) \times 2^{-1022}$. Hence, de-normalized form can be used to represent the number zero with E=0 and F=0. An exponent of all ones (i.e. E=2047) together with a fraction of zero represents positive or negative infinity, depending on the sign bit. Magnitude of numbers that can be represented in this 64-bits double precision format is approximately in the range of 2.23×10^{-308} to 1.8×10^{308}. An exponent of all ones (E=2047) together with a nonzero fraction gives the value NaN, which means Not a Number.

7.7 FLOATING-POINT ARITHMETIC

For addition and subtraction, both the operands need to have the same exponent value. So, shifting of the radix point on one of the operands may be required to achieve proper alignment. Table 7.1 summarizes the basic operations for floating-point arithmetic. A floating-point operation may produce one of the following conditions:

(a) If a positive exponent exceeds the maximum possible exponent value, Exponent overflow occurs.

(b) If a negative exponent is less than the minimum possible exponent value then exponent underflow occurs. This means that the number is too small to be represented, and it may be expressed as zero.

(c) In the process of aligning the significands, digits may get shifted to the right end of the significand and some form of rounding operation may require. Significand underflow occurs in this situation.

(d) The addition of two significands of the same sign may result in a carry out of the most significant bit. This can be fixed by realignment and significand overflow may occur.

Table 7.1: Arithmetic Operations in floating point numbers

Floating Point Numbers	Arithmetic Operations
$X=X_s \times B^{Xe}$ $Y=Y_s \times B^{Ye}$ Where, X_s and Y_s are the significand of X and Y respectively X_e and Y_e are the exponent of X and Y respectively B= base	$X+Y=(X_s \times B^{X_e-Y_e}+Y_s)\times B^{Y_e}$ if $X_e \le Y_e$
	$X-Y=(X_s \times B^{X_e-Y_e}-Y_s)\times B^{Y_e}$ if $X_e \le Y_e$
	$X \times Y=(X_s \times Y_s)\times B^{X_e+Y_e}$
	$\frac{X}{Y}=\left(\frac{X_s}{Y_s}\right)\times B^{X_e-Y_e}$

7.7.1 Addition and Subtraction

Addition and subtraction are identical operation except for a sign change. So, in subtraction operation the sign of the subtrahend is changed.

There are four basic steps of the algorithm for addition and subtraction:

Step 1. Check for zero: If either operand is 0, the other operand is expressed as the result.

Step 2. Align the significands: The alignment is achieved by repeatedly shifting the magnitude portion of the significand of the smaller number to the right by 1 digit and incrementing the exponent until the two exponents are equal. If this process results in a 0 value for the significand, then the other number is expressed as the result.

Step 3. Add or subtract the significands: The two significands are added together, taking into account their signs.

Step 4. Normalization of result: If result of the operation is not in normalized form, normalization is done by left shifting the significand until MSB is nonzero and decrementing its exponent value or by right shifting the result and incrementing its exponent

value. It may cause exponent underflow if exponent value is smaller than minimum exponent allowed. Exponent overflow may occur if its value is larger than maximum exponent allowed in the representation. Finally, the result must be rounded off.

Here, negative significands are first converted to 2's complement form and then the addition is performed. The result is converted back to sign-magnitude form. When adding numbers of opposite sign, cancellation may occur, resulting in an arbitrarily small sum or even zero if the numbers are equal in magnitude. Normalization in this case may require shifting by the total number of bits in the significand results in a large loss of accuracy.

Example 7.5 : Convert the decimal number $(-0.75)_{10}$ to a 64-bits double precision floating point binary number.

Solution: Given decimal number is $(-0.75)_{10}$

Converting the decimal number to binary:

$(-0.11)_2 = -(1.1)_2 \times 2^{-1}$

Sign bit is equals to 1 since the number is negative. The MSB of the number $-(1.1)_2 \times 2^{-1}$ will not occupy a bit position because it is always a 1 in normalized number. Therefore, the significand is the fractional 52-bits binary number 1000 0000 0000 0000 0000 000.....0 and the biased exponent is:

$-1+1023 = (1022)_{10} = (01111111110)_2$

Therefore, the complete floating point number is

1	**01111111110**	**10000000000000000000000.....0**



CHECK YOUR PROGRESS

Q. 5. In IEEE 32-bit representations, the mantissa of the fraction is said to occupy _____ bits.

(a) 24 (b) 23

(c) 20 (d) 16

Q. 6.In double precision format the size of the mantissa is ___ bits.

(a)32 (b) 52

(c) 64 (d) 72

Q. 7. A machine stores floating point numbers in 7-bit word. The first bit is used for the sign of the number, the next three for the biased exponent and the next three for the magnitude of the mantissa. The number $(0010110)_2$ represented in base-10 is

(a) 0.375 (b) 0.875

(c) 1.5 (d) 3.5

Q. 8. A machine stores floating point numbers in 7-bit words. The first bit is stored for the sign of the number, the next three for the biased exponent and the next three for the magnitude of the mantissa. You are asked to represent 33.35 in the above word. The error you will get in this case would be

(a) Underflow (b) Overflow

(c) NaN (d) No error will be registered.

Example 7.6: Convert the following decimal number into IEEE 754 single precision format. Perform floating-point addition. Show the results in normalized form X = $(2345.125)_{10}$ and Y= $(0\ .75)_{10}$.

Solution: 32- bit single precision representation of X and Y is obtained using the method discussed in section 7.4.1.

X = $(2345.125)_{10}$ is represented as:

0 (sign bit)	**10001010 (Exponent bits)**	**00100101001001000000000 (significand bits)**

Y= $(0\ .75)_{10}$ is represented as:

0 (sign bit)	**01111110 (Exponent bits)**	**10000000000000000000 (significand bits)**

Addition operation is done using the formula shown in Table 7.1, i.e.

$$X+Y=(X_s\times B^{X_e-Y_e}+Y_s)\times B^{Y_e}$$

Steps:

Step 1. Here, X = $(2345.125)_{10}$= $(1.00100101001001)_2\times 2^{11}$ and Y= $(0\ .75)_{10}$ = $(1.1)_2\times 2^{-1}$

Xe= 10001010, Xs= 00100101001001000000000, Ye= 01111110

and

Ys= 10000000000000000000

So since Ye $\leq$ Xe , Y is a smaller number. Shifting the significand of **Y** until its exponent matches the exponent of X :

$Ys \times 2^{11}$ = 0.00000000000110000000000

Step 2. Adding the significand of X and Y:

Xs + Ys× 2^{11} = 1.00100101001001000000000 + 0.00000000000110000000000

= 1. 00100101001111000000000

Step 3. So, the result of addition is already a nonzero normalized number.

Step 4. Therefore, X+Y is equal to :

0	10001010	00100101001111000000000
(sign bit)	**(Exponent bits)**	**(significand bits)**

7.7.2 Multiplication

Floating-point multiplication algorithm is as follows:

Step 1. If either operand is zero, result of the operation is expressed as zero. The next step is to add the exponents.

Step 2. If the exponents are stored in biased form, the exponent sum would have doubled the bias value. Thus, the bias value must be subtracted from the sum. The result could be either an exponent overflow or underflow, which is marked by ending the algorithm.

Step 3. If the exponent of the product is within the proper range, the next step is to multiply the significands, taking into account their signs. The multiplication is performed in the same way as for integers.

Step 4. The result of the multiplication will be double the length of the multiplier and multiplicand. The extra bits will be truncated.

Step 5. The result is then normalized and rounded off like in addition and subtraction operations. The normalization could result in exponent underflow.

Example 7. 7: Convert the following decimal number into IEEE 754 single precision format. Perform floating-point multiplication. Show the results in normalized form X = $(-18)_{10}$ and Y= $(9.5)_{10}$.

Solution: 32- bit single precision representation of X and Y is obtained using the method discussed in section 7.4.1.

X = $(-18)_{10}$ is represented as:

1 (sign bit)	**10000011 (Exponent bits)**	**00100000000000000000000 (significand bits)**

Y= $(9.5)_{10}$ is represented as:

0 (sign bit)	**10000010 (Exponent bits)**	**00110000000000000000000 (significand bits)**

Multiplication operation is done using the formula shown in Table 7.1, i.e. $X \times Y = (X_s \times Y_s) \times B^{X_e + Y_e}$

Steps:

Step 1. Here, X = $(-18)_{10}$= $(-1.0010)_2 \times 2^4$ and Y= $(9.5)_{10}$ = $(1.0011)_2 \times 2^3$

Xe=10000011, Xs= 00100000000000000000000,

Ye= 10000010 and

Ys= 00110000000000000000000

Sign of X is negative and Y is positive, so sign of the product will be negative.

Step 2. Multiply the significand of X and Y: The product of the 24 bits significands will be of 48 bits with two bits to the left of the binary point: (01).0101011000000….000000

So, truncate the product to 24 bits: (1).01010110000000000000000

Step 3. Compute exponent of the result:

Xe + Ye - 12710 = 1000 0011 + 1000 0010 - 0111111 = 1000 0110

Step 4. So, the result of multiplication is already a nonzero normalized number.

Step 5. Therefore, X×Y is equal to :

1 (sign bit)	10000110 **(Exponent bits)**	01010101100000000000000 **(significand bits)**

7.7.3 Division

Floating-point division algorithm is as follows:

Step 1. If the divisor is zero, the result is set to infinity. A dividend of zero results in zero. If both divisor and dividend is zero, (i.e. 0/ 0) the result is NaN.

Step 2. The divisor exponent is subtracted from the dividend exponent to remove the bias, which must be added back in the exponent of the result.

Step 3. Check result exponent for overflow or underflow. Overflow occurs if result exponent is larger than maximum exponent allowed and underflow occurs if result exponent is smaller than minimum exponent allowed in the given representation.

Step 4. Divide the significand. The result of division will be double the length of the divisor and dividend. The extra bits will be truncated.

Step 5. The result is then normalized and rounded off.



7.8 LET US SUM UP

- The floating point number system is used to represent large fraction numbers for scientific and computational purposes using two segments namely manias and exponent.
- A normalized number is defined to have the most significant digit of the significand a nonzero value.
- Overflow occurs when an arithmetic operation results in a magnitude greater than the one that can be expressed in the given floating point notation.

- Underflow occurs if the number is not zero but too small to be represented in the given floating point representation.
- Floating point numbers are an important data type used in almost all computer hardware.
- The most important floating-point representation is defined in IEEE Standard 754, was adopted in1985 by all computer manufacturers.
- To increase the precision of the significand, the IEEE 754 Standard uses a normalized significand.
- Addition and subtraction are identical operation except for a sign change. So, in subtraction operation the sign of the subtrahend is changed.



7.9 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1 : (a)

Ans. to Q. No. 2 : (b)

Ans. to Q. No. 3 : (d)

Ans. to Q. No. 4 : (c)

Ans. to Q. No. 5 : (b)

Ans. to Q. No. 6 : (b)

Ans. to Q. No. 7 : (b)

Steps :

1. The number is positive, since the sign bit is 0.
2. The exponent bits represent $(010)_2 = (2)_{10}$. This is 3 more than the actual exponent, and so the actual exponent must be $2^{-3}= -1$. Significand is 110.Thus, in binary scientific notation, we have $(1.110)_2 \times 2^{-1}$.
3. Convert this to binary number: $(1.110)_2 \times 2^{-1} = (0.1110)_2$.
4. Convert the binary number into decimal number: $(0.1110)_2 = (0.875)_{10}$.
5. Therefore, the resultant decimal number is $(0.875)_{10}$.

Ans. to Q. No. 8 : (b)

Binary representation of decimal number $(33.35)_{10}$ is :

$(33.35)_{10} \approx (100001.01)_2$

Its binary scientific notation is $(1.0000101)_2 \times 2^5$.

The biased exponent has 3 bits, so the biased exponent varies from $(000)_2$ to $(111)_2$ i.e. from 0 to $(7)_{10}$. So the exponent can vary from -3 to

Hence the number $(33.35)_{10}$ which has an exponent of 5 would overflow.



7.10 FURTHER READING

1) Overtorn M., (2001), Numerical Computing with IEEE Floating Point Anithmetic.

2) Mano, Moorris. (2007), Digital Logic and Computer Design. Pearson Education.



7.11 MODEL QUESTIONS

Q.1: Convert the decimal number $(-2345.125)_{10}$ to a 32-bits single precision floating point binary number.

Q.2: Consider a 7-bit floating-point representation with 3 bits for the excess-3 exponent and 3 bits for the mantissa.

(a) How would 0.375 be represented in this 7-bit representation?

(b) What decimal value do 0110110 represent?

Q.3: Convert the decimal number $(1460.125)_{10}$ to a 64-bits double precision floating point binary number.

Q.4: The following numbers use the IEEE 32-bit floating-point format. What is the equivalent decimal value?

(a) 1 10000011 11000000000000000000000

(b) 1 00000000 00000000000000000000001

Q.5: Compute the largest and smallest positive numbers that can be represented in the 32-bit normalized form.

*** ***** ***

UNIT 8 : LOGIC FAMILY

UNIT STRUCTURE

8.1 Learning Objectives
8.2 Introduction
8.3 Logic Family
8.4 Resistor Transistor Logic (RTL)
8.5 Integrated Injection Logic (I2L)
8.6 Diode-Transistor Logic (DTL)
8.7 Emitter-Coupled Logic (ECL)
8.8 Transistor-Transistor Logic (TTL)
8.9 Tri State Logic
8.10 Metal Oxide Semiconductor Field-Effect Transistor (MOSFET)
8.11 Let Us Sum Up
8.12 Answers to Check Your Progress
8.13 Further Reading
8.14 Model Questions

8.1 LEARNING OBJECTIVES

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

- define logic family
- describe the different types of logic family
- describe RTL, I2L, DTL, ECL, TTL and MOSFET
- describe tri-state logic.

8.2 INTRODUCTION

In the previous units, we have learnt about logic gates and floating point number representation. Different types of logic gates and their conversion has been discussed in the previous units. We have also learnt to perform arithmetic functions on floating point numbers.

In this unit, we will learn about the concepts of logic family. Different types of logic families are covered in this unit like register transistor logic, integrated injection logic, diode transistor logic among others. Tri State logic and MOSFET technology is also covered in this unit.

8.3 LOGIC FAMILY

Logic family of digital integrated circuits (IC) devices is an assembly of logic gates built by using a number of different designs, methods, components and processes intended to generate binary output. Individual components are formed from various logic families, containing some interrelated basic logical functions. All these can be used as building-blocks to generate systems or as an interconnected structure to form more complex ICs. A logic family can also be referred to as a set of techniques that can be used to execute logic within VLSI ICs. Some of these logic families use static techniques to reduce the design complexities such as memories, central processors etc. Some other like domino logic, uses dynamic techniques to decrease the delay, size and power consumption. Prior to the extensive use of ICs, different vacuum-tubes and solid-state logic systems were used but these were not at all as interoperable and standardized as the IC devices.

ICs are basically used to perform low power circuit operation since they cannot handle large current and voltages. They are fabricated using different technologies like DTL, RTL, TTL, I2L, ECL etc. Some of the terms used in digital IC specifications are discussed below.

Threshold Voltage: It is defined as the voltage of the gate input that causes a change in the state from one logic level to the other at the output.

Propagation Delay: It defined as the average of the signal delay time of the output going from logic 1 to logic 0 and logic 0 to logic 1.

Power Dissipation: It is defined as the power required by a logic cycle to execute with 50% duty cycle at a particular frequency.

Fan-in: It is defined as the number of inputs, a logic gate can handle.

Fan-out: It is defined as the number of loads that the output of the logic gate can drive without damaging its regular operation.

Noise margin: The circuit's capacity to bear noise signals is known as the noise immunity, a quantitative measure of it is the noise margin.

Speed power product: It is defined as the product of gate power dissipation and propagation delay of the gate.

The main types of logic families are :-

A. Resistor Transistor Logic (RTL)

B. Integrated Injection Logic (I2L)

C. Diode-Transistor Logic (DTL)

D. Emitter-Coupled Logic (ECL)

E. Transistor-Transistor Logic (TTL)

F. Metal Oxide Semiconductor Field- Effect Transistor (MOSFET)

A comparison of the above families are given in Table 8.1.

Table 8.1: Comparing the parameters of different logic families

Parameter	RTL	I^2L	DTL	ECL	TTL	MOSFET
Basic Gate	NOR	NOR	NAND	OR-NOR	NAND	NAND
Fan out	5	Depends on Injector Current	8	25	20	20
Power dissipation	12mW	6 to 70 nW	8-12 mW	40-55 mW	10mW	0.2 – 10 mW
Noise immunity	Nominal	Poor	Good	Poor	Very good	Good
Propagation delay	12 nSec	25-30 nSec	30 nSec	1 nSec	10 nSec	300 nSec

8.4 Resistor Transistor Logic (RTL)

The resistor transistor logic (RTL) is a category of digital circuits that uses resistors at the input and bipolar junction transistors (BJTs) as the switching devices. These circuits were previously constructed with discrete components, but later on it became the first digital logic family to be produced, as the IC.

RTL inverter: A simple RTL inverter is shown in Fig 8.1. It consists of a common-emitter (CE) stage and a base resistor that is connected in between the input voltage source and the base. The base resistor's work is to increase the negligible transistor's input voltage range from around 0.7 V to about 3.5 V, by altering the input voltage to current. The job of the collector resistor is to change the collector current into voltage.

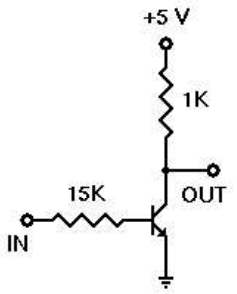


Figure 8.1: RTL inverter

RTL NOR gate: Using two or more base resistors (RC and R4) rather than one, it becomes a two input RTL NOR gate, shown in Fig 8.2. By applying the two arithmetic operations comparison and addition simultaneously, the logical operation OR is performed. The equivalent resistance of all the resistors connected to high state and that of the low state forms the two legs of a voltage divider circuit. The number of the inputs and the base resistances are chosen in such a way that only one high state is enough to make the base-emitter voltage greater than the threshold value and as such the transistor is in saturation mode. The transistor is cut-off, when all the input voltages are low. The pull down resistor (R1) biases the transistor to the proper on-off threshold value. Since the collector-emitter voltage of transistor (Q1) is taken as output, so it is inverted and when the inputs are low, it is high. Thus, it performs the logic function of a NOR gate.

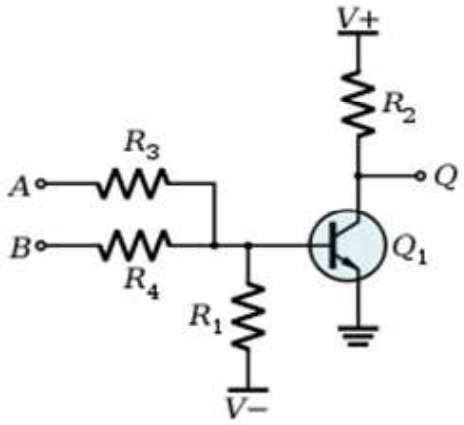


Figure 8. 2: RTL NOR gate with one transistor

8.5 INTEGRATED INJECTION LOGIC (I^2L)

The I^2L is a class of digital circuits that is built with several collector BJTs. In the initial stage, the I^2L had a speed that was almost equivalent to TTL, low power as CMOS, making it the best logic family for use in VLSI ICs. Though, the logic voltage levels are very close i.e. high value of 0.7V and low value of around 0.2V, it has a high noise immunity since it operates by current rather than by voltage. Sometimes, it is also termed as merged transistor logic. Figure 8.3 shows an I^2L circuit.

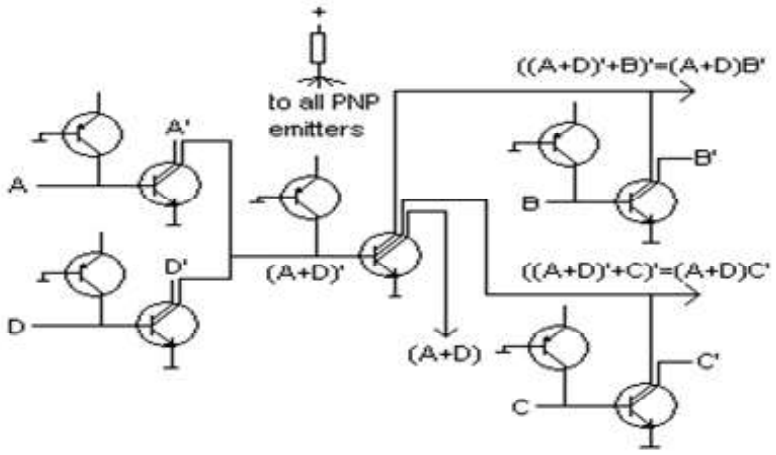


Figure 8.3: An I^2L logic circuit

The main part of an I^2L circuit is the common emitter open collector inverter. It consists of an NPN transistor with the base biased with a forward current and the emitter connected to ground. At the base, the input is applied either as a high-z floating condition or as a current sink. The output is taken from the collector. So, it is either high-z floating condition representing high logic level or a current sink showing the low logic level. If the bias current is grounded, the transistor is in off state and the collector is in floating condition. But, if the bias current is not connected to the ground, then the bias current will flow through the transistor to the emitter, turning the transistor on and this makes the collector to sink current i.e. low logic level. Since, the output can sink current instead of sourcing the current so it is better to connect the outputs of several inverters together, forming a wired AND gate (shown in Figure 8.4). It is a two-input NOR gate, when the outputs of two inverters are wired together.

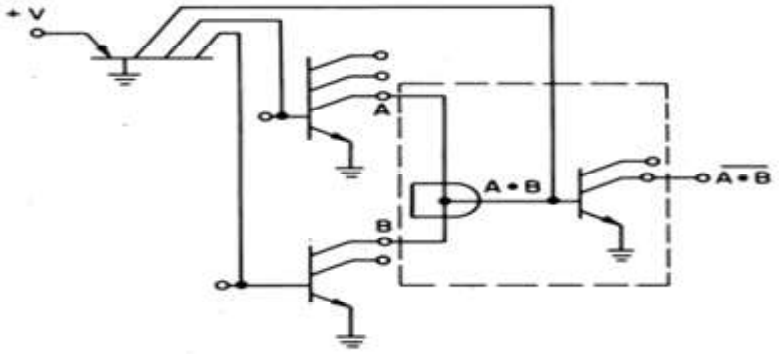


Figure 8.4: A wired AND gate followed by an inversion

8.6 DIODE-TRANSISTOR LOGIC (DTL)

It is a category of digital circuits that is the direct predecessor of transistor transistor logic. It is called so since the ANDing operation is carried out by a diode network and the transistor performs the amplifying part.

The DTL circuit, shown in Figure 8.5, consists of three stages. First is the input diode logic stage (R1, D1 and D2). Second is an intermediate level shifting part (RC and R4). Finally, an output common emitter amplifier stage (R2 and Q1). When both the inputs A and B are at logic 1, the diodes (D1 and D2) are reversed biased. The resistors (R1 and RC) will then provide sufficient current to turn on the transistor (Q1) along with the current required by the resistor (R4). A small voltage on the base (VBE = 0.3 V for Ge and 0.6 V for Si) of the transistor (Q1) will be generated. The Q1 transistor's collector current will then produce low value at the output (Q). If both the inputs or either of them is low, then one of the two input diodes conduct and pull the voltage less than around 2 volts. Resistors (RC and R4) behave as a voltage divider which makes transistor Q1's base voltage to be negative and turns it off. Transistor Q1's collector current will be around zero, so resistor (R2) will give high value at the output (Q).

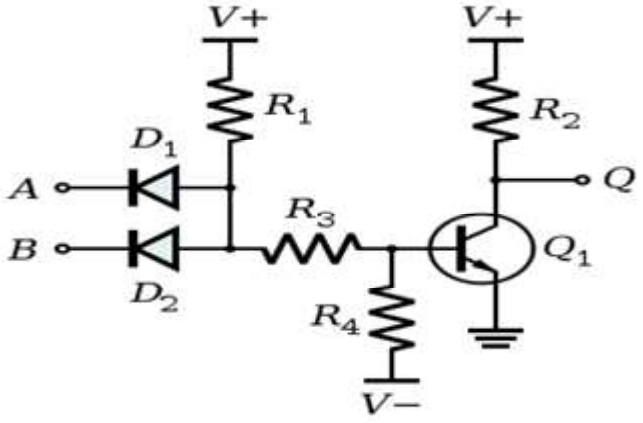


Figure 8.5: DTL logic

8.7 EMITTER-COUPLED LOGIC (ECL)

The ECL is a high speed IC BJT logic family. It uses a BJT differential amplifier having single ended input and low emitter current to avoid the

transistor going to its saturation state and also its slow turn-off characteristics. Since the current is steered in between the two legs of an emitter coupled pairs so it is sometimes called current steering logic (CSL), current switch emitter-follower (CSEF) logic or current mode logic (CML). In case of ECL, the transistors never reach the saturation stage, the input impedance is high, the output resistance is low and the input-output voltages have a small swing (around 0.8 V). So, the transistors change its states fast, fanout capability is high and gate delays are low.

The major limitation of an ECL is that each gate always draws the current, so it consumes more power than the other logic families. The equivalent of an ECL is made out of FETs known as the source-coupled logic (SCFL).

Figure 8.6 shows a NOR-OR gate using ECL logic. Here, A and B are the two inputs. When either one of the two inputs is high then the NOR output is low and the OR output is high. If both the inputs are low then the NOR is high and OR is low. Transistor Q3's base voltage is set at a level where there is a sufficient base current to make Q3 to conduct. Thus, Q3's collector brings the OR output low. Transistor Q3's emitter is high enough in comparison to Q2's base such that Q2 cannot conduct, so Q2's collector is grounded. This makes the NOR's output high. When either of the two inputs is high, then the subsequent transistor conducts brings Q1 or Q2's collector to be low causing the NOR's output to be low and OR's output to be high.

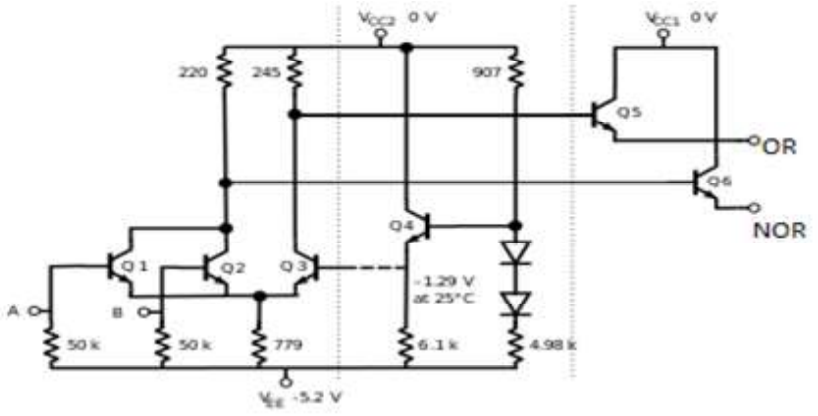


Figure 8.6: Two I/P ECL OR/NOR gate

8.8 TRANSISTOR-TRANSISTOR LOGIC (TTL)

It is a category of digital circuits built using BJTs and resistors. In this case, the transistors perform both the ANDing operation and the amplifying function. The TTL circuits were widely used in applications such as industrial controls, computers, instrumentation, consumer electronics etc. TTL inputs are the emitters of a multiple-emitter transistor. The TTL IC structure is functionally the same as that of the multiple transistors where the collector and bases are tied together. A common emitter amplifier buffers the output. To eliminate the problem with the high output resistance, totem-pole (push-pull) arrangement is used. It consists of the two n-p-n transistors (T_C and T_D), lifting diode (D) and the current-limiting resistor R_C, as shown in figure 8.7. It works by applying the current steering principle.

When both T_B and T_D are off, transistor T_C operates in an active region like a voltage follower producing a high output. When T_B is on, it activates T_D, driving low voltage to the output. The series combination of T_B's CE junction and T_D's BE junction is in parallel with the series of T_C's BE junction, T_D CE junction and D's anode cathode junction. The second series combination has the larger threshold voltage, so no current flows through it and transistor T_C is in off state. In between the transition, the resistor RC lowers the current that flows directly through the series connected transistors T_C, T_D and diode D. The efficiency of the gate may be increased by removing the pull-up and pull-down resistors from the output stage, without affecting the power consumption.

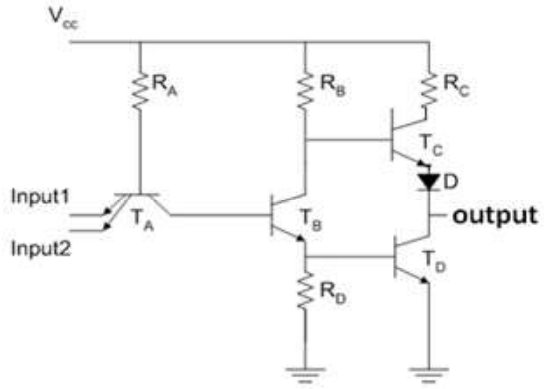


Figure 8.7: TTL logic inverter

TTL NAND

Figure 8.8 shows a NAND gate implemented using TTL logic. If both inputs are at logic 1, the two transistors are in reverse active mode. A current flows through the first resistor through the collector and base of these transistors and then on the right through the base of the transistor, thus saturating it and the output down is in low state. When both inputs are low, the simplest path to ground through the first resistor is through the base of the transistors. Thus the collector voltages are low, keeping that transistor off and bringing the output to logic 1 state. If only one of the inputs is at a logic 0, the input gives the easiest path to ground, keeping the transistor on the right switched off state.

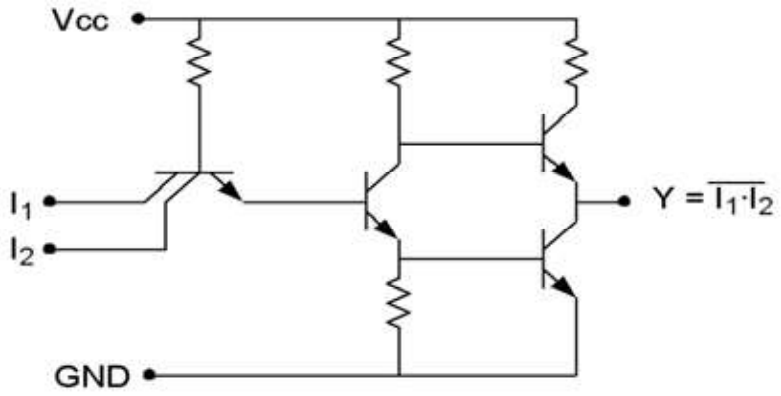


Figure 8.8: TTL NAND gate circuit

8.9 TRI STATE LOGIC

In digital electronics, tri state or three state logic allows an output port to assume a 0 and 1 logic levels in addition to high impedance state. This permits multiple circuits to distribute the same output lines. They are found applicable in many flip-flops, bus, drivers and registers. Some of the other uses are peripherals, computer memory and internal and external buses. Some of the devices are controlled by an active-low input called Output Enable (OE) which gives the information whether the outputs should be held in a logic1 state or a high-impedance state. It uses the advantage of high speed operation of wire ANDing of open collector configuration and totem-pole arrangement.

In the tri state logic, the high impedance state works as a selector which keeps away the circuits that are not being used. Keeping one device on a high-impedance state, prevents the design from a short circuit. Figure 8.9 shows the tri state logic. In this case, when B =1, the circuit operates as a normal inverter and when B=0,the circuit goes to high impedance state.

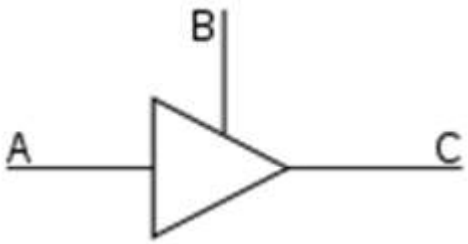


Figure 8.9: Tristate logic buffer

As shown in Fig 8.10, the low value at E forward biases the BE junction of Q1, and turns off Q2 by shunting the current in R1, making Q4 also off. The first diode shunts away from the base of Q3, at a low value of E, thus turning it off.

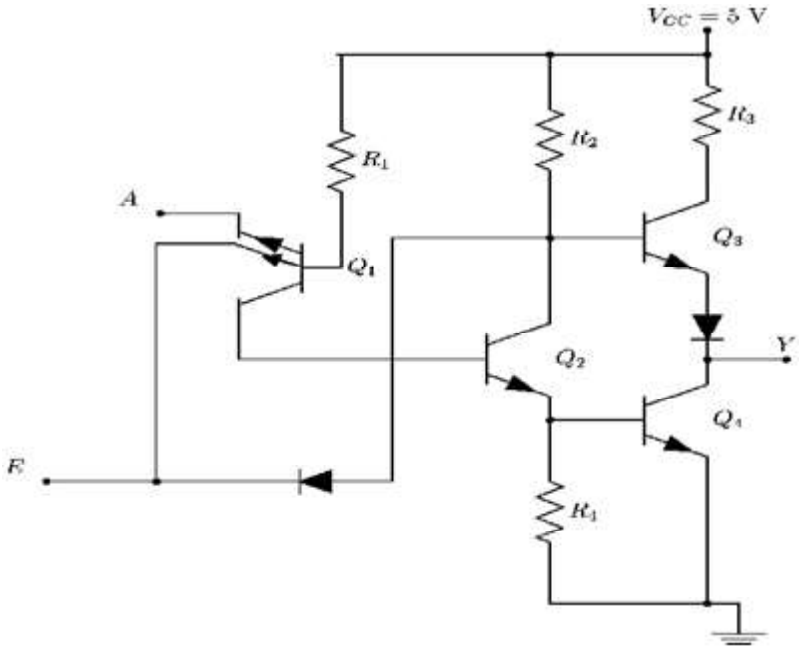


Figure 8.10: Tristate TTL inverter

8.10 METAL OXIDE SEMICONDUCTOR FIELD-EFFECT TRANSISTOR (MOSFET)

MOSFET is a FET that is usually fabricated by the controlled oxidation of Si. The conductivity of the device is determined by the insulated gate's voltage. This capacity to change the conductivity with respect to the amount of applied voltage can be used for switching or amplifying electronic signals. The main advantage of a MOSFET is that it needs almost zero input current to control the load current, when compared to BJTs. They can be classified under two types. One is the enhancement channel MOSFET and other is the depletion channel MOSFET. In case of the enhancement one, the conductivity of the device is increased by the voltage applied to the gate terminal increases. In the depletion mode MOSFET, the conductivity is decreased by applying voltage at the gate. The gate material used in MOSFET is a layer of polysilicon. Here, different dielectric materials are used in the oxide part. MOSFETs can be built with either p-type or n-type semiconductors or using complementary pairs of MOS transistors i.e. with CMOS logic. Figure 8.11 shows the p and n channel, enhancement and depletion type MOSFET.

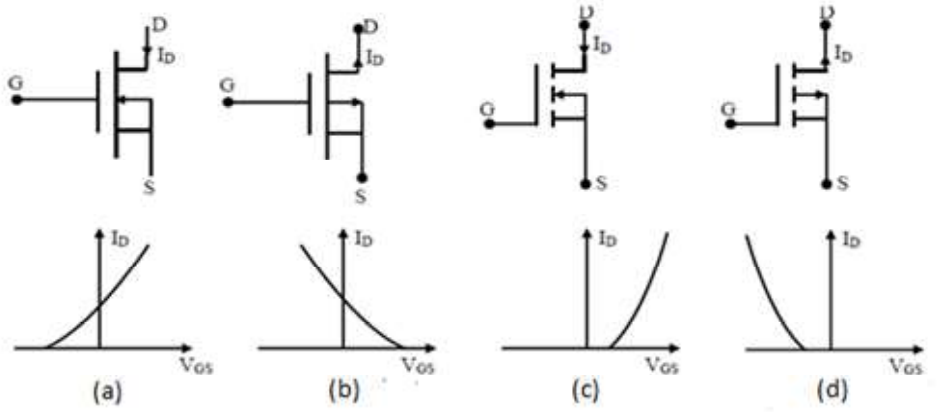


Figure 8.11: (a) n-channel depletion type MOSFET (b) p-channel depletion type MOSFET (c) n-channel enhancement type MOSFET (d) p-channel enhancement type MOSFET

Logic gates with MOSFET's:

Several logic gates such as AND, NAND, OR, and NOR using CMOS technology are discussed below. Figure 8.12 shows a two input NAND gate using CMOS logic. The transistors Q1 and Q3 are controlled by the input signal ($Input_A$), where the upper transistor is turned off and the lower transistor is turned on when the input is at logic 1. Q2 and Q4 are controlled by the same input signal ($Input_B$). The upper transistors Q1 and Q2 have their source and drain terminals parallelly connected while the transistors Q3 and Q4 are series-connected. The output will go high if either of the upper transistors saturates, and output will be low only if both lower transistors saturates.

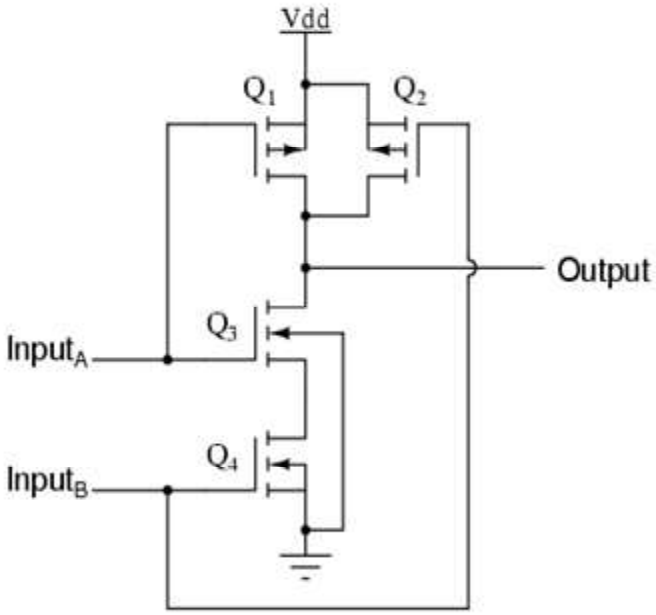


Figure 8.12: NAND gate using CMOS logic

As shown in Figure 8.13, the CMOS AND gate is created in the same way as the CMOS NAND gate but by only inverting the terminal at the output.

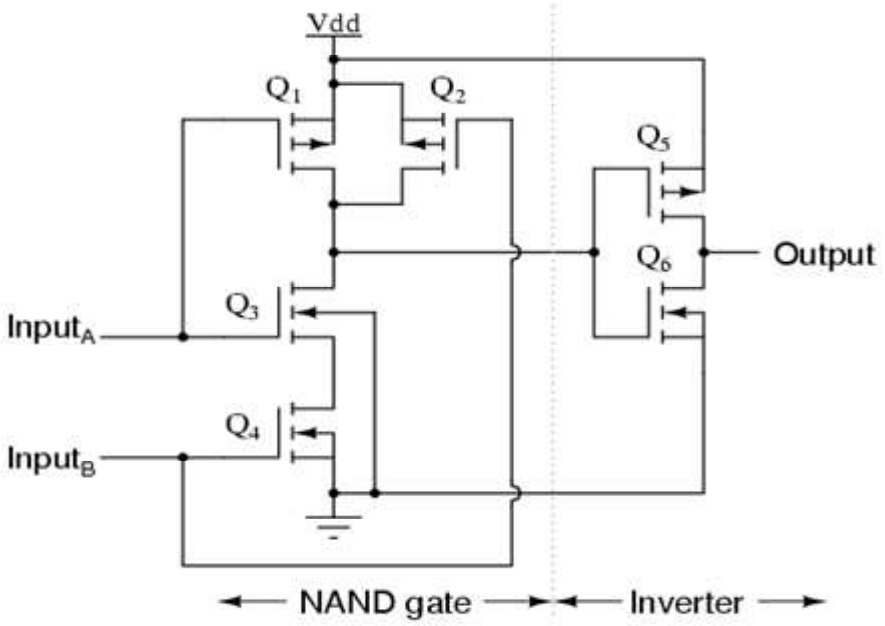


Figure 8.13: AND gate using CMOS logic

A CMOS NOR gate circuit also uses four MOSFETs alike that of the NAND gate, except the arrangements of the transistors. Here, in place of the two parallelly connected upper transistors and two series connected transistors lower ones, the NOR gate uses two two parallel-connected sinking transistors and two series-connected sourcing transistors, as shown in figure 8.14.

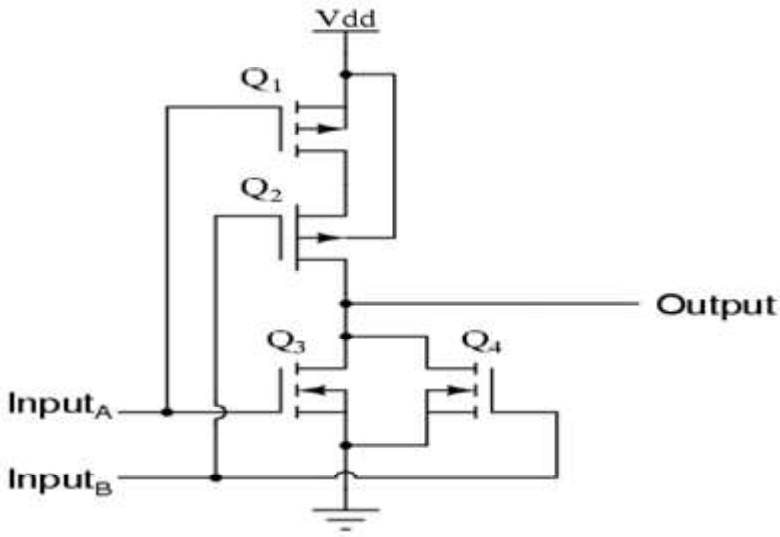


Figure 8.14: NOR gate using CMOS logic

The OR gate is made up from the basic NOR gate with an inverter stage on the output, as shown in Figure 8.15.

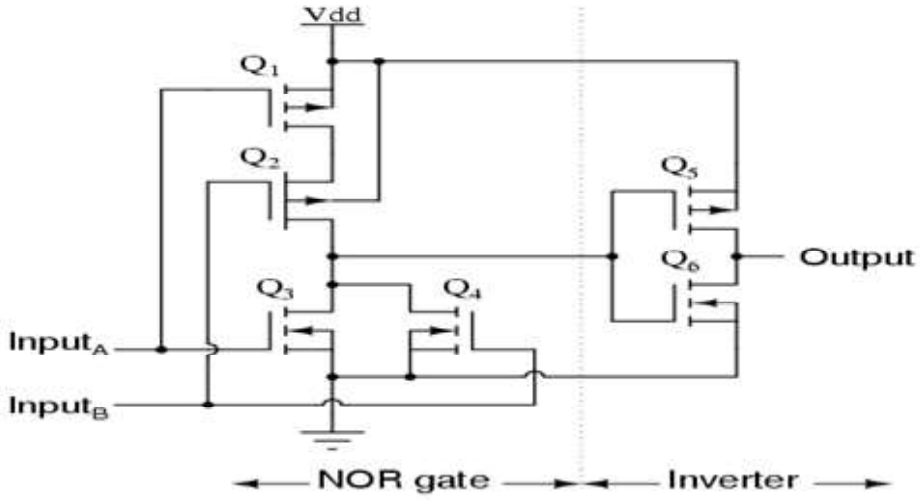


Figure 8.15: OR gate using CMOS logic



CHECK YOUR PROGRESS

Q.1. What is a totem pole output?

Q.2. State the advantages of CMOS logic.

Q.3. Write a note on tri-state gates.

Q.4.What is the significance of high impedance state in tri-state gates?

Q.5. Define the term fan out.

Q.6. How to increase fan-in of an RTL NOR gate?

Q.7. The DTL propagation delay is relatively _____.

Q.8. In an ECL, from where is the output taken?

Q.9. In ECL the fanout capability is _____.

Q.10. Give one disadvantage of ECL.



8.11 LET US SUM UP

- Logic family of digital integrated circuits (IC) devices is an assembly of logic gates built by using a number of different designs, methods, components and processes intended to generate binary output.

- ICs are basically used to perform low power circuit operation since they cannot handle large current and voltages.
- The RTL is a category of digital circuits that uses resistors at the input and bipolar junction transistors (BJTs) as the switching devices.
- The I2L is a class of digital circuits that is built with several collector BJTs. Initially it had a speed that was almost equivalent to TTL, still had a low power as CMOS, making it the best logic family for use in VLSI ICs.
- The DTL is a category of digital circuits that is the direct predecessor of transistor transistor logic.
- The ECL is a high speed IC BJT logic family. It uses a BJT differential amplifier having single ended input and low emitter current to avoid the transistor going to its saturation state and also its slow turn-off characteristics.
- TTL is a category of digital circuits built using BJTs and resistors.
- In digital electronics, tri state or three state logic allows an output port to assume a 0 and 1 logic levels in addition to high impedance state.
- The main advantage of a MOSFET is that it needs almost zero input current to control the load current, when compared to BJTs.



8.12 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q. No. 1 : Totem pole output is a standard output of a TTL gate. It is designed to reduce the propagation delay in the circuit and to provide enough output power for high fan-out.

Ans to Q. No. 2 : The advantages of CMOS logic are :–

- Consumes less power.
- Fan-out is more.
- Better noise margin
- Can be operated at high voltages, resulting in improved noise immunity.

Ans to Q. No. 3 : It is a digital circuit that exhibits three states. Two of the

states are signals equivalent to logic1 and logic 0. The third state is high impedance state.

Answer to Q. 4 : High impedance state of a tri state gate provides a special feature not available in other gates. A larger number of three state gate output can be connected with wires to form a common line without endangering loading effects.

Answer to Q. 5 : It is the maximum number of inputs which have same family that the gate can drive maintaining its output's normal operation.

Answer to Q. 6 : Can be increased by adding more output transistors

Answer to Q. 7 : Large

Answer to Q. 8 : Collector

Answer to Q. 9 : High

Answer to Q. 10 : It requires more power



8.13 FURTHER READING

1) Mano M. M. (2017), marris, 2007, Digital Logic and computer design. Pearson Education India.

2) Kumar, A. A., (2014), Fundamental of Digital Circuit, PHI Learning Pvt. Ltd.



8.14 MODEL QUESTIONS

Q.1: What are the different types of logic families?

Q.2: Compare the different parameters of the main logic families.

Q.3: Describe the working of the RTL taking into account its primary gate.

Q.4: How does the DTL family work? Describe using its primary gate.

Q.5: Write briefly about how the I^2L family works.

Q.6: Write short notes on TTL.

Q.7: What are the advantages and disadvantages of the MOSEFT family?

*** ***** ***

UNIT 9 : REDUCTION TECHNIQUES

UNIT STRUCTURE

9.1 Learning Objectives

9.2 Introduction

9.3 Boolean Expression

9.4 Canonical Form and Standard Form

9.5 Simplification of Boolean Expression

9.5.1 Algebraic Method

9.5.2 Karnaugh Map

9.6 Let Us Sum Up

9.7 Answers to Check Your Progress

9.8 Further Reading

9.9 Model Questions

9.1 LEARNING OBJECTIVES

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

- describe Canonical Form and Standard Form
- convert a function from standard form to canonical form
- describe how to simplify boolean expressions.

9.2 INTRODUCTION

In the previous unit, we have studied different logic gates, truth tables, their boolean expressions and symbols etc.

In this unit we will discuss boolean expressions with different variables which express the logic gates. There are various ways of expressing a boolean function. They are either in canonical form or in standard form. We will also discuss the conversion process of boolean functions from standard

form to canonical form. The main objective for converting a boolean function from standard form to canonical form is to simplify the function. Karnaugh Map, the technique through which a complex boolean function is simplified is discussed in this unit with various examples.

9.3 BOOLEAN EXPRESSION

Postulates form the assumptions from which different theorems, rules and properties are deduced. In Boolean algebra there are various postulates as discussed below:

1) Closure: closure with respect to operator "+" and "."
2) An identity element (a) with respect to "+" designated by 0 (x + 0 = x) and (b) with respect to "." designated by 1 (x.1 = x)
3) Commutative with respect to "+" and "."
   (a) x + y = y + x and (b) x.y = y.x
4) "." is distrutive over "+" and "+" is distributive over "."
   (a) x.(y+z) = x.y + x.z and (b) x + (y.z) = (x+y).(x+z)
5) If $x^{/}$ is the complement of x and x, $x^{/} \in B$, then
   (a) $x + x^{/} = 1$ and (b) $x.x^{/} = 0$
6) There exist at least two boolean elements x and y such that $x \neq y$.

Duality Principle: All the following postulates are in pairs and designated by two parts. One part may be obtained from the other if we interchange the binary operators ("+" to "." and vice versa) and the identity elements (1 to 0 and vice versa). This property of boolean algebra is called the duality principle.

Some important theorems and postulates of Boolean algebra are:

1)	(a) x + 0 = x	(b) x.1 = x
2)	(a) $x + x^{/} = 1$	(b) $x.x^{/} = 0$
3)	(a) x + x = x	(b) x.x = x
4)	(a) x + 1 = 1	(b) x.0 = 0
5)	(a) x + y = y + x	(b) x.y = y.x
6)	(a) x + (y+z) = (x+y) + z	(b) x(yz) = (xy)z
7)	(a) x(y+z) = xy + xz	(b) x + yz = (x+y)(x+z)
8)	(a) $(x+y)^{/} = x^{/}y^{/}$	(b) $(xy)^{/} = x^{/} + y^{/}$

9) (a) $x + xy = x$ (b) $x(x+y) = x$

10) $(x')' = x$

We can deduce the theorems using all the postulates. For example,

Theorem 3(a): $x + x = x$

$x + x = (x+x).1$ (since x.1=x)

$= (x+x).(x+x')$ (since $x + x' = 1$)

$= x + (xx')$ [since $x + yz = (x+y)(x+z)$]

$= x+0$ (since $xx' = 0$)

$= x$ (since x+0=x)

Theorem 4(a): $x + 1 = 1$

$x+1 = 1.(x+1)$ (since 1.x = x)

$= (x+x').(x+1)$ (since $x + x' = 1$)

$= x + (x'.1)$ [since $x + yz = (x+y)(x+z)$]

$= x+x'$ (since x.1=x)

$= 1$ (since $x+x'=1$)

Example 9.1: Find the dual of the following functions:

a) $f_1 = xy' + x'yz + y'z$ b) $f_2 = xyz' + x'z$

c) $f_3 = (a+b+c')(b'+c)$ d) $f_4 = ab'cd + bc'd$

Solution: a) Dual of function f_1 is $(x+y').(x'+y+z).(y'+z)$

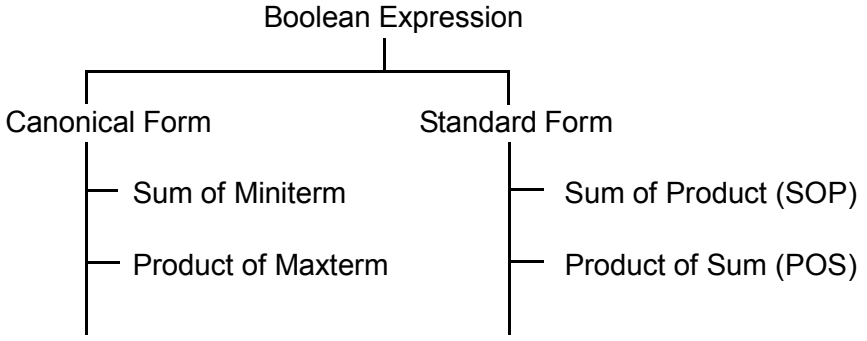
b) Dual of function f_2 is $(x+y+z').(x'+z)$

c) Dual of function f_3 is $abc' + b'c$

d) Dual of function f_4 is $(a+b'+c+d)(b+c'+d)$

9.4 CANONICAL FORM AND STANDARD FORM

A Boolean expression is represented either in standard form or in canonical form.



***a*) Canonical Form:** A function F is said to be in canonical form if all the variables in complemented or uncomplemented from exist in each term of the function F. For example:

$$F(x, y, z) = x'yz + xy'z + x'y'z$$

In the above example, all the variables x, y, and z either in complemented (say x') or uncomplemented (say x), are present in each term of the function F.

In canonical form, a boolean function is expressed either in sum of minterm or in product of maxterm. The two canonical forms can be obtained by reading a function from the truth table.

Minterm: In minterm, the operation between all the variables are AND operation. In reading a function from the truth table, for value 0 the variable is written in complemented form and for value 1 the variable is written in uncomplemented form. For example:

x	y
0	1

For the above table minterm = $x'y$

A binary variable may appear either in its complemented (say x') or uncomplemented (say x) form. If we consider two binary variables x and y combined with AND operation then we get four possible combinations. These are $x'y'$, $x'y$, xy' and xy. Each of the AND term is called a minterm or standard product.

Maxterm: In maxterm, the operation between all the variables are OR operation. In reading a function from the truth table, for value 0 the variable is written in uncomplemented form and for value 1 the variable is written in complemented form. For example:

x	y
0	1

For the above table maxterm = $x + y'$

If we consider two binary variables x and y combined with OR operation then we get four possible combinations. These are $x + y$, $x + y'$, $x' + y$ and $x' + y'$. Each of the OR term is called a maxterm or standard sum.

Table 9.1 : Minterm and maxterm for three variable x, y and z

			Minterms		Maxterms	
x	y	x	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	$x' yz'$	m_2	$x+y' +z$	M_2
0	1	1	$x' yz$	m_3	$x+y' +z'$	M_3
1	0	0	$xy' z'$	m_4	$x' +y+z$	M_4
1	0	1	$xy' z$	m_5	$x' +y+z'$	M_5
1	1	0	xyz'	m_6	$x' +y' +z$	M_6
1	1	1	xyz	m_7	$x' +y' +z$	M_7

Example 9.2 : Find the sum of minterm and the product of maxterm of the function f_1 from the following table:

x	y	x	Function f_1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Solution: For the sum of minterm, we will have to consider all 1's of functionf_1.

Therefore, the sum of minterm of function:

$$f_1 = x'yz' + xy'z' + xyz = m_2 + m_4 + m_7$$
$$= \sum(2,4,7)$$

For the product of maxterm, we will have to consider all 0's of function f_1.

Therefore, the product of maxterm of function:

$$f_1 = (x+y+z)(x+y+z')(x+y'+z')(x'+y+z')(x'+y'+z)$$
$$= M_0 M_1 M_3 M_5 M_6$$
$$= \Pi\ (0, 1, 3, 5, 6)$$

b) Standard Form: Standard form is another way of expressing a boolean function. A function that is in standard form may contain the terms that may have one, two or any number of variables. Like canonical form, there are two types of standard forms. They are **sum of products (SOP)** and **product of sum (POS).**

Sum of Product (SOP): It is a boolean expression containing AND terms called product terms, of one or two or more variables each. The sum denotes the ORing of these terms. For example:

$$f_1 = x'z + x'y$$

The function f_1 contains two product terms with three variables x, y and z. The first term ($x'z$) contains x and z, where y is absent and the second term ($x'y$) contains x and y, where the third variable z is absent.

Product of Sum (POS): It is a boolean expression containing OR terms called sum terms, of one or two or more variables each. The product denotes the ANDing of these terms. For example:

$$F_2 = (x'+z)(x+y')y$$

The function f_2 contains three sum terms with three variables x, y and z. The first term ($x'+z$) contains x and z, where y is absent, the second term ($x+y'$) contains x and y, where the third variable z is absent and the third term y where both x and z are absent.



CHECK YOUR PROGRESS

Q.1: Write examples of sum of minterm and product of maxterm.

Q.2: Find the sum of minterm and the product of maxterm of the function f_1 from the following table:

x	y	x	Function f_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0

1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Algorithm for converting Sum of Product (SOP) to sum of minterm:

i) Choose a term from the given expression

ii) If it is minterm then leave it else pick the missing variable in the non minterm say y.

iii) Multiply the non minterm by 1 that is $(y+y')$

iv) Expand the product

v) Repeat step i to iii until all the product terms contain all the variables

vi) If a term appears more than once, combine them in one, as $x + x = x$

Example 9.3: Express the following logical expressions in sum of minterm.

a) $f_1 = x + xy'$ b) $y = xy' + yz' + xz$

Solution:

a) The function f_1 has two variables x and y. Therefore,

$f_1 = x + xy'$

$= x.1 + xy'$ (since x.1 = x)

$= x (y+y') + xy'$ (since $y + y' = 1$)

$= xy + xy' + xy'$ [since x (y+z) = xy + xz]

$= xy + xy'$ (since x + x = x)

b) The function y has three variables x, y and z. Therefore,

$y = xy' + yz' + xz$

$= xy'.1 + yz'.1 + xz.1$ (since x.1 = x)

$= xy'.(z+z') + yz'.(x+x') + xz.(y+y')$ (since $x + x' = 1$)

$= xy'z + xy'z' + yz'x + yz'x' + xzy + xzy'$ (since x (y+z)= xy + xz)

$= xy'z + xy'z' + xyz' + x'yz' + xyz + xy'z$ (since xy = yx)

$= xy'z + xy'z' + xyz' + x'yz' + xyz$ (since x + x = x)

$= m_5 + m_4 + m_6 + m_2 + m_7$ (from table 7.1)

$= m_2 + m_4 + m_5 + m_6 + m_7$

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

Algorithm for converting product of sum (SOP) to product of maxterm:

i) Choose a term from the given expression

ii) If it is a non maxterm then identify the missing variable and express it in a form $x = (x+y)(x+y')$, when the variable y is assumed to be missing in the original expression, else pass to the next term and so on.

iii) Repeat step i to ii till the last term of the given expression.

iv) Combine the same terms into one as x.x=x and collect all maxterm as a product expression.

Example 9.4 : Express the following expressions in product of maxterm.

a) $f_1 = x(y+z)$ b) $y = (x+yz)(y+z'x)$

Solution:

a) The function f_1 has two terms x and (y+z) and three variables x, y and z. In the first term x, variables y and z are missing and in the second term (y+z), variable x is missing. Therefore, $f_1 = x(y+z)$

Now, consider the first term i.e. x.

$x = x + yy'$ [since $yy' = 0$ and $x + 0 = x$]

$= (x+y)(x+y')$ [distributive law]

Thus, we have two terms for the first term x, but z is missing in both the terms. Therefore,

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

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

Similarly, we can have: $(x+y')$ $= (x+y'+z)(x+y'+z')$

Thus, we have:

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

Now, consider the 2nd term of the function, (y+z).

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

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

Thus, $f_1 = (x+y+z)(x+y+z')(x+y'+z)(x+y'+z')(x'+y+z)$

$= (M_0 M_1 M_2 M_3 M_4)$

$= \prod (0, 1, 2, 3, 4)$

b) The function y has two terms (x+yz) and (y+z′x); and three variables x, y and z.

y = (x+yz)(y+z′x)

= (x+y)(x+z)(y+z′)(y+x) [since (x+yz) = (x+y)(x+z)]

= (x+y)(x+z)(y+z′)(x+y) (since x + y = y + x)

= (x+y)(x+z)(y+z′) (since x.x = x)

= {(x+y+z)(x+y+z′)}{(x+z+y)(x+z+y′)}{(y+z′+x)(y+z′+x′)}

[since (x+y)(x+y′) = x]

= {(x+y+z)(x+y+z′)}{(x+y+z)(x+y′+z)}{(x+y+z′)(x′+y+z′)}

(since x + y = y + x)

= (x+y+z)(x+y+z′)(x+y′+z)(x′+y+z′) (since x.x=x)

= $M_0 M_1 M_2 M_5$ (from table 7.1)

= $\prod$ (0, 1, 2, 5)



CHECK YOUR PROGRESS

Q.3: Express the following boolean expressions in sum of minterm:

a) y = b + b′c b) $f_1(a,b,c) = a'c + bc + ab'c$

Q.4: Express the following expressions in product of maxterm:

a) $f_1 = x(x'+y)$ b) y= (a+b′) (b+c′) (c+a′)

9.5 SIMPLIFICATION OF BOOLEAN EXPRESSION

Boolean expression represents the logic gates. Logic gates are nothing but the electronic circuits which includes diodes, transistors, resistors, capacitors etc. Therefore simple boolean expression means less complexity in circuits; results in efficient cost effective product. We will discuss two ways of simplifying a Boolean expression. They are **Algebraic Method** and **Karnaugh Map** method.

9.5.1 Algebraic Method

The primed or unprimed variables of a boolean algebra are called literals. As boolean functions are finally represented by logic

gates, less number of literals in a boolean function results in a circuit with less components, which is simple to understand and has less cost. The number of literals in a boolean function can be minimized by the algebraic method. Using different postulates and theorems one can minimize the number of literals of a boolean function.

Example 9.5 : Simplify the following boolean function using algebraic method.

a) $x + x'y$ b) $x + xy$

c) $x (x' + y)$ d) $x'y'z + x'yz + xy'$

Solution:

a) $x + x'y = (x + x')(x+y)$ [since a+b.c=(a+b).(a+c)]

$= 1.(x+y)$ (since $x + x' = 1$)

$= x + y$ (since $1.x = x$)

b) $x + xy = x.1 + xy$ (since $x.1=x$)

$= x.(1+y)$ [since a.(b+c) = a.b + a.c]

$= x.1$ (since $1 + x = x$)

$= x$ (since $x.1 = x$)

c) $x(x'+y) = xx' + xy$ [since a.(b+c) = a.b + a.c]

$= 0 + xy$ (since $x.x' = 0$)

$= xy$ (since $0 + x = x$)

d) $x'y'z + x'yz + xy' = x'z (y'+y) + xy'$ [since a.(b+c) = a.b + a.c]

$= x'z.1 + xy'$ (since $x+x'=1$)

$= x'z + xy'$ (since $x.1=x$)

In this example, we see that the function originally had three AND terms, and simplified into two terms with less number of literals.

9.5.2 Karnaugh Map

Simplification of Boolean expression through algebraic method is sometimes difficult as there is no specific rules for simplification. The map method is a straight forward process for minimizing Boolean functions. This method was first proposed by Veitch and modified by Karnaugh. Therefore this method is known as the Veitch diagram or the Karnaugh map or simply Map method.

Karnaugh maps are represented in blocks with 2^n number of squares, where n is the number of variables. Therefore for two variables, there are $2^2 = 4$ blocks for four minterms. For three variables, there are $2^3 = 8$ blocks for eight minterms etc.

Two variable Karnaugh maps: In two variable Karnaugh maps, there are $2^2 = 4$ blocks, each of which represent one minterm.

x \ y	0	1
0	0	1
1	2	3

Fig. 9.1(a): Karnaugh map with block numbers

x \ y	0	1
0	00	01
1	10	11

Fig. 9.1(b): Karnaugh map with values

x \ y	0	1
0	$x'y'$	xy'
1	xy'	xy

Fig. 9.1(c): Karnaugh map with variables

Three variable Karnaugh maps: In three variable Karnaugh maps, there are $2^3 = 8$ blocks, each of which represent one minterm.

x \ yz	00	01	11	10
0	0	1	3	2
1	4	5	7	6

Fig. 9.2(a): Karnaugh map with block numbers

x \ yz	00	01	11	10
0	000	001	011	010
1	100	101	111	110

Fig. 9.2(b): Karnaugh map with values

x \ yz	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
1	$xy'z'$	$xy'z$	xyz	xyz'

Fig. 9.2(c): Karnaugh map with variables

x \ yz	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

Fig. 9.2(d): Karnaugh map with mean terms

Note: Here, we can observe that mean terms are not entered serially in a row: it is as m_0, m_1, m_3, m_2. The reason is that they are entered in such a way that the next min term to be entered is adjacent to the previous one. Adjacency between two mean terms is defined as follows:

Two mean terms are said to be adjacent to each other when they differ in only one literal (i.e. the literal appears in one as normal or unprimed form and in the other in primed form). Thus, m_3 (011 or $x'yz$) appears after m_1 (001 or $x'y'z$) instead of m_2 (010 or $x'yz'$).

Four variable Karnaugh maps: In four variable Karnaugh maps, there are $2^4 = 16$ blocks, each of which represent one minterm.

wx \ yz	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Fig. 9.3(a): Karnaugh map with block numbers

wx \ yz	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Fig. 9.3(b): Karnaugh map with values

wx \ yz	00	01	11	10
00	w′x′y′z′	w′x′y′z	w′x′yz	w′x′yz′
01	w′xy′z′	w′xy′z	w′xyz	w′xyz′
11	wxy′z′	wxy′z	wxyz	wxyz′
10	wx′y′z′	wx′y′z	wx′yz	wx′yz′

Fig. 9.3(c): Karnaugh map with variables

wx \ yz	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

Fig. 9.3(d): Karnaugh map with mean terms

Plotting a function in K-map: The function to be simplified is first converted into cannonical form if it is not already in that form. Next the function is plotted in the map by putting a "1" in the corresponding mean terms available in the function.

Rules to minimize or solve: Boolean functions in canonical form can be minimized through karnaugh map method by combining all the ones present in the squares. There are some specific rules through which one can combine the ones. The rules are:

- All one's can be paired either in vertically or horizontally, but not diagonally.
- One can combine all ones that may form rectangle of 2^n cells or blocks of integer n. For example, we can combine either $2^1 = 2$ or $2^2 = 4$ or $2^3 = 8$ blocks in a 3 variable K-map.

- Preferences will be given to combine maximum number of blocks, i.e. if we combine 8 blocks, we should not combine two blocks of 4.
- Karnaugh map is considered as in spherical, therefore we can combine all corners of a map [example 7.4(c)].
- Sum of all mean terms equal to 1, i.e. if all the mean terms present in a function "f_1", we can write $f_1 = 1$. For example, $f_1(x,y,z) = (0, 1, 2, 3, 4, 5, 6, 7)$ so $f_1(x,y,z) = 1$.

Note : When we combine the adjacent mean terms in a map for simplifying the function, we eliminate those literals/variables which are changing (i.e. from primed to unprimed and vice versa).

Example 9.6 : Minimize the following expressions using Karnaugh map method.

a) $f_1 = xy + xy'$ b) $f_2 = x'yz + x'yz' + xy'z' + xy'z$

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

Solution:

a) $f_1 = xy + xy'$

The function f_1 has two variables x and y and the function is in canonical form. Since xy and xy' are equal to m_3 and m_2 respectively, therefore we can directly put ones in the specific blocks 3 and 2 of the map and combine as shown in figure.

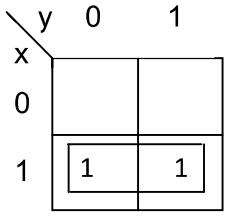


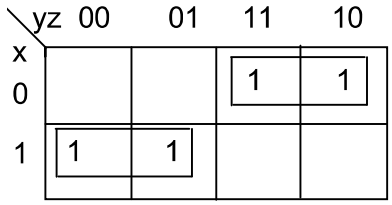
Fig. 9.4(a): Map for function f_1

$x\,\cancel{y}$
$x\,\cancel{y'}$
x

Therefore, $f_1 = x$

b) $f_2 = x'yz + x'yz' + xy'z' + xy'z$

The function f_2 has three variables x, y and z and all the terms are in canonical form. Since $x'yz$, $x'yz'$, $xy'z'$ and $xy'z$ are equal to m_3, m_2, m_4 and m_5 respectively, therefore we can directly put one's in the specific blocks 3, 2, 4 and 5 of the map and combine as shown in figure.



$x' y \bar{z}$ $x \bar{y}' \bar{z}'$

$x' y \bar{z}'$ $x y' \bar{z}$

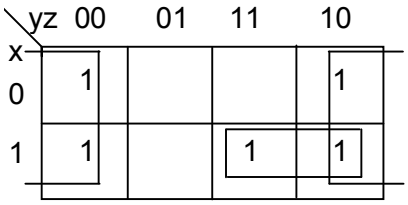
$x y'$ $x y'$

Fig. 9.4(b): Map for function f_2

Therefore, $f_2 = x'y + xy'$

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

The function y has occupied the blocks 0, 2, 4, 6 and 7 in the map, therefore we can directly put ones in the specific blocks as shown in the Figure 7.4(c).



$\bar{x}\bar{y}' z'$ $x y \bar{z}$

$\bar{x} y z'$ $x y \bar{z}'$

$\bar{x} y' z'$ $x y$

$\bar{x} y z'$

z'

Fig. 9.4 (c) : Map for function y

As Karnaugh map is considered to be in spherical, therefore we can combine all 4 corners of the map as in Figure 7.4(c). After combination, block m_7 left. As nearest block of m_7 is m_6, we combine both.

Therefore $y = z' + xy$

Note: We can combine a square/box more than once.

Some examples of possible combination of blocks of three variable karnaugh maps are shown below:

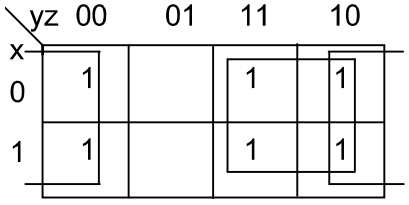


Fig. 9.4(d): $y = \sum(0,2,3,4,6,7)$

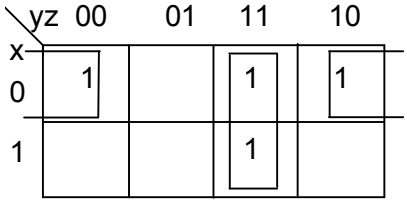


Fig. 9.4(e): $y = \sum(0,2,3,7)$

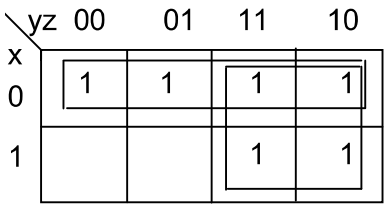


Fig. 9.4(f): $y = \sum(0,1,2,3,6,7)$

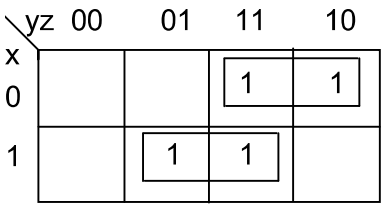


Fig. 9.4(g): $y = \sum(2,3,5,7)$

In the above examples, in Figure 7.4(d), we see that blocks m_2 and m_6 are used for both combinations, as the rule specifies that preferences will be given to combine maximum number of blocks. Combination of maximum number of blocks means most minimization of the expression, which is the main objective of reduction techniques. Similarly, in Figure 7.4(f) we have used m_2 and m_3 for both combinations. In Figure 7.4(e), the group of blocks m_2 and m_3 are not needed as all ones are already used. If we combine blocks m_2 and m_3 then such situation is called redundant. Similarly, in Figure 7.4(g) we cannot combine m_3 and m_7.

Example 9.7: Minimize the following expressions using Karnaugh map method.

a) $f_1 = w'x'y'z' + w'x'y'z + w'x'yz + w'x'yz' + w'xyz + w'xyz'$

b) $f_2 = \sum(0,2,3,6,7,8,10,11,14,15)$

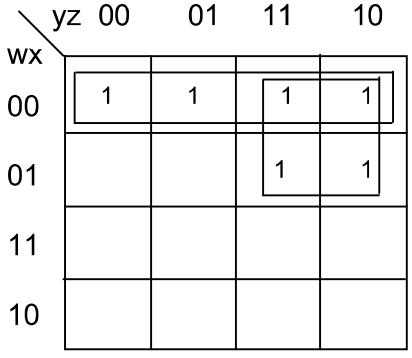
c) $Y = \sum(0, 2, 4, 6, 12, 13, 14)$

Solution:

a) $f_1 = w'x'y'z' + w'x'y'z + w'x'yz + w'x'yz' + w'xyz + w'xyz'$

$= \sum(0,1,2,3,6,7)$

The function f_1 has four variables w, x, y and z and all are in canonical form. Since $w'x'y'z'$, $w'x'y'z$, $w'x'yz$, $w'x'yz'$, $w'xyz$ and $w'xyz'$ are equal to m_0, m_1, m_2, m_3, m_6 and m_7 respectively, therefore we can directly put one's in the specific blocks 0, 1, 2, 3, 6 and 7 of the map and combine as shown in Figure 7.5(a).



$w'x'\cancel{y'z'}$ $w'\cancel{x}'yz$

$w'x'\cancel{y'z}$ $w'\cancel{x}'y\cancel{z}'$

$w'x'\cancel{y}z$ $w'\cancel{x}yz$

$w'x'\cancel{y}z'$ $w'\cancel{x}yz'$

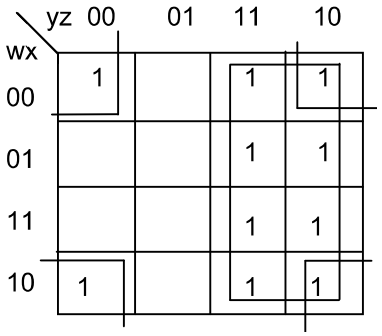
$w'x'$ $w'y$

Fig. 9.5(a): $f_1 = \sum(0,1,2,3,6,7)$

Therefore, $f_1 = w'x' + w'y$

b) $f_2 = \sum(0, 2, 3, 6, 7, 8, 10, 11, 14, 15)$

We put all ones in the specific blocks as shown in the Figure 7.5(b). Since karnaugh map is considered to be in spherical, therefore we can combine all 4 corners of the map. And another rule permits to combine all eight ones, as shown in figure.



$w'x'y\,\cancel{z}$ $w'x'\cancel{y}'z'$

$w'x\,y\,\cancel{z}$ $w'x'\cancel{y}\,z'$

$w'\cancel{x}\,y\,\cancel{z}$ $w\,x'\cancel{y}'z'$

$w'\cancel{x}\,y\,\cancel{z}'$ $w\,x'\cancel{y}\,z'$

$w\,\cancel{x}\,y\,\cancel{z}$ $x'\;z'$

$w\,\cancel{x}\,y\,\cancel{z}'$

$w\,\cancel{x}\,'y\,\cancel{z}$

$w\,\cancel{x}\,'y\,\cancel{z}'$

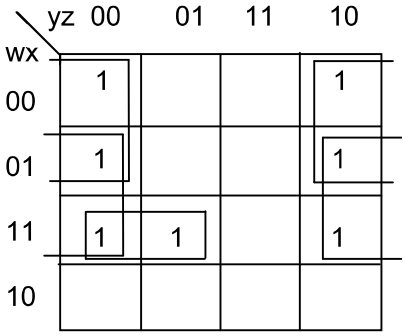
y

Fig. 9.5(b): $f_2 = \sum(0,2,3,6,7,8,10,11,14,15)$

Therefore, $f_1 = y + x'z'$

c) $Y = \sum(2, 4, 6, 12, 13, 14)$

We put all ones in the specific blocks as shown in the Figure 7.5(c). First we combine blocks m_0, m_2, m_4, m_6. Next we can combine blocks m_{12} and m_{14}. But, as a rule, preferences will be given to combine maximum number of blocks; therefore the combination of blocks m_4, m_6, m_{12}, m_{14} may be more acceptable. After that block m_{13} left alone. We can combine block m_{13} is with the nearest block m_{12}.



$w'\cancel{x}'\cancel{y}'\cancel{z}'$ $\cancel{w}'x\cancel{y}'z'$ $wxy'\cancel{z}'$

$w'\cancel{x}\,y'\cancel{z}'$ $\cancel{w}\;x\cancel{y}'z'$ $wx\cancel{y}'\cancel{z}$

$w'\cancel{x}'\cancel{y}\,z'$ $\cancel{w}'x\cancel{y}\;z'$ wxy'

$w'\cancel{x}\,y\,\cancel{z}'$ $\cancel{w}\;x\cancel{y}\;z'$

$w'\;\;z'$ $x\;\;z'$

Fig. 9.5(c): $Y = \sum(0,2,4,6,12,13,14)$

Therefore, $Y = w'z' + xz' + wxy'$

Some examples of possible combination of blocks of four variable Karnaugh maps are shown below:

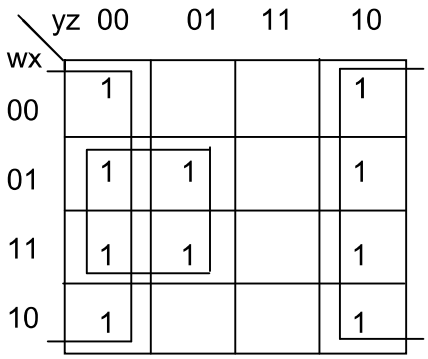


Fig. 9.5(d): Y = $\sum(0,2,4,5,6,8,10,12,13,14)$

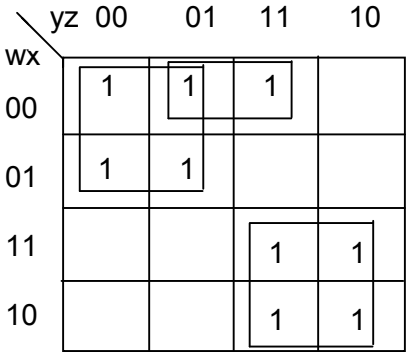


Fig. 9.5(e): Y = $\sum(0,1,3,4,5,10,11,14,15)$

Note: Complement of a function can be obtained from the K-map by considering all the mean terms/boxes not available in the function and these are marked as "0". Combining the adjacent 0's we can obtain the simplified complemented function. For example,

$f_1 = \sum(2, 3, 6, 7)$

$f_1' = \sum(1, 4, 5)$

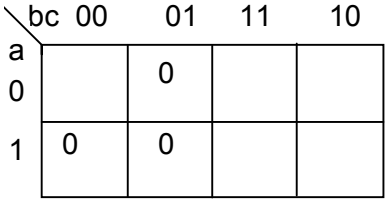


Fig. 9.6: Plotting complement of a function



CHECK YOUR PROGRESS

Q.5: Minimize the following expressions using Karnaugh map method:

a) $f_1 = x'y + xy$

b) $f_2(x,y,z) = \Sigma\,(0,1,4,5,6)$

c) $f(a,b,c,d) = \Sigma\,(0,1,2,5,8,9,10)$

Don't care conditions: Sometimes a situation arises in which some input variable combinations are not allowed in the Boolean expression. So, we do not care what the function output is to be for these combinations of variables as they are guaranteed never to occur. These types of combinations are called don't care conditions. Don't care conditions are marked as X in Karnaugh map which can

be combined with all 1's and all 0's if needed. Combining the X's with 1's or 0's help in further simplification of the function.



9.6 LET US SUM UP

- A boolean algebra is an algebraic structure, supplied with two binary operators '+' (plus) and '.' (dot) and two identity elements 0 and 1.
- Duality principle states that every algebraic expression deducible from the postulates of boolean algebra remains valid if the operators and identity elements are interchanged.
- Postulates of boolean algebra:

  1) (a) $x + 0 = x$ (b) $x.1 = x$

  2) (a) $x + x' = 1$ (b) $x.x' = 0$

  3) (a) $x + y = y + x$ (b) $x.y = y.x$

  4) (a) $x(y+z) = xy + xz$ (b) $x + yz = (x+y)(x+z)$

- Theorems of boolean algebra:

  1) (a) $x + x = x$ (b) $x.x = x$

  2) (a) $x + 1 = 1$ (b) $x.0 = 0$

  3) (a) $x + (y+z) = (x+y) + z$ (b) $x(yz) = (xy)z$

  4) (a) $(x+y)' = x'y'$ (b) $(xy)' = x' + y'$

  5) (a) $x + xy = x$ (b) $x(x+y) = x$

  6) $(x')' = x$

- A boolean expression can be simplified using algebraic methods and karnaugh's map method.



9.7 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: Sum of minterm, $f = ab'c + a'bc + abc'$

Product of maxterm, $f = (a'+b)(a+b')$

Ans. to Q. No. 2: Sum of minterm of $f_1 = \sum(1,2,5,7)$

Product of maxterm of $f_1 = \pi(0,3,4,6)$

Ans. to Q. No. 3: (a) $y = bc + bc' + b'c$

(b) $f_1(a,b,c) = \Sigma(1,3,5,7)$

Ans. to Q. No. 4: (a) $f_1 = (x+y)(x+y')(x'+y)$

(b) $y = \pi(1,2,3,4,5,6,)$

Ans. to Q. No. 5: (a) $f_1 = y$

(b) $f_2(x,y,z) = y' + xz'$

(c) $f(a,b,c,d) = b'd' + b'c' + a'c'd$



9.8 FURTHER READING

1) Floyd, T. L. (2011): *Digital Fundamentals*; Pearson Education India.

2) Morris, M. M. (1987): *Digital Logic and Computer Design*.



9.9 MODEL QUESTIONS

Q.1: Express the following boolean expressions in sum of minterm:

a) $f_1(x,y,z) = xy + y'z$

b) $y = ac + ad$

Q.2: Express the following expressions in product of maxterm:

a) $y = (a+c)(a+d)$

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

Q.3: Minimize the following expressions using Karnaugh map method:

a) $f_3(x,y,z) = \Sigma(0,2,4,5,6)$

b) $f_4(x,y,z) = \Sigma(1,2,3,5,6,7)$

c) $f(w,x,y,z) = \Sigma(2,4,6,8,9,10,12,13)$

d) $f = \Sigma(0,1,2,3,4,5,6,8,12)$

*** ***** ***

UNIT 10 : BINARY ARITHMETIC CIRCUITS

UNIT STRUCTURE

10.1 Learning Objectives

10.2 Introduction

10.3 Half-Adder

10.4 Full-Adder

10.5 Birary Parallel Adder

10.6 Serial Adder

10.7 Subtractor Circuit

- 10.7.1 Half-Subtractor
- 10.7.2 Full-Subtractor

10.8 Let Us Sum Up

10.9 Answer to Check Your Progress

10.10 Further Reading

10.11 Model Questions

10.1 LEARNING OBJECTIVES

After going through this unit you will be able to :

- describe working principle of half-adder
- describe working principle of full-adder
- describe working of parallel adder
- describe working of serial adder
- describe working principle of half-subtractor
- describe working principle of full-subtractor.

10.2 INTRODUCTION

In the previous unit, 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.

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.

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*. Sometimes it becomes necessary to add three 1-bit binary numbers. The circuit for this is called *full-adder*.

10.3 HALF-ADDER

In the truth 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 truth 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.

Truth Table for a Half-Adder

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

The minterms for Sum and CARRY are shown in the bracket. The Sum-Of-Product equation for SUM is :

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

Similarly the SOP equation for the CARRY is :

$C = XY$ ..... (2)

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



Figure 10.1 : Half-Adder Circuit and Symbol

10.4 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 C and S are CARRY & SUM.

Truth Table for Full-Adder

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

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 :

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

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

$= \bar{X}S + XS$..... (3)

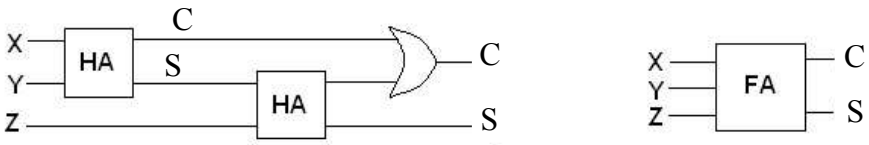
Here S is SUM of Half-Adder.

Again SOP equation for Full – Adder CARRY is :

$$C = \bar{X}YZ + X\bar{Y}Z + XY\bar{Z} + XYZ$$
$$= \bar{X}YZ + XYZ + X\bar{Y}Z + XY\bar{Z}$$
$$= (\bar{X} + X)\,YZ + X\,(\bar{Y}Z + Y\bar{Z})$$
$$= YZ + XS$$
$$= C + XS \quad \ldots\ldots\ldots\ldots\ldots\ldots (4)$$

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.



Figurer 10.2 : Full-Adder Circuit and Symbol

10.5 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 11.3. 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 11.3 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.

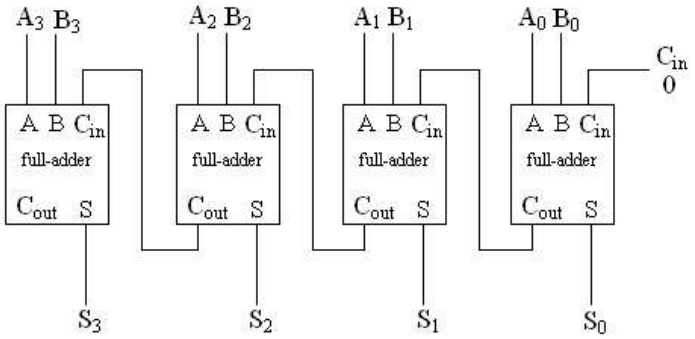


Figure 10.3 : A 4-bit binary parallel adder

10.6 SERIAL ADDER

A serial adder is used to add serial binary numbers. To implement a serial adder, one full-adder is required. In figure 11.4, 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.

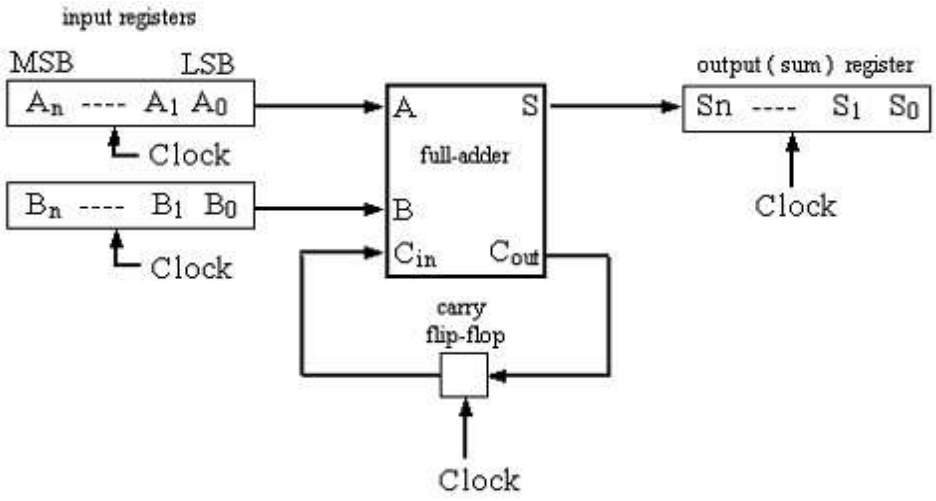


Figure 10.4 : A serial adder

10.7 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*.

10.7.1 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 :

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

$= S$ ..... (5)

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



Fig. 10.5 : The half-subtractor circuit and symbol

10.7.2 Full-Subtractor

A full-subtractor circuit finds the difference and borrow for 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 :

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

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

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

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

And SOP equation for BORROW is :

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

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

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

$= \bar{D}Z + \bar{X}Y$ ..... (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.

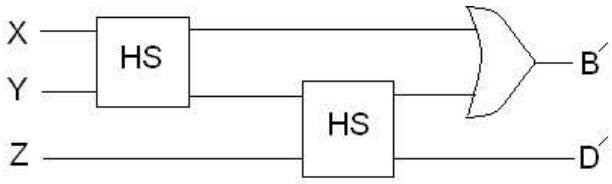
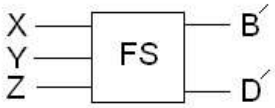


Fig. 10.6 : Full-Subtractor Circuit

The symbol of full-subtractor is :





CHECK YOUR PROGRESS

Q.1: A half-adder can add :

(a) Two binary number of 4 bit each

(b) Two binary bit

(c) Add half of a binary number

(d) None of these

Q.2: A full-adder is a logic circuit that has two output namely:

(a) product & sum

(b) sum & borrow

(c) sum & carry

(d) 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



10.8 LET US SUM UP

- Adder circuits are used to add binary bits.
- Subtractor circuits are used to subtract bits.
- Adder and subtractor circuits can perform binary addition and subtraction.
- There are different types of adder and subtractor like 2 bit adder/ subtractor, 3 bit adder/subtractor etc.
- Serial adder is used to add serial binary numbers.



10.9 ANSWERS 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)



10.10 FURTHER READING

1) Mano, M. M. (2017), *Digital Logic and Computer Design*, Peason Education India.
2) Mano, M. M. (1982), *Computer System Architecture*, Englewood Oliffs, N.J., 53-54.
3) Malvino, A.P. & Leach, P. (1986), *Digital Principles and Applications*, Mcgraw-Hill International.
4) Lee, C. (1976), *Digital Circuits and Logic Design.*
5) Talukdar, P. (2010), *Digital Techniques*, N. L. Publications.
6) Bogart, F. (1992), *Introduction to Digital Circuits*.



10.11 MODEL QUESTIONS

Q.1: Use truth table and logic diagram to explain the working of a half adder and full-adder circuit.

Q.2: Use truth table and logic diagram to explain the working of a half and full-subtractor circuit.

Q.3: Draw and explain the operation of a 4-bit parallel adder.

Q.4: Explain how a serial adder works using block diagram.

Q.5: Give the logic diagram of half adder and half subtractor.

Q.6: Give the truth table for full subtractor.

*** ***** ***

UNIT 11: COMBINATIONAL CIRCUIT

UNIT STRUCTURE

11.1 Learning Objectives

11.2 Introduction

11.3 Combinational Circuit

11.4 Multiplexer

11.4.1 4-to-1 Multiplexer

11.5 Demultiplexer

11.5.1 1-to-4 Line Demultiplexer

11.6 Encoder

11.7 Decoder

11.7.1 3-to-8 Line Decoder

11.8 Half-Adder and Full-Adder

11.9 Half-Subtractor and Full-Subtractor

11.10 Let Us Sum Up

11.11 Answers To Check Your Progress

11.12 Further Reading

11.13 Model Questions

11.1 LEARNING OBJECTIVES

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

- describe combinational circuit and its applications
- learn about multiplexer
- learn about demultiplexer
- describe decoder, encoder
- describe half-adder, full-adder
- describe half-subtractor, full-subtractor.

11.2 INTRODUCTION

In the previous units we have learned the concepts of different number systems and their conversion from one form to another. We have also discussed the different types of logic gates.

The fundamentals of combinational and binary arithmetic circuit are introduced in this unit. These circuits are the backbone of any computer system and knowing them is quite essential. Combinational circuits are interconnected circuits of gates according to certain rules to produce an output depending on its input values.

11.3 COMBINATIONAL CIRCUIT

A combinational circuit is a collection of some interconnected set of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs.

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept binary signals from the inputs and generate some binary signals to the outputs. Here binary signals mean only two possible values, one representing logic-1 and the other logic-0. A combinational circuit can be represented as a network of gates and, therefore, can be expressed by a truth table or boolean expression. A block diagram of combinational circuit is shown in Figure 8.1.

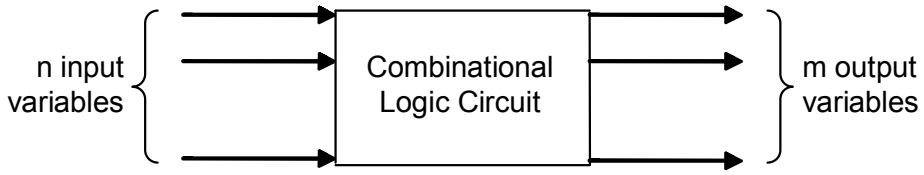


Fig. 11.1: Block Diagram of Combinational Circuit

Here n input binary variables come from an external source and the m output variables go to an external destination. For n input variables, there are 2^n possible combinations of binary input values. For each possible input combination, there is only one possible output combination. A combinational circuit can be described by m boolean functions, one for each output variable. Each output function is expressed in terms of the n input variables.

The basic design issue related to combinational circuits is: the minimization of number of gates. The normal circuit constraints for combinational circuit design are:

- the depth of the circuit should not exceed a specific level.
- number of input to a gate and to how many gates its output can be fed.

11.4 MULTIPLEXER

A multiplexer is a combinational circuit where binary information from one of many input lines are selected and directs it to a single output line. Hence, it is also called a data selector. There are some selection lines in a multiplexer, present to control the selection of the particular input line. In general there are 2n input lines and n selection lines whose bit combinations determine the selection of the input that will be directed as output.

11.4.1 4-to-1 Multiplexer

A 4-to-1 multiplexer consists of four input lines, two selection lines, one output line, four AND gates, one OR gate and two inverters. A 4-to-1 multiplexer circuit is shown in Figure 8.2(a). Here A, B, C and D are input lines, O is output line and S_0, S_1 are selection lines. Selection lines S_0 and S_1 are responsible for selecting a particular AND gate. All possible input to output paths for each possible bit combination of selection lines are shown in the function table in Figure 8.2(b). Now, let us assume that $S_0 S_1$ = 01. In this case the AND gate associated with input B has two of its inputs equal to 1 and third input connected to B. The other three AND gates have atleast one input equal to 0, so it makes their output equal to 0. So the OR gate output is now equal to the value of B, thus providing the selected input to output path.

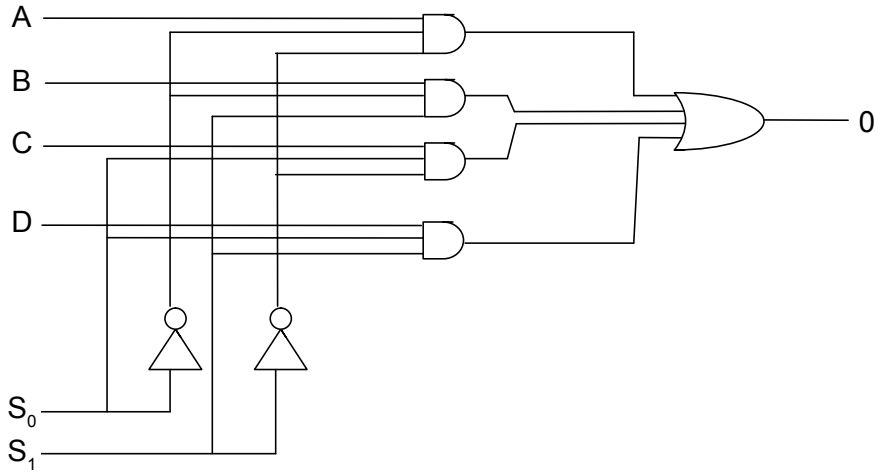


Fig. 11.2(a): Logic Diagram of 4-to-1 Multiplexer

S_1	S_0	0
0	0	A
0	1	B
1	0	C
1	1	C

Fig. 11.2 (b): Function Table

11.5 DEMULTIPLEXER

Demultiplexing is the reverse process of multiplexing; i.e., a demultiplexer is a combinational circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. There are n selection lines to select the specific output line from 2^n output lines i.e the bit values of n selection lines control the selection of the specific output line.

11.5.1 1-to-4 Demultiplexer

A decoder with an enable input can be referred to as a demultiplexer.

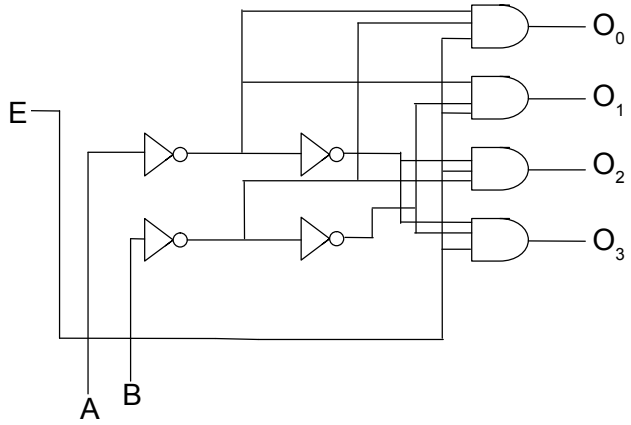


Fig. 11.3(a): Logic Diagram of 1-to-4 Line Demultiplexer

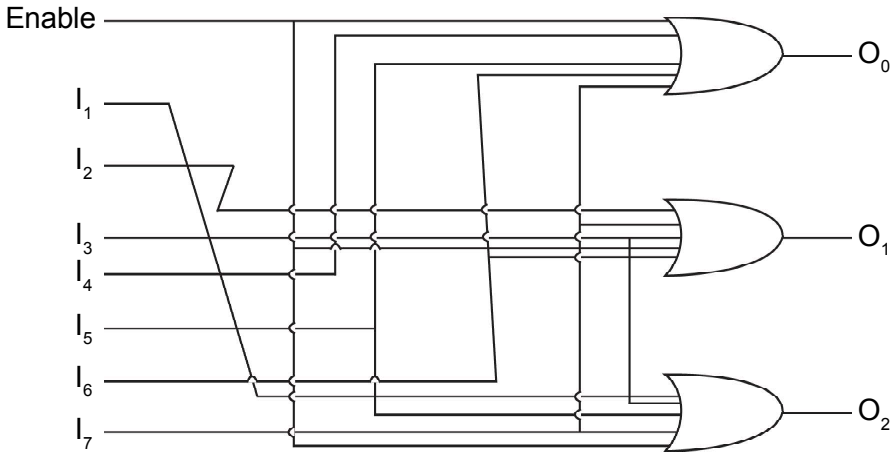
In Figure 8.3(a), E is the input line and A,B are selection lines. Now depending upon the values of AB, the informations from E transmit to only one of output lines O_0,O_1,O_2 and O_3. The truth table for all possible combination of values of AB are given in Figure 8.3(b).

A	B	Output Line
0	0	O_0
0	1	O_1
1	0	O_2
1	1	O_3

Fig. 11.3 (b): Truth Table for 1-to-4 Line Demultiplexer

11.6 ENCODER

An encoder is a combinational circuit that produces the reverse function from that of a decoder. An encoder consists of 2^n (or less) input lines and n output lines. The output lines generate the binary code for the 2^n input variables.



$O_0 = I_4 + I_5 + I_6 + I_7$

$O_1 = I_2 + I_3 + I_6 + I_7$

$O_2 = I_1 + I_3 + I_5 + I_7$

Fig. 11.4(a): Logic Diagram of Octal-to-Binary Encoder

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	O_0	O_1	O_2
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Fig. 11.4(b): Truth Table for Octal-to-Binary Encoder

An octal to binary encoder logic circuit is shown in Figure 8.4(a). It has eight inputs, one for each of the eight digits, and three outputs that generate the corresponding binary number. It is constructed with OR gates and its inputs can be determined from the truth table given in Table 8.4(b). Here we assume that only one input line can be equal to 1 at any time. In this case the circuit has eight inputs and could have 28 = 256 possible input combinations. Only eight of these combinations have any meaning. The other input combinations are “don’t care conditions”. The type of encoder available in integrated circuits form is called a priority encoder. In these types of encoders, only the highest priority input line is encoded and so an input priority is established.

11.7 DECODER

A decoder is a combinational logic circuit that receives coded information on n input lines and converts them to maximum of 2^n unique output lines. The decoder output will have less than 2^n outputs when some of the n-bit decoded information are unused or are don’t care combinations.

11.7.1 3-to-8 Line Decoder

A 3-to-8 line decoder is shown in Figure 8.5(a). Here there are 3 input lines and 8 output lines. The 3 inputs are decoded into eight outputs, each output representing one of the minterms of the 3 input variables. The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of minterms. A 3-to-8 line decoder can be used for decoding any 3-bit code to provide eight outputs, one for each element of the code.

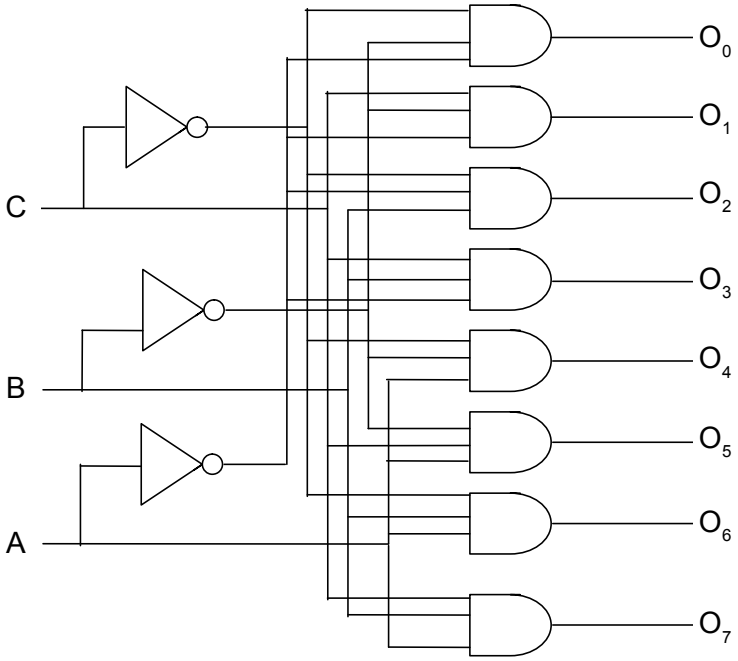


Fig. 11.5(a): Logic Diagram of 3-to-8 Line Decoder

The decoder is used in some code converters, for example BCD- to-seven segment decoder. A 3-to-8 line decoder can be used in binary to octal conversion. Another use of a decoder is that it can be used to implement any combinational circuit.

A	B	C	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_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. 11.5(b): Truth Table for 3-to-8 Line Decoder

11.8 HALF-ADDER AND FULL-ADDER

Half Adder: A half-adder is a combinational circuit that performs the addition of two bits. So in this logic circuit two binary inputs and two binary outputs are required. Let us assume, two input variables A , B and

two output variables S for sum and C for carry. Now the truth table according to the function of the half-adder is given in Figure 8.6(a).

Now the simplified boolean function for the two outputs can be obtained directly from the truth table. The simplified Sum-of-Product expressions are:

$S = A'B + AB'$

$C = AB$

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Fig.11.6(a): Truth Table for Half Adder

Observing the truth table we see that the function S = 1 only when A = 0, B = 1 and A = 1, B = 0. So, $S = A'B + AB'$. Similarly C = 1, only when A = 1 and B = 1 so C = AB.

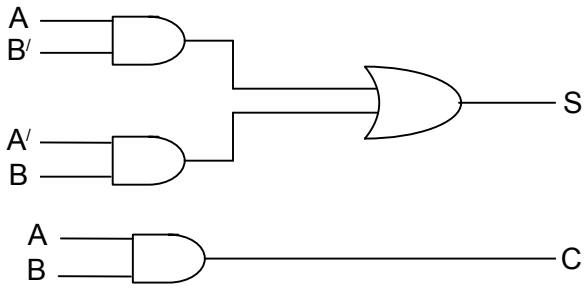


Fig. 11.6(b): Logic Circuit of Half Adder

Full Adder: The addition of three bits is performed by a combinational logic circuit called full-adder. It consists of three inputs and two outputs. A half adder has only two inputs and there is no provision to add a carry coming from the lower order bits when multibit addition is performed, so in this case a full-adder can be used. In case of a full-adder two of the input variables (A and B) represent the two significant bits to be added and the third variable (C) represent the carry from the previous lower significant position. The output variable S gives the value of the least significant bit of the sum and output variable C gives the output carry. The truth table of the full-adder is given below in Figure 8.7(a).

Now, using K-map the simplified Boolean expression for full-adder are:

$S = X'Y'Z + X'YZ' + XY'Z' + XYZ$

$C = XY + XZ + YZ$

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Fig. 11.7(a): Truth Table for Full-Adder

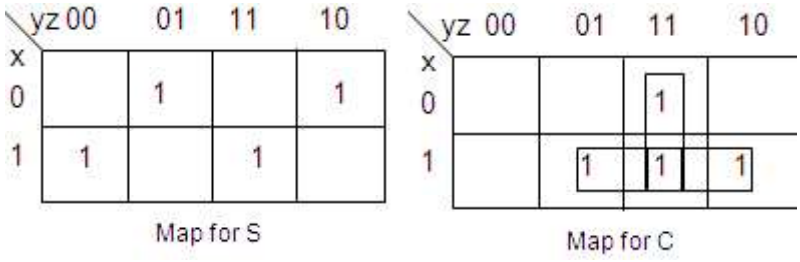


Fig. 11.7(b): Map for Full-Adder

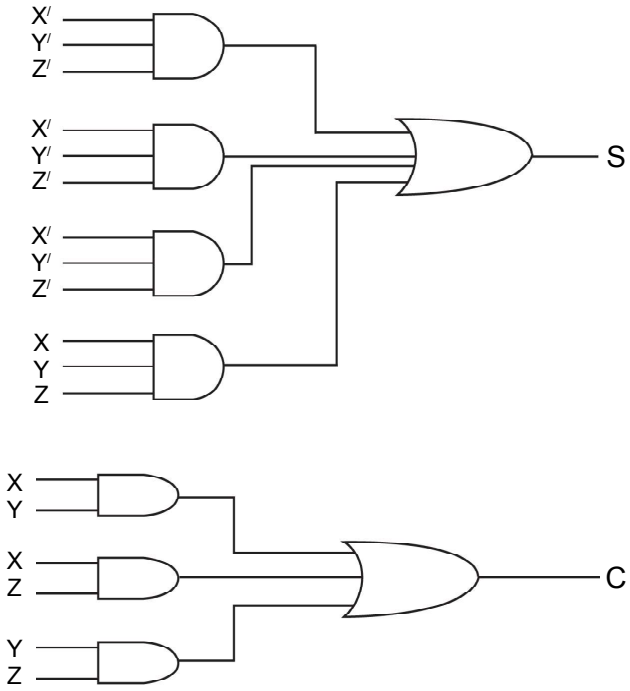


Fig. 11.7(c): Logic Circuit of Full-Adder

11.9 HALF-SUBTRACTOR AND FULL-SUBTRACTOR

Half Subtractor: The half subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, the minuend X and subtrahend Y and two outputs the difference D and borrow out $B_{out.}$ The borrow out signal is set when the subtractor needs to borrow from the next digit in a multi-digit subtraction. That is, $B_{out} = 1$ when $X < Y$. Since X and Y are bits, $B_{out} = 1$ if and only if $X = 0$ and $Y = 1$ i.e. $B_{out} = X' .Y$

Using the truth table for half subtractor, we find the following logic equations for D and B_{out}:

$$D = X'Y + XY'$$

$$B_{out} = X.Y$$

The truth table for the half subtractor is:

Inputs		Outputs	
X	Y	D	B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Fig. 11.8(a): Truth Table for Half Subtractor

Full Subtractor: The full subtractor is a combinational circuit which is used to perform subtraction of three input bits: the minuend X, subtrahend Y, and borrow in B_{in}. The full subtractor generates two output bits: the difference D and borrow out B_{out}.B_{in} is set when the previous digit is borrowed from X. Thus, B_{in} is also subtracted from X as well as the subtrahend Y, or in symbols: $X – Y – B_{in}$.

Like the half subtractor, the full subtractor generates a borrow out when it needs to borrow from the next digit. Since we are subtracting X by Y and B_{in}, a borrow out needs to be generated when $X < Y + B_{in}$. When a borrow out is generated, 2 is added in the current digit. This is similar to the subtraction algorithm in decimal. Instead of adding 2, we add 10 when we borrow. Therefore, $D = X – Y – B_{in} + 2B_{out}$

Inputs			Outputs	
X	Y	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Fig. 11.8(b): Truth Table for Full Subtractor



CHECK YOUR PROGRESS

Q.1: Choose the correct option:

i) In 4-to-1 multiplexer there are:

(a) two selection lines (b) three selection lines

(c) four selection lines (d) none of these

ii) Demultiplexing is the reverse process of:

(a) decoder (b) encoder

(c) full-adder (d) none of these

iii) The type of encoder available in IC form is called:

(a) octal to binary encoder (b) priority encoder

(c) decoder/demultiplexer (d) none of these

iv) Which of the following combinational circuit is called data selector:

(a) decoder (b) decoder/ demultiplexer

(c) demultiplexer (d) none of these

v) Addition of three bits can be performed by:

(a) half-adder (b) ful-adder

(c) both half adder and full adder (d) none of these

vi) A decoder with n input lines contains maximum of:

(a) 2^n output lines (b) n output lines

(c) 1 output line (d) n^2 output lines



11.10 LET US SUM UP

- A combinational circuit consists of some interconnected set of logic gates with some input variables and output variables.
- A multiplexer is a combinational circuit where binary information is selected from one of 2^n input lines and transmitted to a single output line. Here n is the number of selection lines.
- An encoder is a combinational circuit which generates the binary code for the 2^n input variables.
- A decoder is a combinational circuit that recieves coded information on n input lines and transmits them to maximum of 2^n unique output lines after conversion.
- A half-adder is a combinational circuit that adds two bits.
- A full-adder is combinational circuits that adds three bits.



11.11 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) (a); ii) (d); iii) (b); iv) (d); v) (b); vi) (a)



11.12 FURTHER READING

1) Floyd, T. L. (2011): *Digital Fundamentals*; Pearson Education India.
2) Morris, M. M. (1987): *Digital Logic and Computer Design*.
3) Morris, M. M.: *Computer System Architecture*; PHI Publication.



11.13 MODEL QUESTIONS

Q.1: Explain the function of multiplexer. Draw the logic diagram of 4-to-1 multiplexer.

Q.2: Explain full-adder with logic diagram.

Q.3: Describe half-subtractor using truth tables.

*** ***** ***

UNIT 12: SEQUENTIAL CIRCUITS

UNIT STRUCTURE

12.1 Learning Objectives

12.2 Introduction

12.3 Sequential Circuit

12.3.1 Synchronous and Asynchronous Sequential Circuit

12.4 Latch

12.5 Flip-Flop

12.5.1 RS Flip-Flop

12.5.2 D Flip-Flop

12.5.3 T Flip-Flop

12.5.4 JK Flip-Flop

12.5.5 Master Slave Flip-Flop

12.5.6 JK Master Slave Flip-Flop

12.6 Let Us Sum Up

12.7 Answers to Check Your Progress

12.8 Further Reading

12.9 Model Questions

12.1 LEARNING OBJECTIVES

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

- describe sequential circuits
- describe different types of flip-flops
- define latch.

12.2 INTRODUCTION

In the previous units we have learnt about the concepts of combinational circuits. We have also already discussed the basic binary arithmetic circuits.

The fundamentals of sequential circuit are introduced in this unit. These circuits are the backbone of any computer system and knowing them

is quite essential. Different types of flip-flops and their uses will also be covered in the unit. The flip-flops are a basic building block for counters, registers, and other sequential control logic and are used in certain types of memories. In the same way, combinatorial circuits are generalizations of gates, sequential circuits are generalizations of flip-flops.

12.3 SEQUENTIAL CIRCUIT

A circuit is said to be sequential circuit if its output is the function of present input and past output. The past output of the circuit is available in memory element storage known as flip-flop. The basic element used in sequential circuits is a flip-flop, which can be made using logic gates and is available in IC (Integrated Circuit) form. A block diagram of a sequential circuit is in Figure 9.1(a).

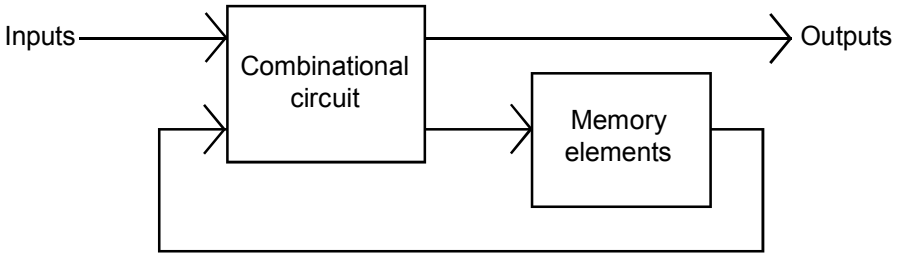


Fig. 12.1(a): Block Diagram of a Sequential Circuit

It consists of a combinational circuit to which memory elements are connected to form a feedback path. The memory elements are devices capable of storing binary information within them. These circuits store and remember information. The sequential circuit receives binary information from external inputs.

Classification of sequential circuits depends on the timing of their signals. They are classified as follows:

i) synchronous sequential circuits and
ii) asynchronous sequential circuits.

12.3.1 Synchronous and Asynchronous Sequential Circuit

Synchronous Sequential Circuits use flip-flops and their status can change only at discrete instant of time. The

synchronisation in sequential circuits can be achieved using a clock pulse generator. It synchronises the effect of input over output. It presents signal of the following form as shown in Figure 9.1(b):

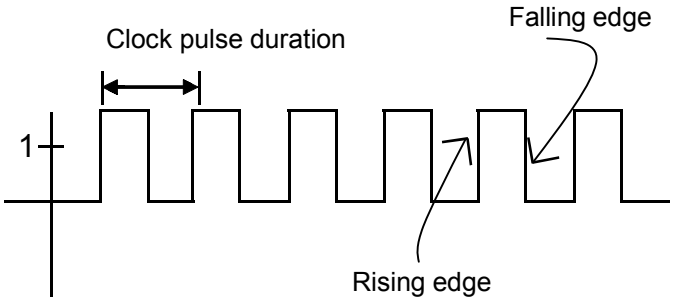


Fig. 12.1(b): Clock Signal of a Clock Pulse Generator

From the figure we can see that the clock pulse is the time between successive transitions in the same direction, that is, between two rising or two falling edges. State transitions in synchronous sequential circuits are made to take place at times when the clock is making a transition from 0 to 1 (rising edge) or from 1 to 0 (falling edge). There is no change in the information stored in memory, once it has changed at the time of transition (from 0 to 1 or 1 to 0), within the pulse duration.

The behaviour of ***Asynchronous Sequential Circuits*** depends upon the order in which its input signals change and can be affected at any instant of time.

Asynchronous sequential circuits:

- do not use clock pulses. The change of internal state occurs when there is a change in the input variable.
- memory elements are either flip-flops (FF) without clocked inputs or time delay-elements.
- often resemble combinational circuits with feedback.
- are used when speed of operation is important.

12.4 LATCH

Latch is one of the building blocks of sequential circuits. One latch can store one bit of information i.e. either 0 or 1. The outputs of latches are constantly affected by their inputs as long as the enable signal is asserted.

In other words, when they are enabled, their content changes immediately when their inputs change.

12.5 FLIP-FLOPS

Flip-Flops are the basic building blocks of sequential circuits. A flip-flop is a binary cell which can store a bit of information. A basic function of flip-flop is storage, which means memory. A flip-flop (FF) is capable of storing 1 (one) bit of binary data. It has two stable states either '1' or '0'. A flip-flop maintains any one of the two stable states which can be treated as zero or one depending on presence and absence of output signals.

A flip-flop circuit has two outputs, one for the normal value and the other for the complemented value of the bit stored in it. A basic flip-flop circuit can be constructed from two NAND gates Figure 9.2(c) or two NOR gates Figure 9.2 (a). The flip-flop in the figure has two inputs R (reset) and S (set) and two outputs Q and Q/.

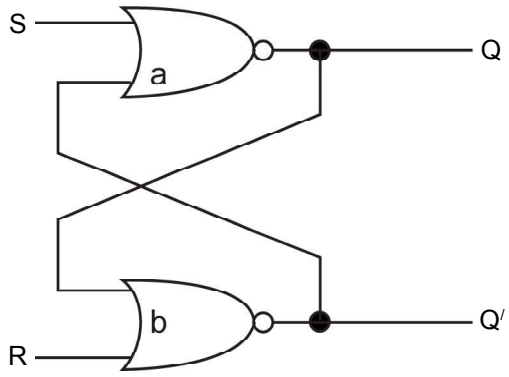


Fig. 12.2 (a): Basic Flip-Flop Circuit with NOR Gates

Input		Output	
S	R	Q	Q/
1	0	1	0
0	0	1	0 (after S = 1), R = 0)
0	1	0	1
0	0	0	1 (after S = 0, R = 1)
1	1	0	0

Fig. 12.2 (b): Truth Table for Basic Flip-Flop Circuit with NOR Gate

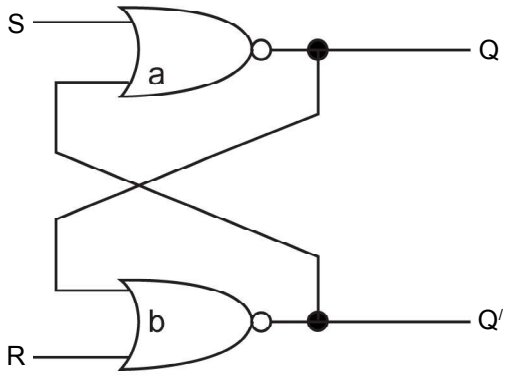


Fig. 12.2(c): Basic Flip-Flop Circuit with NAND Gate

Input		Output	
S	R	Q	$Q^{/}$
1	0	0	1
1	1	0	1 (after S = 1), R = 0)
0	1	1	0
1	1	1	0 (after S = 0, R = 1)
0	0	1	1

Fig. 12.2(d): Truth Table for Basic Flip-Flop Circuit with NAND Gate

Functions of the basic flip-flop circuit with NOR gates:

In normal mode of operation both the flip-flop inputs are at 0 i.e., S = 0 and R = 0. This flip-flop can show two states: either the value of Q is 1 (i.e., set state) or the value of Q is 0. We call it clear state. Let us see how the S and R input can be used to set and clear the state of the flip-flop. Suppose the flip-flop was in set state i.e., Q = 1 and $Q^{/}$ = 0 and as S = 0 and R = 0, the output of NOR gate ‘‘a’’ will be 1 since both its input Q' and R are zero and NOR gate “b” will show output 0 as one of its input Q is 1. Similarly if flip-flop was in clear state then $Q^{/}$ = 1 and R = 0, therefore, output of gate ‘‘a’’ will be ‘0’ and gate will “b” 1. Thus, flip-flop maintains a stable state at S = 0 and R = 0 i.e. states of the flip flop would not change if both the inputs S and R are zero.

The flip-flop is taken to set if the S input momentarily goes to 1 and then goes back to 0. R remains at zero during this time. Suppose, initially the flip-flop was in state 0, i.e., the value of Q was 0. As soon as S becomes

1 the output of NOR gate 'b' goes to '0' i.e., Q' becomes 0 and immediately Q becomes 1 as both the input (Q' and R) to NOR gate 'a' become 0. The change in the value of S back to 0 does not change the value of Q again as the input to NOR gate 'b' now are Q = 1 and S = 0. Thus, the flip-flop stays in set state 1 even after S returns to zero.

If the flip-flop was in state 1 then, when S goes to 1 there is no change in value of Q' as both the input to NOR gate 'b' are 1 at this time. Thus, Q' remains in state 0 or in other words flip-flop stays in set state.

If R input changes to value 1 then flip-flop goes to the clear state. In changing momentarily the value of R to 1 the Q output changes to 0 irrespective of the state of flip-flop and as Q is 0 and S is 0 the Q' becomes 1. Even after R changes back to value 0, Q remains 0 i.e., flip-flop stays in the clear state.

When both S and R goes to 1 at the same time, well this is the situation which may create a set or clear state depending on which of the S and R stays longer in zero state. But meanwhile when both of them are 1 the value of Q and Q' becomes 1 which implies that Q and its complement both are one, which is an impossible situation. Therefore, the transition of both S and R to 1 simultaneously is an undesirable condition for this basic circuit. Function of the other basic flip flop i.e. with NAND gates as shown in Figure 9.2 can be similarly described remembering the following points.

i) The output of a NAND gate is 0 if all the inputs are 1 and the output is 1 if any input is 0.

ii) The flip flop maintains a stable state when S = 1, R = 1.

iii) A momentary change in the S input i.e. applying a 0, will set the flip flop (Q = 1, Q' = 0)

iv) The flip-flop will be in clear state (Q' = 1, Q = 0) on applying the inputs R = 0, S = 1.

There are many types of flip-flops. They are:

- RS Flip-Flop (Reset-Set Flip-Flop)
- D type Flip-Flop (Data Latch)
- T type Flip-Flop (Triggered Flip-Flop)
- JK Flip-Flop

- Master-Slave Flip-Flop
- JK Master Slave Flip-Flop

These flip-flops are covered in detail in the next few sections in this unit.

12.5.1 RS Flip-Flop

We can construct a flip-flop circuit from two NOR gates Figure 9.3(a) or two NAND gates Figure 9.3(c). The cross-coupled connection from the output of one gate to the input of other gate constitutes a feedback path. Each flip-flop has two outputs, Q and Q/, and two inputs, S(set) and R(reset). The normal output Q determines the state of the flip flops i.e. whether a "0" or "1" is stored in the flip flop. If Q = 1, it implies that the flip flop is in set state and a "1" is stored. This type of flip-flop is sometimes called a direct-coupled RS flip-flop or SR latch.

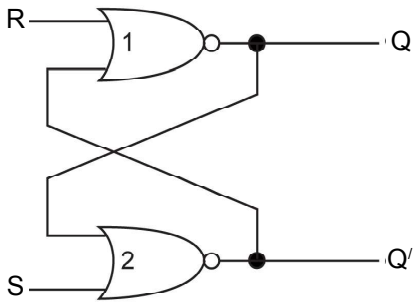


Fig. 12.3(a): Basic Flip-Flop Circuit with NAND Gates

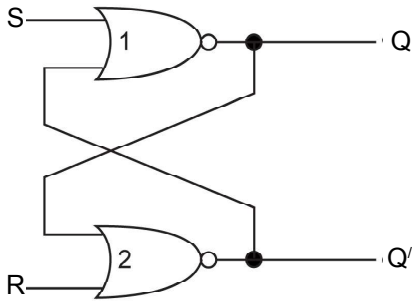


Fig. 12.3(b): Basic Flip-Flop Circuit with NOR Gates

In both the circuits, the output of one gate feeds as input of the other.

Considering the basic flip flop of Figure 9.3(b), a 0 (low) at set S of

gate 1 gives an output 1 (high) at Q. This output 1 is fed to gate 2 along with 1 at R. The output of 2nd gate is 0 (low) at $Q^{/}$. So the output at Q and $Q^{/}$ are 1 and 0 for inputs of 0 and 1 at S and R respectively. This is the set condition and remains in that until cleared. To clear the set condition, we operate the reset and the operation is reverse of set. R is activated by 0 (low). The output of 2nd gate at $Q^{/}$ is 1 (high). The 1 (high) output of 2nd gate is fed to 1st gate along with 1 at S. The two 1's give output of 0 at Q. This state also is a stable one. Thus we see the normal output at Q is either 1 or 0, which is one bit of information in binary. Hence a flip-flop can store one bit of information.

If both R and S are 1 (high) no change can occur in the output from the previous state. This is known as hold state.

If both S and R are 0 (low), both outputs are driven to 1 (high). This is a prohibited state which gives rise to racing condition.

RS Flip-Flop (Clocked): The main feature in clocked RS flip-flop is the addition of a clock pulse input. In this flip-flop, change in the input value of R or S will change the state of the flip-flop only if the clock pulse at that moment is 1(one). This is implemented by ANDing the clock (CP) input with the other two inputs S and R.

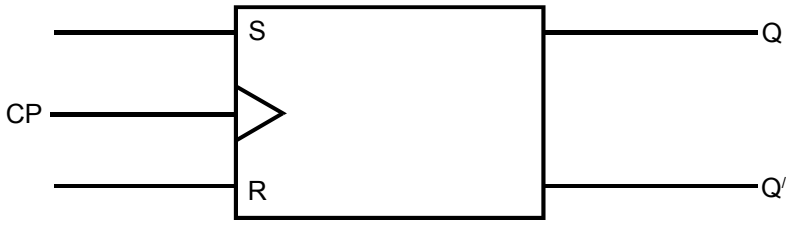


Fig. 12.4: Graphic Symbol of RS Flip-Flop

Here two inputs are set (S) and reset (R). The two outputs are Q (Normal) and $Q^{/}$ (Complement of Q). If Q is 1 $Q^{/}$ is 0 and vice versa.

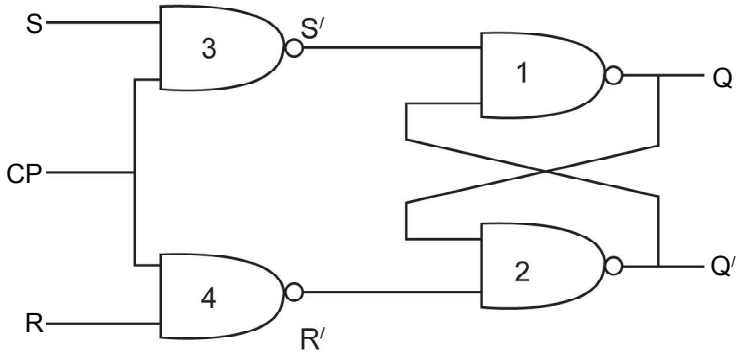


Fig. 12.5: Logic Diagram of Clocked RS Flip-Flop using NAND Gates

12.5.2 D Flip-Flop

The D flip-flop tracks the input, making transitions which match those of the input D. The D stands for ‘‘data’’. It is also known as a ‘‘data’’ or ‘‘delay’’ flip-flop. This flip-flop stores the value that is on the data line. It can be thought of as a basic memory cell. A D flip-flop can be made from a S/R flip-flop by tying the set to the reset through an inverter. The result may be clocked.

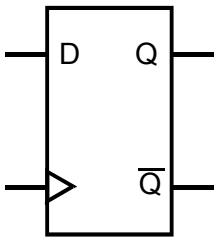


Fig. 12.6: Graphic Symbol of D Flip-Flop

12.5.3 T Flip-Flop

The T flip-flop has one input in addition to the clock. T stands for toggle. When T is asserted (T = 1), the flip-flop state toggles back and forth, and when T is de-asserted, the flip-flop keeps its current state. The T flip-flop can be constructed using a D flip-flop, with two outputs Q and Q' feedback to D input through a multiplexer that is controlled by T input.

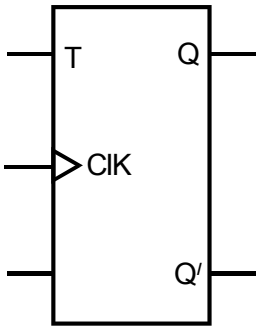


Fig. 12.7: Graphic Symbol of T Flip-Flop

The truth table for the T flip flop is as follows:

T	Q	Q_{next}	Q'_{next}
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Here, T is the only input to the flip flop, Q is the present state of the flip-flop and Q' next is the next state of the flip-flop, which is attained after occurrence of the clock pulse.

12.5.4 JK Flip-Flop

The JK flip-flop is the most versatile flip-flop, and the most commonly used flip flop when discrete devices are used to implement arbitrary state machines. A JK Flip-Flop is a refinement of the RS flip-flop. Like the RS flip-flop, it has two data inputs, J and K, and a clock input. Input J is similar to S and K is similar to R input. It has no undefined states or race condition.

The JK flip-flop constructed with the basic flip flop with NOR gates, has the following characteristics:

- If one input (J or K) is at logic 0, and the other is at logic 1, then the output is set or reset (by J and K respectively), just like the RS flip-flop, but on the (falling) clock edge. This implies that the JK flip flop is made sensitive to the negative edge transition (from 1 to 0) of CP input. However, it can also make positive edge transition sensitive.
- If both inputs are 0, then it remains in the same state as it was before the clock pulse occurred; again like the RS flip-flop.
- If both inputs are high or 1, however the flip-flop changes state whenever the (falling) edge of a clock pulse occurs; i.e., the clock pulse toggles the flip-flop.

A clocked JK flip-flop is shown in Figure 9.8(a). Output Q is ANDed with K and clock pulse (CP) inputs so that the flip-flop is

cleared during a clock pulse only if Q was previously 1. Similarly, output Q' is ANDed with J and CP inputs so that the flip-flop is set with a clock pulse only if Q' was previously 1.

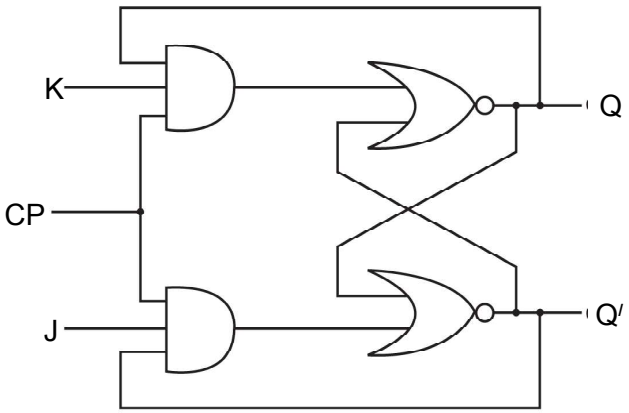


Fig. 12.8(a): Logic Diagram of JK flip-flop

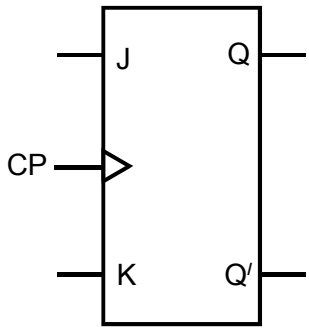
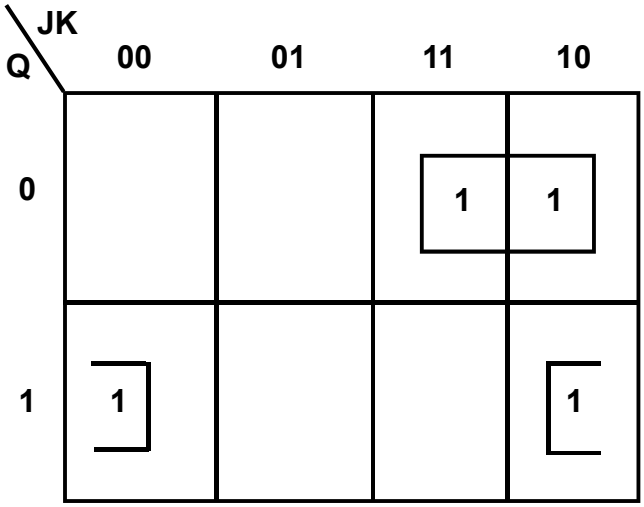


Fig. 12.8(b): Graphic Symbol

Q	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Fig. 12.8(c): Transition Table/Characteristic Table

Note: Here, Q is the present state (before occurrence of the clock pulse) and Q(t+1) is the next state (after occurrence of the clock pulse) of the flip flop.



$Q(t+1) = JQ^{/} + K^{/}Q$

Fig. 12.8 (d): Characteristic Equation

12.5.5 Master Slave Flip-Flop

The Master-Slave flip-flop is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse. The outputs from Q and $Q^{/}$ from the ‘‘Slave’’ flip-flop are fed back to the inputs of the ‘‘Master’’ with the outputs of the ‘‘Master” flip-flop being connected to the two inputs of the ‘‘Slave” flip-flop. The feedback configuration from the slave’s output to the master’s input gives the characteristic toggle of the JK flip-flop.

12.5.6 JK Master Slave Flip-Flop

JK Master slave flip-flop is a cascade of two S-R flip-flop with feedback from the output of second to input of first as shown in figure Figure 9.9. Master is a positive level triggered, but due to the presence of the inverter in the clock line, the slave will respond to the negative level. Hence when the clock = 1 (positive level) the master is active and the slave is inactive. Whereas when clock = 0 (low level) the slave is active and master is inactive.

Here, clockwise input is applied directly to the master flip-flop through an inverter to the slave flip flop. It implies that the master flip flop is activated when CP = 1, and produces the output Y and $Y^{/}$

(which are the input to the slave flip flop). The slave flip flop will not be activated at that time since the CP input is 0 to this flip flop. When the value of CP is 0 after some time, then the CP input to the slave will be 1, and the inputs Y and Y′ to this flip flop will be activated and accordingly affect the state of the flip flop (Q and Q′). Thus, when the clock pulse is high (1), the J and K inputs to the master will determine its state, the slave flip flop is then not activated, since CP input applied to it is low (0), which makes both the inputs S and R to the slave 0. The slave flip flop will go to the state already attained by the master, when the CP will be low (0).

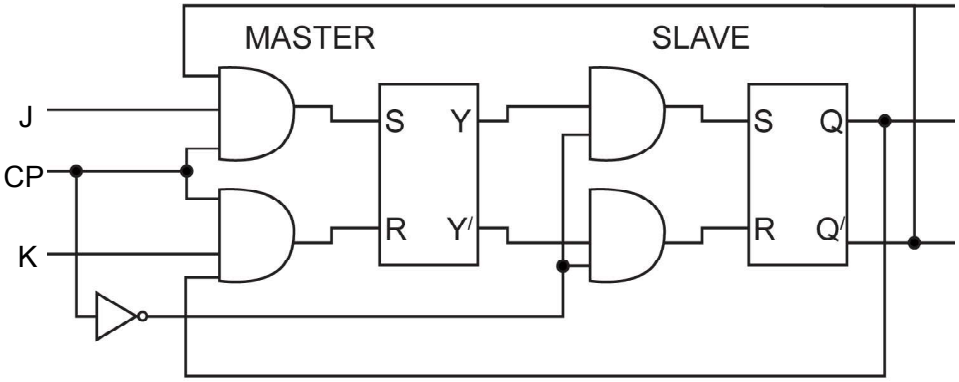


Fig. 12.9: Logic Diagram of Master Slave Flip-Flop



CHECK YOUR PROGRESS

Q.1: Choose the correct option:

i) Flip-Flop circuits can be used for:

(a) scaling (b) rectification

(c) modulation (d) counting

ii) Popular application of flip-flop are:

(a) counters (b) shift register

(c) transfer register (d) all of these

iii) A simple flip-flop:

(a) is 2 bit memory (b) is 1 bit memory

(c) is a four state device

(d) has nothing to do with memory.

iv) The purpose of the clock input to a flip-flop is to:

(a) clear the device (b) set the device

(c) always cause the output to change states

(d) cause the output to assume a state dependent on the controlling (RS, JK, or D) inputs.

v) The feature that distinguishes the JK flip flop is the:

(a) toggle condition (b) preset input

(c) type of clock (d) clear input

Q.2: State whether the following statements are True (T) or False (F):

i) Sequential circuits can be synchronous and asynchronous.

ii) A synchronous sequential circuit changes its state at discrete instants of time.

iii) A transition of a clock from 0 to 1 is called the falling edge.

iv) The memory used in synchronous sequential circuit is flip-flops.



12.6 LET US SUM UP

- A circuit is said to be sequential circuit if its output is the function of present input and past output. The basic element used in sequential circuits is a flip-flop, which can be made using logic gates and is available in IC (Integrated Circuit) form.
- ***Synchronous Sequential Circuits*** use flip-flops and their status can change only at discrete instants. The synchronisation in sequential circuits can be achieved using a clock pulse generator.
- The behaviour of ***Asynchronous Sequential Circuits*** depends upon the order in which its input signals change and can be affected at any instant of time.
- Flip-Flops are the basic building blocks of sequential circuits. A flip-flop circuit has two outputs, one for the normal value and other for the complement value of the bit stored in it.

- The main feature of clocked RS flip-flop is the addition of a clock pulse input. In this flip-flop, change in the value of R or S will change the state of the flip-flop only if the clock pulse at that moment is 1(one).
- The D flip-flop tracks the input, making transitions which match those of the input D. The D stands for "data". It is also known as a "data" or "delay" flip-flop. This flip-flop stores the value that is on the data line.
- The T flip-flop has one input in addition to the clock. T stands for toggle. When T is asserted (T = 1), the flip-flop state toggles back and forth, and when T is de-asserted, the flip-flop keeps its current state.
- A JK Flip-Flop is a refinement of the RS flip-flop. Like the RS flip-flop, it has two data inputs, J and K, and a clock input. It has no undefined states or race condition
- JK Master slave flip flop is a cascade of two S-R flip flop with feedback from the output of second to input of first.



12.7 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1: i) (b); ii) (d); iii) (b); iv) (d); v) (a)

Ans. to Q. No. 2: i) True; ii)True; iii) False; iv) True



12.8 FURTHER READING

1) Mano, M. M.: *Digital Design*; PHI Publication.

2) Talukdar. P. H. (2004): *Fundamental of Digital Circuit*.



12.9 MODEL QUESTIONS

Q.1: Define flip-flop. Explain RS flip-flop.

Q.2: Draw the logic diagram of a clocked RS flip-flop with four NAND

gates.

Q.3: Explain JK flip-flop.

Q.4: Explain D flip-flop.

Q.5: Explain T flip-flop.

Q.6: Why do we need Master Slave Flip Flops?

Q.7: Explain JK Master Slave flip-flop.

*** ***** ***

UNIT 13 : REGISTERS

UNIT STRUCTURE

13.1 Learning Objectives

13.2 Introduction

13.3 Registers

13.4 Serial In—Serial Out(SISO) Register

13.5 Serial In –Parallel Out (SIPO) Register

13.6 Parallel In –Serial Out(PISO) Register

13.7 Parallel In –Parallel Out(PIPO) Register

13.8 Let Us Sum Up

13.9 Answers to Check Your Progress

13.10 Further Reading

13.11 Model Questions

13.1 LEARNING OBJECTIVES

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

- define registers
- describe the working principle of registers
- describe different registers like Serial In - Serial Out, Serial In - Parallel Out, Parallel In - Serial Out, Parallel In - Parallel Out.

13.2 INTRODUCTION

In the previous units, we have learnt about sequential and combinational circuits. In this unit, we will learn about registers. Different types of registers like SISO, SIPO, PISO and PIPO are covered in detail in this unit.

13.3 REGISTERS

A register is a set of flip-flops used to store binary data. The data to be stored is entered or shifted into the register and also taken out or shifted out as per necessity. Hence registers with the facility to shift their contents

right or left or in serial or parallel are known as shift registers.

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)

13.3 SERIAL IN – SERIAL OUT (SISO) REGISTER

Figure 13.1 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.

(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.)

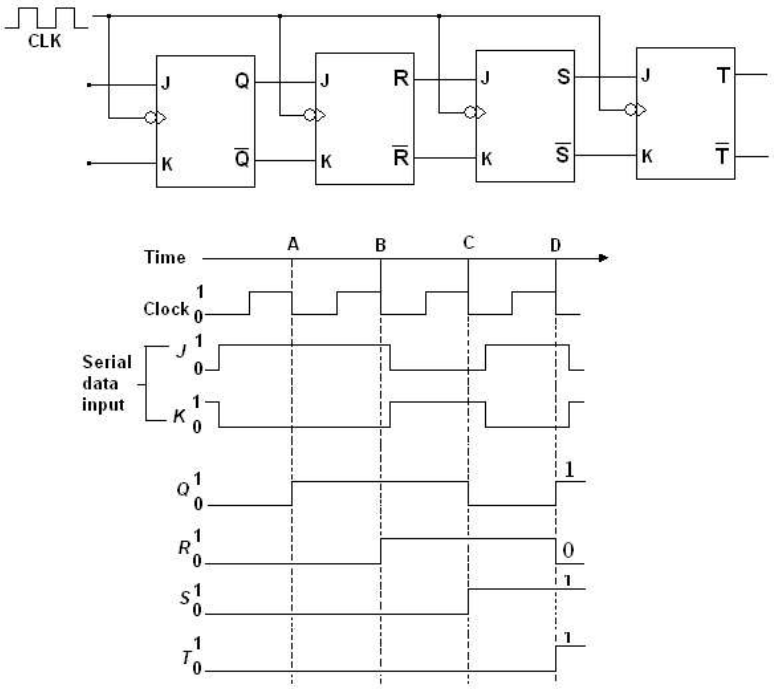


Figure 13.1 : 4 Bit Serial In – Serial Out Shift Register

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 1st first flip-flop. At the negative CLK edge, the O shifts to Q. The 1 of Q shifts into R,1 of R shifts into S, O 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 O 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 O 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.

13.5 SERIAL IN—PARALLEL OUT (SIPO) REGISTER

In this type of shifts register, data is entered serially into the register and once data entry is completed it can be taken out parallely. To take the data parallely, it is simply required, to connect the output of each flip-flop to an output pin. All other constructional features are same as in Serial In—Serial Out (SISO) register.

The shifting of data into SIPO is same as SISO registers. In the SIPO of Figure 13.2 , 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.

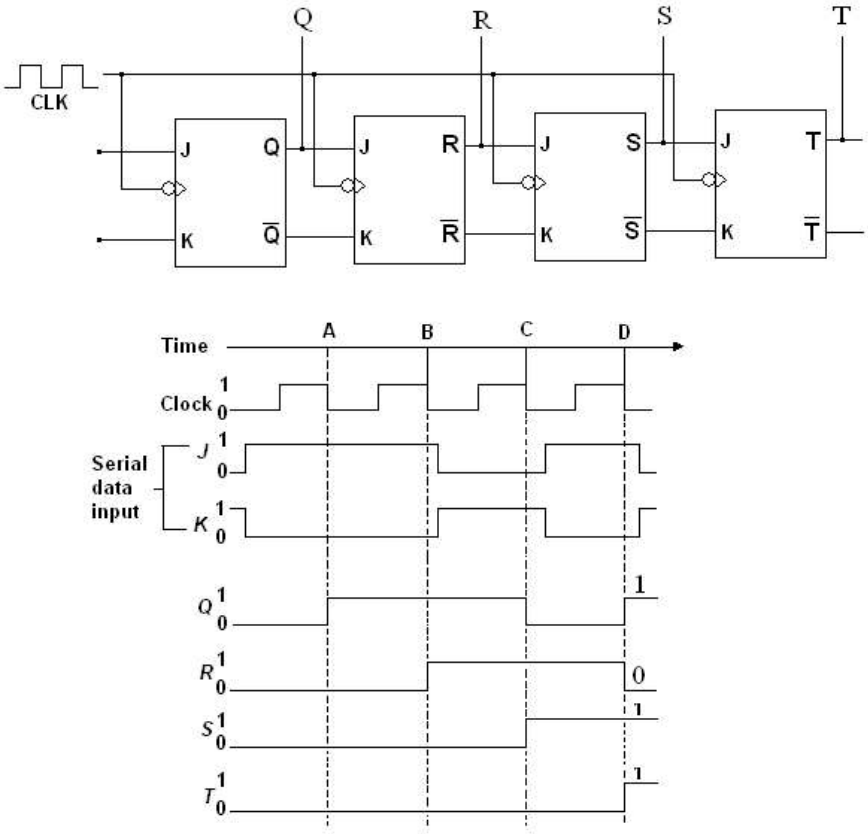


Figure 13.2 : 4 Bit Serial In – Parallel Out Shift Register

13.6 PARALLEL IN-SERIAL OUT (PISO) REGISTER

PISO registers take data parallely and shift 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 Figure 13.3.

Figure 13.3 (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 we add a NOR gate as in Figure 13.3(b). Here, if X_2 is at ground level, X_1 will be inverted by the NOR gate. As for example, if =1, then output of the NOR gate will be =0, thereby a 1 will be clocked into the flip-flop.

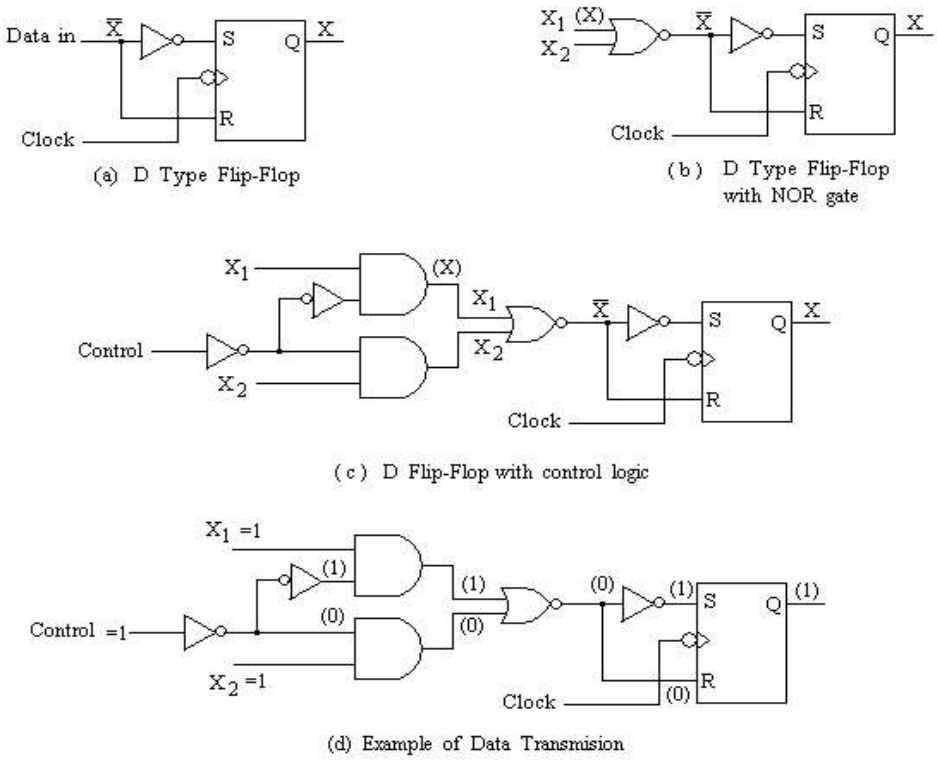
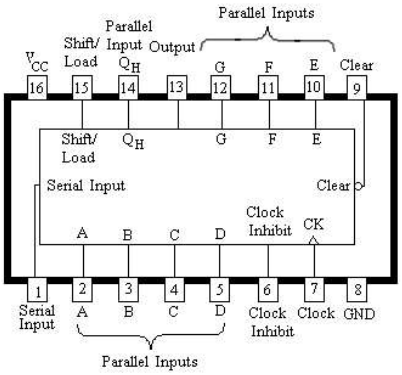


Figure 13.3 : Building Block of Parallel In- Serial Out Register

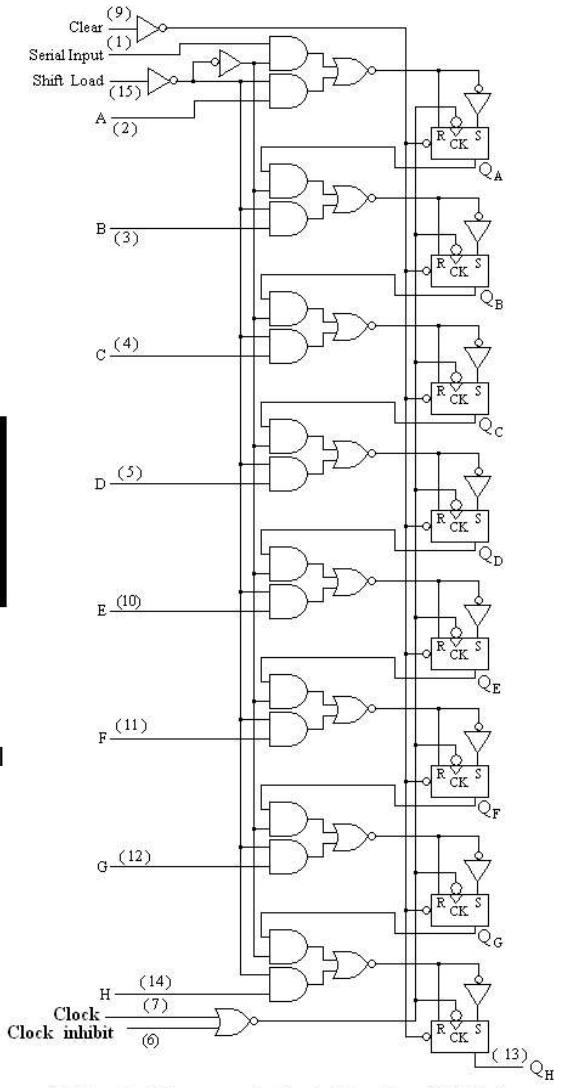
Now in Figure 13.3(c), two AND gates and two NOT gates are added. These will allow the selection of data . If the control line is high, the upper AND gate is enabled and the lower AND gate is disabled. Thus the data will enter at the upper leg of the NOR gate and at the same time the lower leg of the 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.

If we study the Figure 13.4 of PISO, we see that circuit of Figure 13.3 (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.



(a) Pin Out Diagram of Serial In-Parallel Out Register



(b) Logic Diagram of Serial In-Parallel Out Register

Figure 13.4 : Circuit of 54/74166

In Figure 13.4, the X_2 input of Figure 13.3(c) is taken out from each flip-flop to form 8 inputs named as ABCDEFGH to enter 8 bit data. parallely 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 parallely. 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. This means data will be shifted serially.

13.7 PARALLEL IN –PARALLEL OUT (PIPO) REGISTER

The register of Figure 13.4 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 Figure 13.3(c). The parallel data outputs are simply taken out from the Q sides of each flip-flop. In Figure 13.5 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 parallely 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.

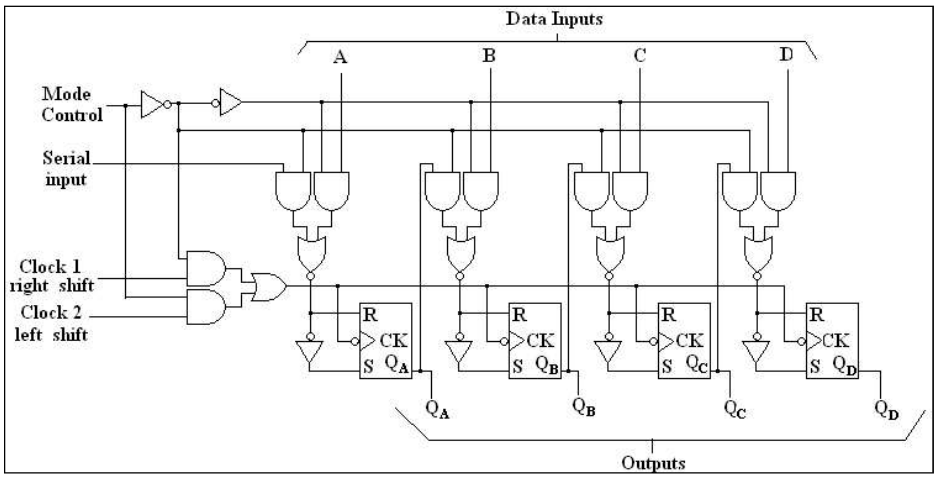


Figure 13.5 : Parallel In- Parallel Out Shift Register



CHECK YOUR PROGRESS

Q.1: Shift registers are.....

(a) basically a sequential circuit

(b) a combinational circuit

(c) permanent memory (d) none of these

Q.2: In SIPO

(a) data enters parallelly and leaves serially

(b) data enters serially and leaves serially

(c) data enters serially and leaves parallely

(d) data enters parallely and leaves parallely



13.8 LET US SUM UP

- A register is a set of flip flop used to store binary data.
- Shift registers are 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.
- PISO registers take data parallelly and shift data serially.
- In SIPO registers, data is entered serially and taken out parallely.



13.9 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1 : (a)

Ans. to Q. No. 2 : (c)



13.10 FURTHER READING

1) Mano, M. M. (2017), *Digital Logic and Computer Design*, Peason Education India.

2) Mano, M. M. (1982), *Computer System Architecture*, Englewood Oliffs, N.J., 53-54.

3) Malvino, A.P. & Leach, P. (1986), *Digital Principles and Applications*, Mcgraw-Hill International

4) Lee, C. (1976), *Digital Circuits and Logic Design*

5) Talukdar, P. (2010), *Digital Techniques*, N. L. Publications

6) Bogart, F. (1992), *Introduction to Digital Circuits*.



13.11 MODEL QUESTIONS

Q 1: What is register? Explain different types of register.

Q.2: 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.

Q.3: What are Shift registers?

Q.4: Describe the working of SISO register.

Q.5: Compare SISO register with PISO registers.

Q.6: Compare PIPO register with PISO registers.

Q.7: Compare SISO register with SIPO registers.

*** ***** ***

UNIT 14 : COUNTERS

UNIT STRUCTURE

14.1 Learning Objectives

14.2 Introduction

14.3 Counters

14.4 Asynchronous Counter

14.5 Synchronous Counter

14.6 Let Us Sum Up

14.7 Answers to Check Your Progress

14.8 Further Reading

14.9 Model Questions

14.1 LEARNING OBJECTIVES

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

- define and describe the working of counters
- describe the different types of counters
- describe asynchronous counter
- describe synchronous counter.

14.2 INTRODUCTION

In the previous units we have learnt about different sequential and combinational circuits. We have also learnt about the different types of registers like SIPO, PIPO, SISO and PISO.

In this unit, you will be introduced to the concept of counters. Different types of counters like asynchronous and synchronous counters are discussed in this unit.

14.3 COUNTERS

A counter is a sequential circuit that goes through a prescribed sequence of states when input pulses, also called count pulses are applied. These pulses may be clocked pulses or they may originate from an external

source. They may also occur at prescribed intervals of time or at random. A counter may follow the sequence of binary count or any other sequence of states. Counters are used for counting the number of occurences of an event and are useful for generating timing sequences to control operations in a digital system.

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 have 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, sometimes it is required to reset all the flip-flops to zero. It is called “Clear”. Sometimes, it is required to set the flip-flops before hand. It is called preset. To do this, two extra inputs are there in every flip-flop called CLR and PR.

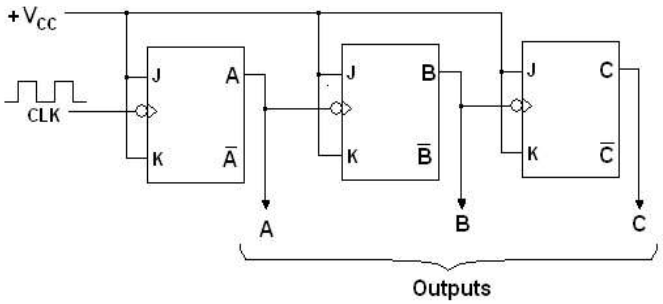
14.4 ASYNCHRONOUS COUNTER

When the output of a flip-flop is used as the clock input for the next flip-flop it is called asynchronous counter.

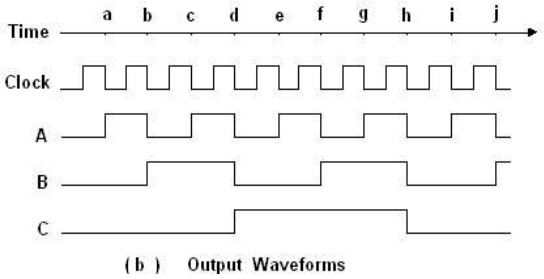
Asynchronous counters are also called ripple counter because flip-flop transitions ripple through from one flip-flop to the next in sequence until all flip-flop reach a new state.

A binary ripple counter can be constructed by using clocked JK flip-flop. Figure 14.1(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 14.1 (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.



(a) Three-bit binary counter



(b) Output Waveforms

Clock transitions	C	B	A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
0	0	0	0

(c) Truth Table

Figure 14.1 : 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 Figure 14.1 (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 then 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.

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.

14.5 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. Figure 14.2 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 the first 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 count enable input to drive the 2^{nd} 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 inputs J and K are steered to each individual flip-flop.

In the figure, the clock pulse is directly applied to all the flip-flops. J and K are both high for the 1st flip-flop, 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 the inputs J and K=1 high to reach the 2^{nd} flip-flop. So the 2^{nd} flip-flop toggles with every other negative clock transition at points b, d, f and h on the time line.

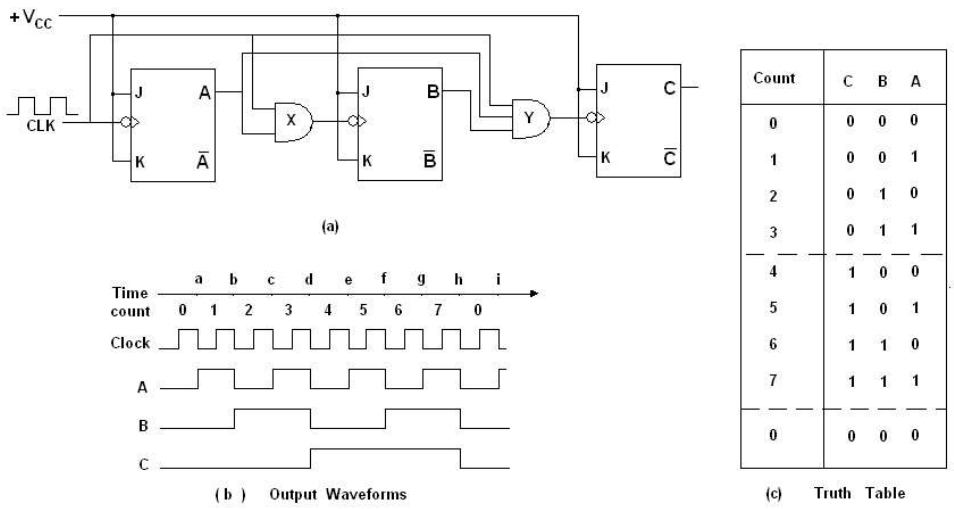


Figure 14.2 : 3 bit Parallel Binary Counter

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 3rd flip-flop. The 3rd 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.1: State *true or false*

(a) Counters are non sequential digital circuits.

(b) Asynchronous counters are fast in operation than synchronous counters.

(c) The total number natural progression of a counter is called MODULUS.

(d) Counters can be used to build digital clock.

Q.2: Define Counters.



14.6 LET US SUM UP

- Modules of a counter is the number of clock pulses that cause a counter to reset its initial state.
- 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.
- Asynchronous counters are also called ripple counters.
- Synchronous counters overcome the delay of asynchronous counters.



14.7 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1 : (a) False, (b) False, (c) True, (d) True

Ans. to Q. No. 2 : A counter is a sequential circuit that goes through a perscribed sequence of states when input pulses are applied.



14.8 FURTHER READING

1) Mano, M. M. (2017), *Digital Logic and Computer Design*, Peason Education India.
2) Mano, M. M. (1982), *Computer System Architecture*, Englewood Oliffs, N.J., 53-54.
3) Malvino, A.P. & Leach, P. (1986), *Digital Principles and Applications*, Mcgraw-Hill International
4) Lee, C. (1976), *Digital Circuits and Logic Design*
5) Talukdar P. (2010), *Digital Techniques*, N. L. Publications
6) Bogart, F. (1992), *Introduction to Digital Circuits*.



14.9 MODEL QUESTIONS

Q.1: What are the differences between asynchronous and synchronous counter? Draw a MODE-8 counter and explain its working principle.

Q.2: What do you mean by modulus of a counter? What is a forced counter?

Q.3: Describe asynchronous counters in detail.

Q.4: Describe synchronous counters in detail.

Q.5: What are ripple counters?

*** ***** ***

UNIT 15: MEMORY ORGANIZATION

UNIT STRUCTUTRE

15.1 Learning Objectives

15.2 Introduction

15.3 Memory Types

15.4 Random Access Memory

15.5 Access Time

15.6 RAM Package

15.7 Static RAM

15.8 CMOS Memory Cell

15.9 Dynamic RAM

15.10 Read Only Memory

15.11 Memory Expansion

15.12 Let Us Sum Up

15.13 Answers to Check Your Progress

15.14 Further Readings

15.15 Model Questions

15.1 LEARNING OBJECTIVE

After going through this unit, you will be able to

- identify the difference between primary and secondary memory.
- explain the characteristics and purpose of Random Access Memory (RAM).
- classify different types of RAM and their features.
- identify and differentiate between the various types of RAM packages.
- explain the advantages and disadvantages of static RAM (SRAM) over other types of memory.
- describe the construction and operation of a CMOS memory cell.
- analyse the suitability of different memory types for various computing applications.

15.2 INTRODUCTION

In today's technologically advanced society, memory is essential for effectively storing and accessing data. The many forms of memory employed in contemporary computing systems are the main topic of this chapter. Topics like Random Access Memory (RAM), its packaging variations, and access times are covered in this unit. It also examines read-only memory (ROM), memory expansion techniques, complementary metal oxide semiconductor (CMOS) memory cells, dynamic RAM (DRAM), and static RAM (SRAM). Learness will have a thorough understanding of the various forms of memory and how they are used in contemporary computing by the end of this unit.

In this unit, we'll start by exploring the various types of memory, including Random Access Memory (RAM), Read Only Memory (ROM), and their differences. We'll also delve into the concept of access time and how it impacts memory performance. Additionally, we'll examine different RAM packages, such as Static RAM (SRAM) and Dynamic RAM (DRAM), as well as the functioning of CMOS memory cells. Lastly, we'll discuss the importance of memory expansion and the benefits it can offer.

15.3 MEMORY TYPES

Memory is the fundamental component that stores data and instructions in a computing system. It is essential for the functioning of any digital device, including computers, smartphones, tablets, and other electronic devices. In general, memory is classified into two categories: primary memory and secondary memory.

1. **Primary Memory:** Primary memory is also known as main memory, internal memory, or primary storage. It is a type of memory that is directly accessible by the CPU (Central Processing Unit) and is used to temporarily store data and instructions that the CPU needs to access quickly. There are two types of primary memory : RAM and ROM.
   - **Random Access Memory (RAM):** RAM is a volatile type of memory that stores data and instructions temporarily. When the

computer is turned off, all the data stored in RAM is lost. The CPU uses RAM to temporarily store data and instructions that are currently being used. It is a fast and temporary storage that allows the CPU to access data quickly. The capacity of RAM varies from device to device, but it is generally measured in gigabytes (GB).

- **Read-Only Memory (ROM) :** ROM is a non-volatile type of memory that stores data and instructions permanently. It is called read-only because the data stored in ROM cannot be modified once it is written. ROM is used to store firmware and other essential programs that are required to boot the computer. The capacity of ROM is generally smaller than that of RAM and is measured in kilobytes (KB) or megabytes (MB).

2. **Secondary Memory:** Secondary memory is also known as external memory or secondary storage. It is a type of memory that is used to store data and instructions permanently. Secondary memory is not directly accessible by the CPU and is usually slower than primary memory. There are several types of secondary memory, including hard disk drives (HDD), solid-state drives (SSD), magnetic tapes, CDs, DVDs, and USB flash drives.
   - **Hard Disk Drives (HDD) :** HDD is a type of secondary memory that stores data on a rotating disk. It uses magnetic storage to store data and instructions. HDD is the most common type of secondary memory used in computers and laptops. The capacity of an HDD varies from a few hundred gigabytes to several terabytes.
   - **Solid-State Drives (SSD) :** SSD is a type of secondary memory that uses NAND flash memory to store data and instructions. It has no moving parts and is faster than HDD. SSD is more expensive than HDD, but it is also more durable and reliable. The capacity of an SSD varies from a few hundred gigabytes to several terabytes.
   - **Magnetic Tapes :** Magnetic tapes are a type of secondary

memory that uses magnetic storage to store data and instructions. They are mainly used for backup and archival purposes. The capacity of magnetic tapes varies from a few gigabytes to several terabytes.

- **CDs and DVDs :** CDs and DVDs are a type of secondary memory that uses optical storage to store data and instructions. They are mainly used for distributing software, music, and movies. The capacity of CDs and DVDs varies from a few hundred megabytes to several gigabytes.
- **USB Flash Drives :** USB flash drives are a type of secondary memory that uses NAND flash memory to store data and instructions. They are portable and can be easily connected to a computer or a laptop. The capacity of USB flash drives varies from a few gigabytes to several terabytes.

Memory is an essential component of any digital device, and it comes in various types, including primary and secondary memory. Primary memory is used to temporarily store data and instructions that the CPU needs to access quickly, while secondary memory is used to store data and instructions permanently. Each type of memory has its advantages and disadvantages, and the choice of memory dependson the specific requirements of the device and the intended use case. As technology continues to evolve, new types of memory are constantly being developed to meet the increasing demands for faster and more efficient storage solutions. Understanding the different types of memory and their characteristics is crucial for individuals involved in computer hardware and software development, as well as for end-users who rely on these devices in their daily lives.

15.4 RANDOM ACCESS MEMORY

Random Access Memory (RAM) is a type of primary memory that is used to temporarily store data and instructions that the CPU (Central Processing Unit) needs to access quickly. RAM is an essential component of any computing device and is used extensively in personal computers,

servers, smartphones, tablets, and other electronic devices.

RAM is a volatile type of memory, which means that the data stored in RAM is lost when the device is turned off or the power is disconnected. This is in contrast to non-volatile memory, such as ROM (Read-Only Memory), where the data is retained even when the power is turned off. RAM is designed to be a fast and temporary storage solution that allows the CPU to access data quickly, reducing the processing time required to retrieve and manipulate data.

The capacity of RAM varies depending on the device and its intended use. In general, the capacity of RAM is measured in gigabytes (GB). For example, a typical personal computer may have 8 GB, 16 GB, or even 32 GB of RAM, while a smartphone or tablet may have 2 GB, 4 GB, or 8 GB of RAM.

There are several types of RAM, including Dynamic RAM (DRAM), Static RAM (SRAM), and Synchronous Dynamic RAM (SDRAM).

- **Dynamic RAM (DRAM):** DRAM is the most common type of RAM used in personal computers and other electronic devices. DRAM requires constant refreshing to maintain the data stored in memory, which is why it is called dynamic. DRAM is relatively cheap and has a high density, which means that a large amount of data can be stored in a small space.
- **Static RAM (SRAM):** SRAM is a type of RAM that does not require constant refreshing, making it faster than DRAM. SRAM is more expensive than DRAM and has a lower density, which means that it can store less data in a given space. SRAM is commonly used in cache memory, which is a small and fast memory used to temporarily store frequently accessed data.
- **Synchronous Dynamic RAM (SDRAM):** SDRAM is a type of DRAM that is synchronized with the system clock, allowing data to be transferred between memory and the CPU more efficiently. SDRAM is faster than conventional DRAM and is commonly used in personal computers and servers.

RAM is installed on the motherboard of the device and is typically in

the form of small rectangular chips called DIMMs (Dual Inline Memory Modules). The number of DIMMs and their capacity vary depending on the device and its intended use.

RAM is an essential component of any computing device, and it plays a crucial role in the performance of the device. Understanding the different types of RAM and their characteristics is important for individuals involved in computer hardware and software development, as well as for end-users who rely on these devices for their daily computing needs.

15.5 ACCESS TIME

Access time is a critical performance metric when it comes to the organization of computer memory. It refers to the time required for a computing device to retrieve data from or write data to a particular memory location. It is determined by several factors, including the type of memory, the memory organization, and the distance between the memory and the CPU. The organization of memory is crucial because it determines how the data is stored and accessed by the CPU.

In general, there are two types of memory organization: sequential and random. In sequential memory organization, data is stored in a contiguous block of memory locations, and access to these locations is performed in a sequential manner. This means that if the CPU wants to access a particular memory location, it must first access all the memory locations that come before it in the sequence.

In contrast, random memory organization allows for non-sequential access to memory locations. In this type of organization, each memory location is assigned a unique address, and the CPU can access any memory location directly by specifying its address. This means that the CPU does not need to access all the memory locations that come before the one it needs, resulting in faster access times.

There are several factors that influence access time in memory organization. These include:

1. **Clock Speed:** The clock speed of the CPU and the memory controller determine how fast the CPU can access memory. A higher clock speed

generally results in faster access times.

2. **Memory Type:** Different types of memory have different access times. For example, SRAM (Static Random Access Memory) has a faster access time than DRAM (Dynamic Random Access Memory).
3. **Bus Width:** The width of the memory bus, which is the path used to transfer data between the CPU and memory, can also affect access time. A wider bus can transfer more data at once, resulting in faster access times.
4. **Latency:** Latency refers to the delay between when the CPU requests data from memory and when the data is actually available. Lower latency results in faster access times.
5. **Cache:** Cache memory is a small, high-speed memory that stores frequently accessed data. By keeping frequently used data in cache, the CPU can access it more quickly than if it had to retrieve it from main memory.

Access time is a critical performance metric in memory organization. The organization of memory plays a crucial role in determining access time, and factors such as clock speed, memory type, bus width, latency, and cache can all affect access time. Understanding the factors that influence access time can help computer hardware and software developers optimize performance and improve the overall user experience.



CHECK YOUR PROGRESS

Q 1: What is RAM?

.....

.....

Q 2: What is the difference between RAM and ROM?

.....

.....

Q 3: What are some types of secondary memory?

.....

.....

15.6 RAM PACKAGE

A RAM package is a term used to refer to the physical module that contains Random Access Memory (RAM) chips. The RAM package is an essential component of a computer system that allows for temporary storage of data and instructions that the computer's CPU can quickly access.

RAM packages come in various shapes, sizes, and types, depending on the specific computer system requirements. Some of the common RAM package types include:

- **Single In-Line Memory Module (SIMM):** This was the first type of RAM package used in personal computers. SIMMs are small circuit boards that have pins on one side and are installed vertically onto the motherboard. They are available in 30-pin and 72-pin configurations and are primarily used in older computers.
- **Dual In-Line Memory Module (DIMM):** DIMMs are the most common RAM package type used in modern computers. They are longer than SIMMs and have pins on both sides. DIMMs come in various configurations, including DDR, DDR2, DDR3, and DDR4, each with different data transfer rates and capacities.
- **Small Outline Dual In-Line Memory Module (SODIMM):** SODIMMs are smaller versions of DIMMs used in laptops and other small form factor devices. They have pins on both sides and are available in DDR, DDR2, DDR3, and DDR4 configurations.
- **Rambus In-Line Memory Module (RIMM):** RIMMs were developed by Rambus Inc. as a high-speed memory solution. They have a unique design that uses a series of RIMM slots on the motherboard and require special Rambus memory controllers to function.
- **Buffered and Registered DIMMs:** Buffered and registered DIMMs are specialized RAM packages that are used in high-performance computing systems. They have additional chips that buffer and register data, which allows for increased reliability and speed.

RAM packages have several specifications that must be considered when selecting a suitable module for a computer system. These include

the type of RAM used, capacity, data transfer rate, latency, and voltage requirements.

When installing a RAM package, it is important to ensure that it is compatible with the computer system's motherboard and other components. The installation process typically involves inserting the RAM package into the appropriate slot and securing it in place using retaining clips or screws.

RAM packages are essential components of modern computer systems that allow for temporary storage of data and instructions. They come in various types and configurations, including SIMMs, DIMMs, SODIMMs, RIMMs, and buffered and registered DIMMs. Understanding the different RAM package types and specifications is crucial when selecting a suitable module for a computer system.

15.7 STATIC RAM

Static RAM (SRAM) is a type of computer memory that uses flip-flops to store data. SRAM is volatile memory, which means that its contents are lost when power is removed from the system. SRAM is faster than dynamic RAM (DRAM) and has lower power consumption, making it ideal for use in cache memory and other applications where fast access times are crucial.

SRAM is also called static random access memory (SRAM), as it retains its data without requiring periodic refreshes, unlike dynamic RAM (DRAM). SRAM is constructed from a set of transistors arranged in a matrix pattern to form a memory cell. Each memory cell is capable of storing one bit of information. The transistor pairs in the memory cell are cross-coupled, which enables the cell to maintain its state even when not being accessed. Static RAM has several advantages over other types of memory, including:

- **High speed:** SRAM is much faster than dynamic RAM because it does not need to be refreshed constantly. This makes it ideal for use in cache memory and other applications where fast access times are essential.
- **Low power consumption:** SRAM uses less power than other types

of memory because it does not require constant refreshing.

- **Durability:** SRAM is more durable than other types of memory because it does not wear out over time.
- **Easy to interface:** SRAM is easy to interface with other digital circuits because it uses standard logic levels.

However, SRAM also has some disadvantages, including:

- **Higher cost:** SRAM is more expensive than other types of memory because it requires more transistors to store a single bit of information.
- **Limited capacity:** SRAM has limited capacity because it requires more transistors per bit than other types of memory.
- **Susceptible to noise:** SRAM is susceptible to noise and other forms of interference, which can cause data errors.

Overall, SRAM is an important type of computer memory that offers high speed, low power consumption, and durability. It is used in a wide range of applications, including cache memory, microprocessors, and other digital circuits. The choice of memory type depends on the specific requirements of the application, and SRAM is a popular choice for many high-performance computing systems.

15.8 CMOS MEMORY CELL

A Complementary Metal-Oxide-Semiconductor (CMOS) memory cell is a type of memory cell that is commonly used in integrated circuits. CMOS memory cells are designed to consume low power and have a small cell size. They are used in a wide range of applications, including microprocessors, digital signal processors, and other types of digital circuits.

The basic structure of a CMOS memory cell consists of a pair of transistors: a p-type metal-oxide-semiconductor field-effect transistor (PMOS) and an n-type metal-oxide-semiconductor field-effect transistor (NMOS). The PMOS and NMOS are arranged in a complementary configuration, which means that when one transistor is off, the other is on. This allows for the cell to be designed to consume very little power.

The CMOS memory cell operates in two states: writing and reading. During the writing state, the input data is stored in the cell. This is

accomplished by applying a voltage to the input, which charges or discharges the capacitor formed by the PMOS and NMOS transistors. The charging or discharging of the capacitor determines whether the cell is storing a “0” or a “1”.

During the reading state, the output of the memory cell is determined by the voltage on the capacitor. This voltage is amplified and read out by a sense amplifier. The voltage on the capacitor is maintained by the PMOS and NMOS transistors, which are designed to hold the charge for long periods of time.

One of the main advantages of CMOS memory cells is their low power consumption. This is because they use very little power when in the standby mode, and only consume power when data is being written or read. The small size of the cell also makes it ideal for use in large-scale integrated circuits (LSIs), which require a large number of memory cells to be packed into a small space.

In summary, a CMOS memory cell is a type of memory cell that is commonly used in integrated circuits. The cell is designed to consume low power and has a small cell size, making it ideal for use in LSIs. The basic structure of the cell consists of a pair of complementary transistors that operate in two states: writing and reading. The low power consumption and small cell size of CMOS memory cells make them an important component of modern digital circuits.

15.9 DYNAMIC RAM

Dynamic Random Access Memory (DRAM) is a type of computer memory that is commonly used in computers and other digital devices. DRAM is a volatile memory, which means that its contents are lost when power is removed from the system. DRAM stores each bit of data in a capacitor within an integrated circuit, and it needs to be refreshed regularly to prevent the loss of data.

The DRAM cell consists of a capacitor and a transistor. The capacitor stores the charge that represents the bit of data, and the transistor controls access to the capacitor. The capacitor is made up of two conducting plates,

separated by a dielectric material. The charge on the capacitor represents the bit of data, with a high charge representing a “1” and a low charge representing a “0”.

DRAM has several advantages over other types of memory, including:

- High density: DRAM is capable of storing a large amount of data in a small space, making it an ideal choice for use in computers and other digital devices.
- **Low cost:** DRAM is relatively inexpensive compared to other types of memory, which makes it a popular choice for many applications.
- **Low power consumption:** DRAM uses less power than other types of memory, which makes it ideal for use in portable devices such as laptops and mobile phones.

However, DRAM also has some disadvantages, including:

- **Slower access times:** DRAM has slower access times compared to other types of memory, which can slow down the overall performance of the system.
- **Limited lifespan:** DRAM has a limited lifespan due to the nature of the capacitor, which can degrade over time and lose its ability to hold a charge.
- **Requires refreshing:** DRAM requires regular refreshing to maintain the charge on the capacitor, which can add to the complexity and cost of the system.

Overall, DRAM is an important type of computer memory that is widely used in digital devices. The choice of memory type depends on the specific requirements of the application, and DRAM is a popular choice for many general-purpose computing systems due to its high density and low cost.



CHECK YOUR PROGRESS

Q 4: What is Static RAM (SRAM)?

.....

.....

Q 5: What is a CMOS memory cell?

.....

.....

Q 6: What are some common types of RAM packages? Answer: Some common types of RAM packages include SIMMs, DIMMs, SODIMMs, RIMMs, and buffered and registered DIMMs.

.....

.....

15.10 READ ONLY MEMORY

Read Only Memory (ROM) is a type of non-volatile memory that is used in computers and other digital devices to store data that cannot be modified or erased. Unlike RAM, which is volatile and loses its data when power is turned off, ROM retains its data even when the power is off.

ROM is primarily used to store firmware, which is the software that is permanently embedded in a device's hardware. Firmware typically includes the operating system, drivers, and other low-level software that is necessary for the device to function properly.

There are several different types of ROM, including:

- **Mask ROM :** This is the earliest type of ROM, which is manufactured with a specific data pattern. Once the pattern is programmed, it cannot be changed.
- **Programmable ROM (PROM) :** This type of ROM can be programmed by the user once. Once the programming is done, the data cannot be changed.
- **Erasable Programmable ROM (EPROM):** This type of ROM can be erased and reprogrammed multiple times by using ultraviolet light to erase the data.
- **Electrically Erasable Programmable ROM (EEPROM):** This type of ROM can be erased and reprogrammed electronically.

ROM has several advantages over other types of memory, including:

- **Non-volatile :** ROM retains its data even when the power is off, which makes it ideal for storing critical data that must be available at all times.
- **High reliability :** ROM is less susceptible to data corruption and errors

than other types of memory, which makes it a reliable choice for storing critical data.

- **Low power consumption:** ROM uses very little power, which makes it ideal for use in portable devices.

However, ROM also has some disadvantages, including:

- **Limited capacity:** ROM typically has a limited capacity, which can limit its usefulness in certain applications.
- **Inflexible:** The data stored in ROM cannot be changed or modified, which can be a limitation in certain applications.

Overall, ROM is an important type of memory that is widely used in digital devices. The choice of memory type depends on the specific requirements of the application, and ROM is a popular choice for storing critical data that must be available at all times.

15.11 MEMORY EXPANSION

Memory expansion refers to the process of increasing the amount of memory available to a computer system. Memory expansion is a common approach used to increase the storage capacity of a computer, and is used in a variety of different contexts, including for personal computers, servers, and embedded systems.

Because it enables the development of bigger, more complex applications that need more memory to execute, memory expansion is a crucial factor in the organisation of computer memory. Memory expansion is normally accomplished by adding more memory modules to a computer system, either by replacing the old modules with larger ones or by inserting the new modules into the expansion slots that are available.

To extend the amount of memory that a computer system has accessible, a variety of memory expansion techniques can be used. Among the most popular methods are:

- **RAM expansion:** This involves adding more RAM to a computer system, which provides additional memory space for running applications and storing data. RAM expansion can be achieved by adding additional RAM modules to available expansion slots, or by

replacing existing modules with larger ones.

- **Virtual memory expansion**: This involves using hard disk space to supplement the available RAM in a computer system. Virtual memory expansion is typically used when a computer system runs out of physical memory, and is able to provide additional memory space by temporarily storing data in the hard disk.
- **Cache expansion:** This involves increasing the size of the cache memory in a computer system, which provides faster access to frequently used data and instructions. Cache expansion is typically achieved by adding additional cache memory modules to a computer system, or by replacing existing modules with larger ones.
- **Disk expansion:** This involves adding more hard disk space to a computer system, which provides additional storage capacity for data and applications. Disk expansion can be achieved by adding additional hard disks to a computer system, or by replacing existing disks with larger ones.

The selection of a memory extension method will be based on the particular needs of the computer system and the running applications. The best method for memory extension will depend on a number of factors, including cost, performance, and compatibility.

The ability to create larger, more complex applications that require more memory makes memory expansion a crucial factor in how computer memory is organised. RAM expansion, virtual memory expansion, cache expansion, and disc expansion are just a few of the methods that can be used to increase memory. Each method has pros and cons of its own.



CHECK YOUR PROGRESS

Q 7: What is ROM and what is it primarily used for?

.....

.....

Q 8: What is memory expansion?

.....

.....



15.12 LET US SUM UP

- Memory is a fundamental component that stores data and instructions in a computing system, and it is essential for the functioning of any digital device.
- Memory is classified into two categories: primary memory (RAM and ROM) and secondary memory (HDD, SSD, magnetic tapes, CDs, DVDs, and USB flash drives).
- Primary memory is used to temporarily store data and instructions that the CPU needs to access quickly, while secondary memory is used to store data and instructions permanently.
- RAM is a volatile type of primary memory that is used to temporarily store data and instructions that the CPU needs to access quickly. RAM is an essential component of any computing device and is used extensively in personal computers, servers, smartphones, tablets, and other electronic devices.
- There are several types of RAM, including Dynamic RAM (DRAM), Static RAM (SRAM), and Synchronous Dynamic RAM (SDRAM).
- HDD is the most common type of secondary memory used in computers and laptops, while SSD is faster and more durable but more expensive than HDD.
- Understanding the different types of memory and their characteristics is crucial for individuals involved in computer hardware and software development, as well as for end-users who rely on these devices in their daily lives.
- Static RAM (SRAM) is a type of computer memory that uses flip-flops to store data and does not require constant refreshing, making it faster and more power-efficient than dynamic RAM (DRAM).

- SRAM is more durable than other types of memory, but it is more expensive and has limited capacity.
- Complementary Metal-Oxide-Semiconductor (CMOS) memory cells are used in a wide range of applications and are designed to consume low power and have a small cell size.
- The basic structure of a CMOS memory cell consists of a pair of complementary transistors that operate in two states: writing and reading.
- Dynamic Random Access Memory (DRAM) is a volatile memory that needs to be refreshed regularly to prevent the loss of data, but it is capable of storing a large amount of data in a small space and is relatively inexpensive compared to other types of memory.
- The choice of memory type depends on the specific requirements of the application, and each type of memory has its own advantages and disadvantages that should be considered when selecting the appropriate type.



15.13 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q. No 1: RAM is a type of primary memory used to temporarily store data and instructions that the CPU needs to access quickly.

Ans to Q. No 2: RAM is volatile memory used for temporary storage, while ROM is non-volatile memory used for permanent storage.

Ans to Q. No 3: Some types of secondary memory include hard disk drives (HDD), solid-state drives (SSD), magnetic tapes, CDs, DVDs, and USB flash drives.

Ans to Q. No 4: Static RAM (SRAM) is a type of computer memory that uses flip-flops to store data and retains its data without requiring periodic refreshes.

Ans to Q. No 5: A CMOS memory cell is a type of memory cell that consists of a pair of transistors arranged in a complementary configuration,

designed to consume low power and have a small cell size.

Ans to Q. No 6: Some common types of RAM packages include SIMMs, DIMMs, SODIMMs, RIMMs, and buffered and registered DIMMs.

Ans to Q. No 7: ROM is a type of non-volatile memory used to store data that cannot be modified or erased, and it is primarily used to store firmware, including the operating system, drivers, and low-level software required for a device to function properly.

Ans to Q. No 8: Memory expansion refers to the process of increasing the amount of memory available to a computer system, and it can be accomplished by adding more memory modules, using virtual memory, increasing cache memory, or adding more hard disk space.



15.14 FURTHER READINGS

1) Hennessy, J. L., & Patterson, D. A. (2017). Computer architecture: A quantitative approach. Morgan Kaufmann Publishers.

2) Tanenbaum, A. S., & Goodman, J. R. (2014). Structured computer organization. Prentice Hall.



15.15 MODEL QUESTIONS

Q.1. What are the two main categories of memory in computing systems, and how do they differ in terms of their function and purpose?

Q.2. Explain the differences between Dynamic RAM (DRAM), Static RAM (SRAM), and Synchronous Dynamic RAM (SDRAM) in terms of their performance, cost, and usage.

Q.3. What are the advantages and disadvantages of using HDD and SSD as types of secondary memory in computers and laptops, and how do they compare in terms of speed, durability, and cost?

Q.4. Describe the basic structure of a Complementary Metal-Oxide-Semiconductor (CMOS) memory cell, and explain its advantages and

limitations compared to other types of memory.

Q.5. Why is Random Access Memory (RAM) an essential component of any computing device, and what are the benefits of using different types of RAM in various electronic devices such as personal computers, servers, smartphones, and tablets?

Q.6. How does the choice of memory type affect the performance and functionality of computing systems, and what factors should be considered when selecting the appropriate type of memory for a particular application or device?

*** ***** ***