GCA(S1) 03-01

Digital Techniques

SEMESTER - 1

BACHELOR OF COMPUTER APPLICATION

Block 1



KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Anjana Kakati Mahanta, Gauhati University
Prof. (Retd.) Pranhari Talukdar, Gauhati University
Dr. Jyotiprokash Goswami, Assam Engineering College

Course Co-ordinator :

Dr. Sanjib Kr. Kalita, KKHSOU
Dr. Tapashi Kashyap Das, KKHSOU
Sruti Sruba Bharali, KKHSOU

SLM Preparation Team

UNITS	CONTRIBUTORS
1, 2, 3, 4 & 6	Dr. Sangeeta Kakoty, Assam Down Town University
5	Dr. Nabankur Pathak, KKHSOU

Editorial Team

Content : Jyotiprokash Goswami, Assam Engineering College
Language : Prof. (Retd.) Robin Goswami, Cotton College
Structure, Format & Graphics: Dr. Tapashi Kashyap Das, Sruti Sruba Bharali, KKHSOU

June, 2018

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State University is made available under a Creative Commons Attribution-Non Commercial-ShareAlike4.0 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.

Head Office : Patgaon, Rani Gate, Guwahati-781 017; Web: www.kkhsou.in
Sub-Head Office : Housefed Complex, Dispur, Guwahati-781 006

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

BACHELOR OF COMPUTER APPLICATION

Digital Techniques

DETAILED SYLLABUS

Block-1

UNIT 1: Introduction to Number Systems **Pages: 7-21**

Number System; Categories of Number System: Decimal Number System, Binary Number System, Octal Number System, Hexadecimal Number System; Number System Conversion: Binary to Decimal Conversion, Decimal to Binary Conversion, Octal to Decimal Conversion, Decimal to Octal Conversion, Octal to Binary Conversion, Binary to Octal Conversion, Hexadecimal to Decimal Conversion, Decimal to Hexadecimal Conversion, Hexadecimal to Binary Conversion, Binary to Hexadecimal Conversion

UNIT 2: Binary Arithmetic **Pages: 22-35**

Sign Number Representation; Sign Magnitude; Complement of Numbers: (r–1)'s Complement, r's Complement, 1's Complement, 2's Complement; Binary Arithmetic: Addition, Subtraction, Multiplication, Division

UNIT 3: Data Representation **Pages: 36-45**

Data Representation: Fixed Point Representation, Floating Point Representation; IEEE 746 Standards

UNIT 4: Code Coding Techniques **Pages: 46-55**

BCD; ASCII; EBCDIC; Gray Code: Conversion from Binary to Gray Code, Conversion from Gray to Binary Code

UNIT 5: Boolean Algebra **Pages: 56-61**

Boolean Algebra; Properties of Boolean Algebra; De-Morgan's Theorem

UNIT 6: Logic Gates **Pages: 62-76**

Logic Gates: Truth Table, AND Gate, OR Gate, NOT Gate, NAND Gate, NOR Gate, XOR Gate, XNOR Gate; Conversion of Logic Gates

COURSE INTRODUCTION

Digital techniques are the basis of electronic systems, such as computers and cell phones. Digital techniques are founded on binary code, which represents information in the form of zeros and ones. This system facilitates the design of electronic circuits that convey information, including logic gates with functions that include AND, OR, and NOT commands. Learning digital techniques provides an understanding to combinational and sequential circuits, adders, subtractors, flip flops, registers and counters.

This course on **Digital Techniques** comprises of eleven units and seeks to present an overview of the digital representations used in computer system. This course includes an introduction to number systems, binary arithmetic and data representation. The concepts of Boolean algebra and logic gates are also introduced in this course. Apart from these, important topics like reduction techniques, sequential and combinational circuits have also been included in this course. The course also contains detailed topics like registers and counters.

The course is divided into two blocks:

Block 1 introduces the fundamentals of number systems, binary arithmetic and data representation. Code coding techniques are also included in this block. Concepts of Boolean algebra and logic gates are also covered in this block.

Block 2 concentrates on the reduction techniques of Boolean expressions. Combinational and sequential circuits concepts are introduced in this block. In addition to this, registers and counters are also included in this block.

BLOCK INTRODUCTION

This is the *First Block* of the course '***Digital Techniques***'. After completing this block, you will be able to solve numerical problems of Digital Techniques.

This block comprises of the following six units:

Unit 1: introduces you to the different number systems used in computer system. Decimal, Binary, Octal and Hexadecimal number systems are discussed in this unit.

Unit 2: concentrates on the concept of binary arithmetic like addition, subtraction, multiplication and division.

Unit 3: deals with the concepts of representing decimal numbers. Fixed point and floating point representation of numbers are discussed in this unit.

Unit 4: concentrates on code coding techniques. This unit includes the concepts of Gray code, BCD, Excess-3 code and their conversion from one form to another.

Unit 5: introduces concepts on Boolean algebra, its properties and De Morgan's theorem.

Unit 6: includes the basic concepts of logic gates. Logic gates like AND, OR, NOT, NAND, NOR and XOR are discussed in this unit.

The structure of Block 1 is as follows:

UNIT 1: Introduction to Number Systems
UNIT 2: Binary Arithmetic
UNIT 3: Data Representation
UNIT 4: Code Coding Techniques
UNIT 5: Boolean Algebra
UNIT 6: Logic Gates

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 PPROGRESS" 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.

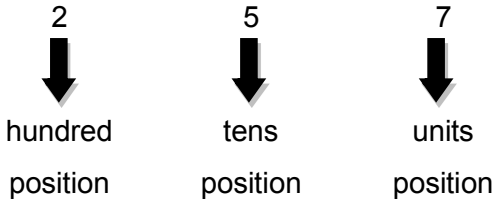
1.4 CATEGORIES OF NUMBER SYSTEM

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:

	Remainder	
2 |75		
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:

	Remainder	MSB	.75	
2|12				
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: What do you mean by 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

1

1 0000000

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 CODING TECHNIQUES

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.

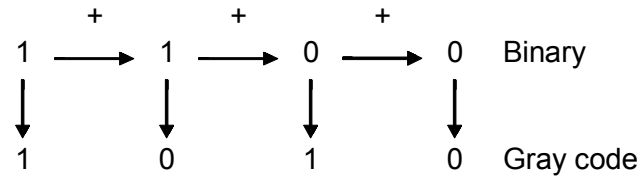
Now, by adding each pair of adjacent bits of Binary code:

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:

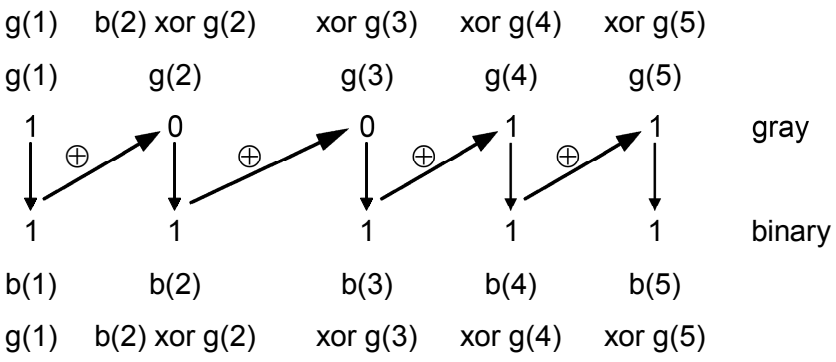
Now: (MSB) 1 + 0 (2^{nd} bit position of gray code) = 1 (2^{nd} bit position of resultant binary code). In the same way,

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.

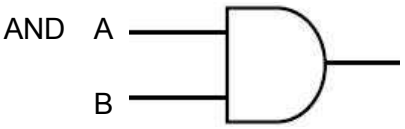
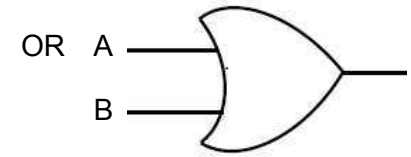
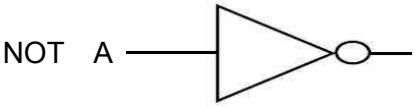
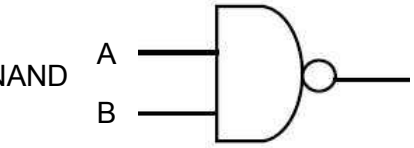
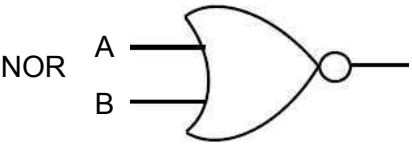
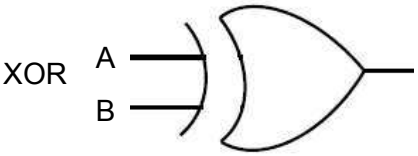
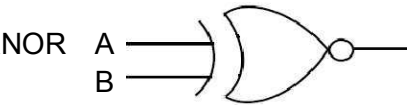
6.3 LOGIC GATES

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 = $\overline{(A.B)}$ or F = (AB)′	A B F 0 0 1 0 1 1 1 0 1 1 1 0
NOR		F = $\overline{(A+B)}$ or F = (A+B)’	A B F 0 0 1 0 1 0 1 0 0 1 1 0
XOR		F = A ⊕ 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		F = A B or F = AB + A′B′	A B F 0 0 1 0 1 0 1 0 0 1 1 1

(Exclusive NOR)

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

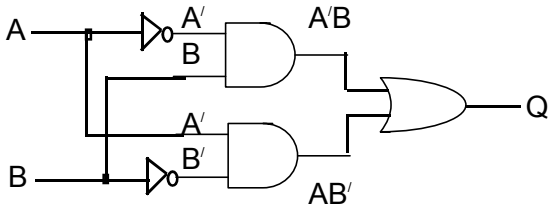
6.3.1 Truth Table

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.

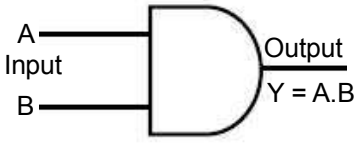


Fig. 6.1: AND Gate

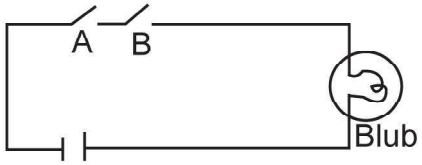


Fig. 6.2: A circuit containing two switches in series

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.

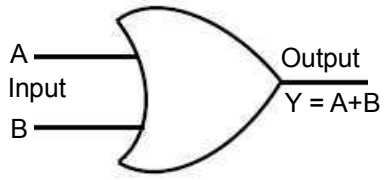


Fig. 6.3: Grapical Symbol OR Gate

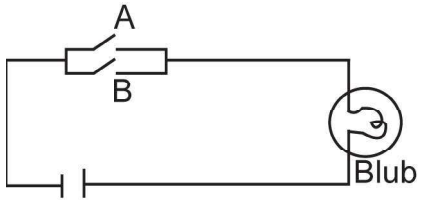


Fig. 6.4: A Circuit containing two switches in parallel

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.

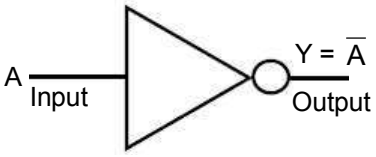


Fig. 6.5: Graphical Symbol NOR gate

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.

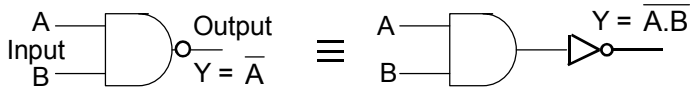


Fig. 6.6: Graphical Symbol NAND Gate

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.

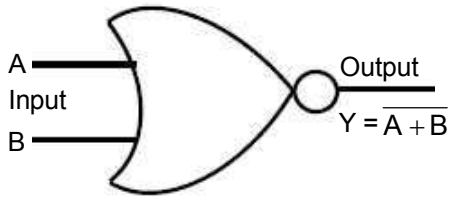


Fig. 6.7: NOR gate

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

$Y = A \oplus B$

$= AB + AB$

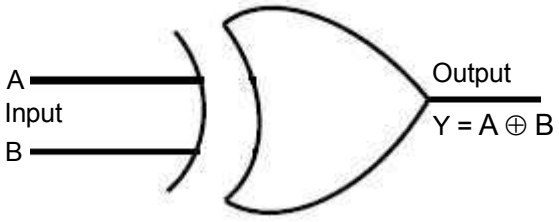


Fig. 6.8 (a): EXCLUSIVE-OR gate

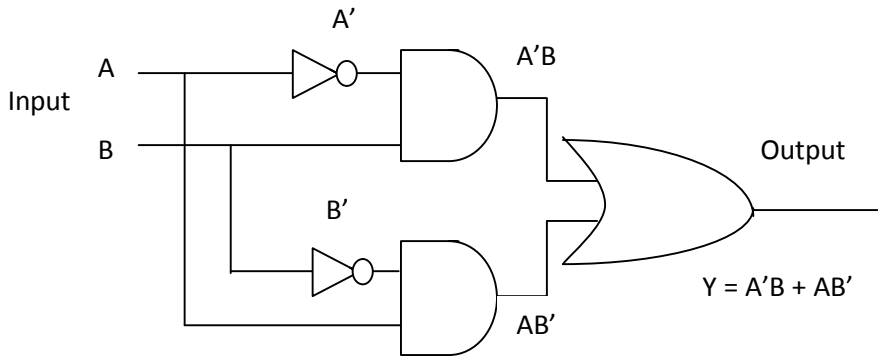


Fig. 6.8 (b): Logic diagram for a 2-input EXCLUSIVE-OR gate

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

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

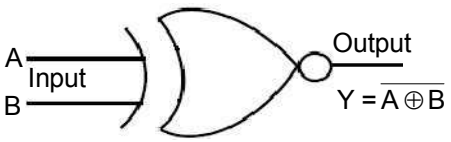


Fig. 6.9: Graphical symbol XNOR Gate

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

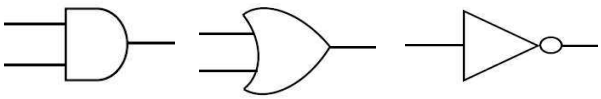
- A logic gate is an electronic circuit that operates on one or more input signals to produce standard output signals. The logic gates are the building blocks of all electronic circuits in computer.
- To represent different logic gates, specific shapes are assigned. These are called graphical symbols.
- 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:

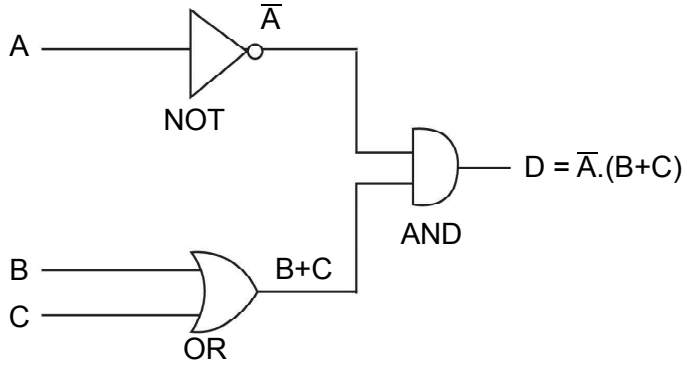


AND gate OR gate NOT gate

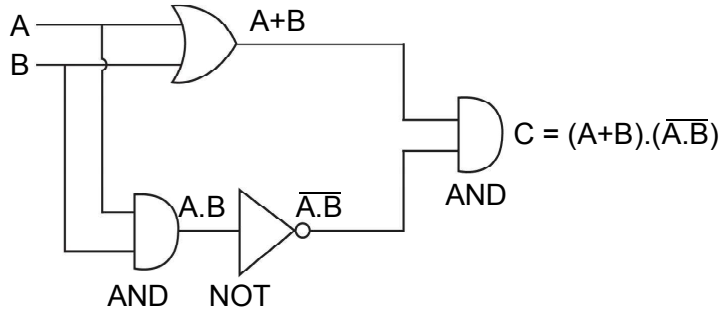
Ans. to Q. No. 3: Truth table for a 3-input OR gate is as follows:

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.

*** ***** ***