

Learning Objectives

- To know how the computer interprets and stores data in the memory.
- To learn various data representations and binary arithmetic.
- To learn conversion between various Number Systems.

2.1 Introduction

The term data comes from the word **datum**, which means a raw fact. The data is a fact about people, places or some objects.

Example:

Let 'Name', 'Age', 'Class', 'Marks' and 'Subject' be some defined variables. Now, let us assign a value to each of these variables.

Name	=	Rajesh
Age	=	16
Class	=	XI
Mark	=	65
Subject	=	Computer Science

Figure 2.1 Example for Data

In the above example, the values assigned to the five different variables are called **data**. When the above data is processed, we get an information "Rajesh is 16 years old, studying in Class XI, has scored 65 marks in Computer Science subject".

Number Systems

2.2 Data Representations

Computer handles data in the form of '0' (Zero) and '1' (One). Any kind of data like number, alphabet, special character should be converted to '0' or '1' which can be understood by the Computer. '0' and '1' that the Computer can understand is called **Machine language**. '0' or '1' are called '**Binary Digits**' (BIT). Therefore, the study of data representation in the computer is important.

- A **bit** is the short form of **Binary digit** which can be '0' or '1'. It is the basic unit of data in computers.
- A **nibble** is a collection of 4 bits (Binary digits).
- A collection of 8 bits is called **Byte**. A byte is considered as the basic unit of measuring the memory size in the computer.
- **Word length** refers to the number of bits processed by a Computer's CPU. For example, a word length can have 8 bits, 16 bits, 32 bits and 64 bits (Present day Computers use 32 bits or 64 bits)

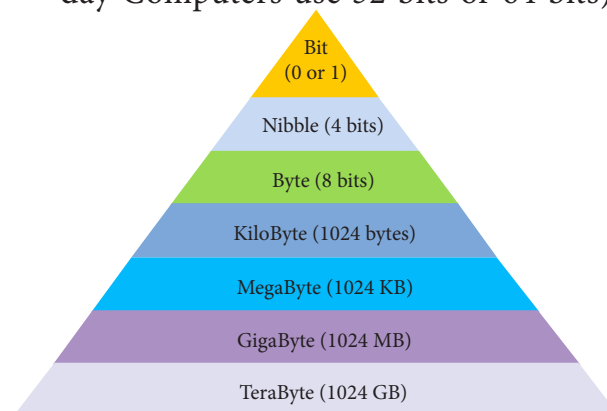


Figure 2.2 Data Representation

Computer memory (Main Memory and Secondary Storage) is normally represented in terms of KiloByte (KB) or MegaByte (MB). In decimal system, 1 Kilo

represents 1000, that is , 10^3 . In binary system, 1 KiloByte represents 1024 bytes that is 2^{10} . The following table represents the various memory sizes:

Table 2.1 Memory Size (Read 2^{10} as 2 power 10)

Name	Abbr.	Size
Kilo	K	$2^{10} = 1,024$
Mega	M	$2^{20} = 1,048,576$
Giga	G	$2^{30} = 1,073,741,824$
Tera	T	$2^{40} = 1,099,511,627,776$
Peta	P	$2^{50} = 1,125,899,906,842,624$
Exa	E	$2^{60} = 1,152,921,504,606,846,976$
Zetta	Z	$2^{70} = 1,180,591,620,717,411,303,424$
Yotta	Y	$2^{80} = 1,208,925,819,614,629,174,706,173$



Bytes are used to represent characters in a text. Different types of coding schemes are used to represent the character set and numbers. The most commonly used coding scheme is the **American Standard Code for Information Interchange** (ASCII). Each binary value between 0 and 127 is used to represent a specific character. The ASCII value for (blank space) is 32 and the ASCII value of numeric 0 is 48. The range of ASCII values for lower case alphabets is from 97 to 122 and the range of ASCII values for the upper case alphabets is 65 to 90.



The speed of a computer depends on the number of bits it can process at once. For example, a 64-bit computer can process 64-bit numbers in one operation, while a 32-bit computer break 64-bit numbers down into smaller pieces, making it slower.

2.3 Different Types of Number Systems

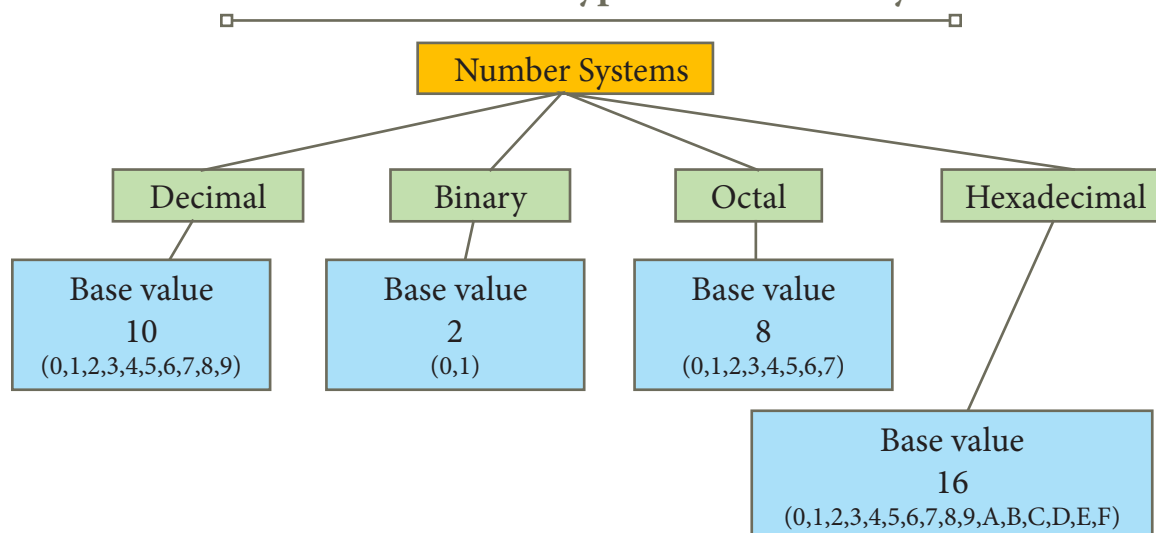


Figure 2.3. Number Systems

A numbering system is a way of representing numbers. The most commonly used numbering system in real life is Decimal number system. Other number systems are Binary, Octal, Hexadecimal number system. Each number system is uniquely identified by its **base value** or **radix**. Radix or base is the count of number of digits in each number system. Radix or base is the general idea behind positional numbering system.

2.3.1 Decimal Number System

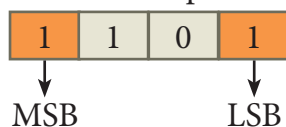
It consists of 0,1,2,3,4,5,6,7,8,9(10 digits). It is the oldest and most popular number system used in our day to day life. In the positional number system, each decimal digit is weighed relative to its position in the number. This means that each digit in the number is multiplied by 10 raised to a power corresponding to that digit's position.

Example

$$\begin{aligned}(123)_{10} &= 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 \\ &= 100 + 20 + 3 \\ &= (123)_{10}\end{aligned}$$

2.3.2 Binary Number System

There are only two digits in the Binary system, namely, 0 and 1. The numbers in the binary system are represented to the base 2 and the positional multipliers are the powers of 2. The left most bit in the binary number is called as the **Most Significant Bit (MSB)** and it has the largest positional weight. The right most bit is the **Least Significant Bit (LSB)** and has the smallest positional weight.



Example

The binary sequence $(1101)_2$ has the decimal equivalent:

$$\begin{aligned}(1101)_2 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 8 + 4 + 0 + 1 \\ &= (13)_{10}\end{aligned}$$

2.3.3 Octal Number System

Octal number system uses digits 0,1,2,3,4,5,6 and 7 (8 digits). Each octal digit has its own positional value or weight as a power of 8.

Example

The Octal sequence $(547)_8$ has the decimal equivalent:

$$\begin{aligned}(547)_8 &= 5 \times 8^2 + 4 \times 8^1 + 7 \times 8^0 \\ &= 5 \times 64 + 4 \times 8 + 7 \times 1 \\ &= 320 + 32 + 7 \\ &= (359)_{10}\end{aligned}$$

2.3.4 Hexadecimal Number System

A hexadecimal number is represented using base 16. Hexadecimal or Hex numbers are used as a shorthand form of binary sequence. This system is used to represent data in a more compact manner. Since 16 symbols are used, 0 to F, the notation is called hexadecimal. The first 10 symbols are the same as in the decimal system, 0 to 9 and the remaining 6 symbols are taken from the first 6 letters of the alphabet sequence, A to F, where A represents 10, B is 11, C is 12, D is 13, E is 14 and F is 15.

Table 2.2 Binary, Octal, Hexadecimal equivalent of Decimal Numbers

Decimal	Binary	Octal	Hexadecimal
0	0000	000	0000
1	0001	001	0001
2	0010	002	0002
3	0011	003	0003
4	0100	004	0004
5	0101	005	0005
6	0110	006	0006
7	0111	007	0007
8	1000	010	0008
9	1001	011	0009
10	1010	012	A
11	1011	013	B
12	1100	014	C
13	1101	015	D
14	1110	016	E
15	1111	017	F

Example

The hexadecimal sequence $(25)_{16}$ has the decimal equivalent:

$$\begin{aligned}
 (25)_{16} &= 2 \times 16^1 + 5 \times 16^0 \\
 &= 32 + 5 \\
 &= (37)_{10}
 \end{aligned}$$

Workshop



1. Identify the number system for the following numbers

S. No.	Number	Number system
1	$(1010)_{10}$	Decimal Number system
2	$(1010)_2$	
3	$(989)_{16}$	
4	$(750)_8$	
5	$(926)_{10}$	

2. State whether the following numbers are valid or not. If invalid, give reason.

S.No.	Statement	Yes / No	Reason (If invalid)
1.	786 is an Octal number		
2.	101 is a Binary number		
3.	Radix of Octal number is 7		

2.4 Number System Conversions

2.4.1 Decimal to Binary Conversion

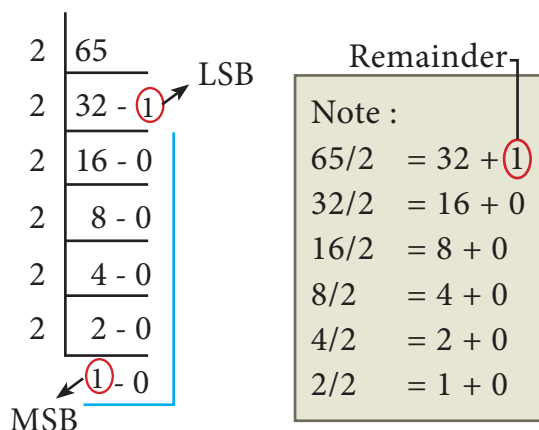
Generally two methods followed.

Method 1: To convert Decimal to Binary “Repeated Division by 2” method can be used. Any Decimal number divided by 2 will leave a remainder of 0 or 1. Repeated division by 2 will leave a sequence of 0s and 1s that become the binary equivalent of the decimal number. Suppose it is required to convert the decimal number N into binary form, dividing N by 2 in the

decimal system, we will obtain a quotient N1 and a remainder R1, where R1 can have a value of either 0 or 1. The process is repeated until the quotient becomes 0 or 1. When the quotient is ‘0’ or ‘1’, it is the final remainder value. Write the final answer starting from final remainder value obtained to the first remainder value obtained.

Example

Convert $(65)_{10}$ into its equivalent binary number



$$(65)_{10} = (1\ 0\ 0\ 0\ 0\ 0\ 1)_2$$

Method 2 : Sum of Powers of 2.

A decimal number can be converted into a binary number by adding up the powers of 2 and then adding bits as needed to obtain the total value of the number.

- Find the largest power of 2 that is smaller than or equal to 65.

$$65_{10} > 64_{10}$$

- Set the 64's bit to 1 and subtract 64 from the original number

$$65 - 64 = 1$$

- 32 is greater than the remaining total. Therefore, set the 32's bit to 0.

- 16 is greater than the remaining total. Therefore, set the 16's bit to 0.

- 8 is greater than the remaining total. Therefore, set the 8's bit to 0.

- 4 is greater than the remaining total. Therefore, set the 4's bit to 0.

- 2 is greater than the remaining total. Therefore, set the 2's bit to 0.

- As the remaining value is equivalent to 1's bit, set it to 1.

$$1 - 1 = 0$$

Conversion is complete $65_{10} = (1000001)_2$

Example

The conversion steps can be given as follows:

Given Number : 65

Equivalent or value less than power of 2 is : 64

$$(1) \ 65 - 64 = 1$$

$$(2) \ 1 - 1 = 0$$

Power's of 2	64	32	16	8	4	2	1
Binary Number	1	0	0	0	0	0	1

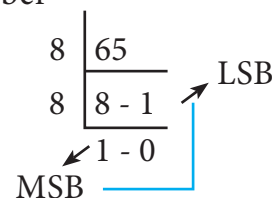
$$65_{10} = (1000001)_2$$

2.4.2 Decimal to Octal Conversion

To convert Decimal to Octal, "Repeated Division by 8" method can be used. The method is the same we have learnt in 2.4.1, but in this method, we have to divide the given number by 8.

Example

Convert $(65)_{10}$ into its equivalent Octal number



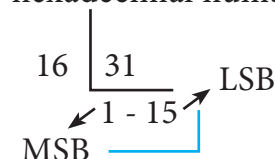
$$(65)_{10} = (1\ 0\ 1)_8$$

2.4.3 Decimal to Hexadecimal Conversion

To convert Decimal to Hexadecimal, "Repeated division by 16" method can be used. The method is the same as we have learnt in 2.4.1, but in this method, we have to divide the given number by 16.

Example

Convert $(31)_{10}$ into its equivalent hexadecimal number.



$$(31)_{10} = (1F)_{16} \text{ (Refer Table 2.2 } F=15 \text{)}$$

2.4.4 Conversion of fractional Decimal to Binary

The method of **repeated multiplication by 2** has to be used to convert such kind of decimal fractions.

The steps involved in the method of **repeated multiplication by 2**:

- Step 1: Multiply the decimal fraction by 2 and note the integer part. The integer part is either 0 or 1.
- Step 2: Discard the integer part of the previous product. Multiply the fractional part of the previous product by 2. Repeat Step 1 until the same fraction repeats or terminates (0).
- Step 3: The resulting integer part forms a sequence of 0s and 1s that become the binary equivalent of decimal fraction.
- Step 4: The final answer is to be written from first integer part obtained till the last integer part obtained.

	Integer part
$0.2 \times 2 = 0.4$	0 (first integer part obtained)
$0.4 \times 2 = 0.8$	0
$0.8 \times 2 = 1.6$	1
$0.6 \times 2 = 1.2$	1
$0.2 \times 2 = 0.4$	0 (last integer part obtained)

Note: Fraction repeats, the product is the same as in the first step.

Write the integer parts from top to bottom to obtain the equivalent fractional binary number. Hence $(0.2)_{10} = (0.00110011...)_{2} = (0.00110011)_{2}$

Workshop

3. Convert the following Decimal numbers to its equivalent Binary, Octal, Hexadecimal.

- 1) 1920 2) 255 3) 126

2.4.5 Binary to Decimal Conversion

To convert Binary to Decimal we can use positional notation method.

- Step 1: Write down the Binary digits and list the powers of 2 from right to left (Positional Notation)
- Step 2: For each positional notation written for the digit, now write the equivalent weight.
- Step 3: Multiply each digit with its corresponding weight
- Step 4: Add all the values.

Table 2.3 Positional Notation and Weight

Positional Notation	Weight	Positional Notation	Weight
2^0	1	2^6	64
2^1	2	2^7	128
2^2	4	2^8	256
2^3	8	2^9	512
2^4	16	2^{10}	1024
2^5	32		

Example

Convert $(111011)_2$ into its equivalent decimal number.

Weight	32	16	8	4	2	1
Positional Notation	2^5	2^4	2^3	2^2	2^1	2^0
Given number	1	1	1	0	1	1

$$\begin{aligned}
 32+16+8+0+2+1 &= (59)_{10} \\
 (111011)_2 &= (59)_{10}
 \end{aligned}$$

2.4.6 Binary to Octal Conversion

Step 1: Group the given binary number into 3 bits from right to left.

Step 2: You can add preceding 0 to make a group of 3 bits if the left most group has less than 3 bits.

Step 3: Convert equivalent octal value using "2's power positional weight method"

Table 2.4 Octal numbers and their Binary equivalent

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

Example

Convert $(11010110)_2$ into octal equivalent number

Step 1: Group the given number into 3 bits from right to left.

011 010 110

Note: The left most groups have less than 3 bits, so 0 is added to its left to make a group of 3 bits.

Step-2: Find Octal equivalent of each group

$$\begin{array}{ccc}
 \underbrace{011} & \underbrace{010} & \underbrace{110} \\
 3 & 2 & 6 \\
 (11010110)_2 = (326)_8
 \end{array}$$

2.4.7. Binary to Hexadecimal Conversion

Step 1: Group the given number into 4 bits from right to left.

Step 2: You can add preceding 0's to make a group of 4 bits if the left most group has less than 4 bits.

Step 3: Convert equivalent Hexadecimal value using "2's power positional weight method"

Example

Convert $(1111010110)_2$ into Hexadecimal number

Step 1: Group the given number into 4 bits from right to left.

0011 1101 0110

Note: 0's are added to the left most group to make it a group of 4 bits

$$\begin{array}{ccc}
 \underbrace{0011} & \underbrace{1101} & \underbrace{0110} \\
 3 & D & 6 \\
 (1111010110)_2 = (3D6)_{16}
 \end{array}$$

2.4.8 Conversion of fractional Binary to Decimal equivalent

Follow the steps to convert fractional Binary number to its Decimal equivalent.

Step 1: Convert integral part of Binary to Decimal equivalent using positional notation method (Procedure is same as discussed in 2.4.5)

Step 2: To convert the fractional part of binary to its decimal equivalent.

Step 2.1: Write down the Binary digits in the fractional part

Step 2.2: For all the digits write powers of 2 from left to right starting from 2^{-1} , 2^{-2} , 2^{-3} ,..... 2^{-n} , now write the equivalent weight.

Step 2.3: Multiply each digit with its corresponding weight

Step 2.4: Add all the values which you obtained in Step 2.3

Table 2.5 Positional notation and weight

Positional notation	Weight
2^{-1} (1/2)	0.5
2^{-2} (1/4)	0.25
2^{-3} (1/8)	0.125
2^{-4} (1/16)	0.0625
2^{-5} (1/32)	0.03125
2^{-6} (1/64)	0.015625
2^{-7} (1/128)	0.0078125

Step 3: To get final answer write the integral part (after conversion), followed by a decimal point(.) and the answer arrived at Step 2.4

Example

Convert the given Binary number $(11.011)_2$ into its decimal equivalent Integer part $(11)_2 = 3$ (Refer table 2.2)

2^1	2^0		2^{-1}	2^{-2}	2^{-3}
↑	↑		↑	↑	↑
1	1	.	0	1	1

$$3 + . (0 \times 0.5 + 1 \times 0.25 + 1 \times 0.125)$$

$$= 3.375$$

$$(11.011)_2 = (3.375)_{10}$$

Workshop

4. Convert the given Binary number into its equivalent Decimal, Octal and Hexadecimal number.

1) 101110101 2) 1011010 3) 101011111

2.4.9. Octal to Decimal Conversion

To convert Octal to Decimal, we can use positional notation method.

1. Write down the Octal digits and list the powers of 8 from right to left (Positional Notation)

2. For each positional notation of the digit write the equivalent weight.

3. Multiply each digit with its corresponding weight

4. Add all the values

Example

Convert $(1265)_8$ to equivalent Decimal number

Weight	512	64	8	1
Positional Notation	8^3	8^2	8^1	8^0
Given number	1	2	6	5

$$(1265)_8 = 512 \times 1 + 64 \times 2 + 8 \times 6 + 1 \times 5$$

$$= 512 + 128 + 48 + 5$$

$$(1265)_8 = (693)_{10}$$

2.4.10 Octal to Binary Conversion

For each Octal digit in the given number write its 3 digits binary equivalent using positional notation.

Example

Convert $(6213)_8$ to equivalent Binary number

6	2	1	3
↓	↓	↓	↓
110	010	001	011

$$(6213)_8 = (110010001011)_2$$

Workshop

5. Convert the following Octal numbers into Binary numbers.

(A) 472 (B) 145 (C) 347
(D) 6247 (E) 645

2.4.11 Hexadecimal to Decimal Conversion

To convert Hexadecimal to Decimal we can use positional notation method.

1. Write down the Hexadecimal digits and list the powers of 16 from right to left (Positional Notation)
2. For each positional notation written for the digit, now write the equivalent weight.
3. Multiply each digit with its corresponding weight
4. Add all the values to get one final value.

Example

Convert $(25F)_{16}$ into its equivalent Decimal number.

Weight	256	16	1
Positional Notation	16^2	16^1	16^0
Given number	2	5	F(15)

$$(25F)_{16} = 2 \times 256 + 5 \times 16 + 15 \times 1$$

$$= 512 + 80 + 15$$

$$(25F)_{16} = (607)_{10}$$

2.4.12 Hexadecimal to Binary Conversion

Write 4 bits Binary equivalent for each Hexadecimal digit for the given number using positional notation method.

Example

Convert $(8BC)_{16}$ into equivalent Binary number

8	B	C
↓	↓	↓
1000	1011	1100
$(8BC)_{16} = (100010111100)_2$		

Workshop

6. Convert the following Hexadecimal numbers to Binary numbers

- (A) A6 (B) BE
(C) 9BC8 (D) BC9

2.5 Binary Representation for Signed Numbers

Computers can handle both positive (unsigned) and negative (signed) numbers. The simplest method to represent negative binary numbers is called **Signed Magnitude**. In signed magnitude method, the left most bit is Most Significant Bit (MSB), is called **sign bit or parity bit**.

The numbers are represented in computers in different ways:

- Signed Magnitude representation
- 1's Complement
- 2's Complement

2.5.1 Signed Magnitude representation

The value of the whole numbers can be determined by the sign used before it. If the number has '+' sign or no sign it will be considered as positive. If the number has '-' sign it will be considered as negative.

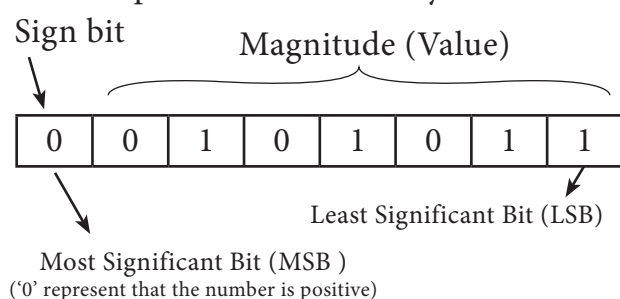
Example:

+43 or 43 is a positive number

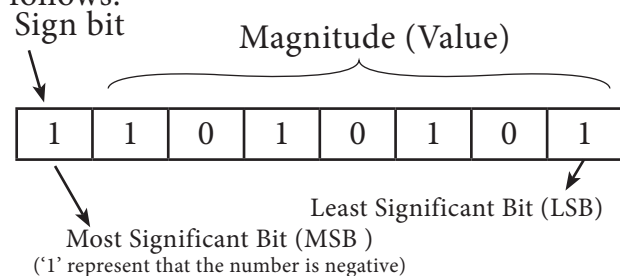
-43 is a negative number

In signed binary representation, the left most bit is considered as sign bit. If this bit is 0, it is a positive number and if it 1, it is a negative number. Therefore a signed binary number has 8 bits, only 7 bits used for storing values (magnitude) and the 1 bit is used for sign.

+43 is represented in memory as follows:



-43 can be represented in memory as follows.



2.5.2 1’s Complement representation

This is an easier approach to represent signed numbers. This is for negative numbers only i.e. the number whose MSB is 1.

The steps to be followed to find 1’s complement of a number:

- Step 1: Convert given Decimal number into Binary
- Step 2: Check if the binary number contains 8 bits, if less add 0 at the left most bit, to make it as 8 bits.
- Step 3: Invert all bits (i.e. Change 1 as 0 and 0 as 1)

Example

Find 1’s complement for $(-24)_{10}$

Given Number	Binary Number	1’s Compliment
$(-24)_{10}$	00011000	11100111

2.5.3 2’s Complement representation

The 2’s-complement method for negative number is as follows:

- Invert all the bits in the binary sequence (i.e., change every 0 to 1 and every 1 to 0 i.e., 1’s complement)
- Add 1 to the result to the Least Significant Bit (LSB).

Example

2’s Complement represent of $(-24)_{10}$

Binary equivalent of +24:	11000
8bit format:	00011000
1’s complement:	11100111
Add 1 to LSB:	+1
2’s complement of -24:	11101000

Workshop

7. Write the 1’s complement number and 2’s complement number for the following decimal numbers:
(A) 22 (B) -13 (C) -65 (D) -46

2.6 Binary Arithmetic

As decimal numbers, the binary numbers also permit computations like addition, subtraction, multiplication and division. The following session deals only with binary addition and subtraction.

2.6.1 Binary Addition

The following table is useful when adding two binary numbers.

A	B	SUM (A + B)	Carry
0	0	0	-
0	1	1	-
1	0	1	-
1	1	0	1

In $1 + 1 = 10$, is considered as sum 0 and the 1 as carry bit. This carry bit is added with the previous position of the bit pattern.

Example Add: $1011_2 + 1001_2$

(Carry Bit)→

	1	0	1	1	
+	1	0	0	1	
	1	0	1	0	0

$1011_2 + 1001_2 = 10100_2$

Example Perform Binary addition for the following: $23_{10} + 12_{10}$

Step 1: Convert 23 and 12 into binary form

23_{10}					
2's power	16	8	4	2	1
Binary Number	1	0	1	1	1
$23_{10} = 00010111_2$					

12_{10}				
2's power	8	4	2	1
Binary Number	1	1	0	0
$12_{10} = 00001100_2$				

Step 2: Binary addition of 23 and 12:

Carry Bit →	1	1	1			
$23_{10} = 0$	0	0	1	0	1	1
$12_{10} = 0$	0	0	0	1	1	0
$35_{10} = 0$	0	1	0	0	0	1

2.6.2 Binary Substraction

The table for Binary Substraction is as follows:

A	B	Difference (A-B)	Borrow
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

When subtracting 1 from 0, borrow 1 from the next Most Significant Bit, when borrowing from the next Most Significant Bit, if it is 1, replace it with 0. If the next Most

Significant Bit is 0, you must borrow from a more significant bit that contains 1 and replace it with 0 and 0s upto that point become 1s.

Example Subtract $1001010_2 - 10100_2$

	0	1	10	0	10	
	1	0	0	1	0	1
(-)			1	0	1	0
			1	1	0	1

Example Perform binary addition for the following: $(-21)_{10} + (5)_{10}$

Step 1: Change -21 and 5 into binary form

21_{10}					
2's power	16	8	4	2	1
Binary Number	1	0	1	0	1
$21_{10} = 00010101_2$					

5_{10}		
2's power	4	2
Binary Number	1	0
$5_{10} = 00000101_2$		

Step 2:

21_{10}	0	0	0	1	0	1	0	1
1's Complement	1	1	1	0	1	0	1	0
2's Complement	1	1	1	0	1	0	1	1

Step 3:

Binary Addition of -21 and 5 :

Carry bit				1	1	1	1	
-21_{10}	1	1	1	0	1	0	1	1
5_{10}	0	0	0	0	0	1	0	1
-16_{10} (Result)	1	1	1	1	0	0	0	0

Workshop

8. Perform the following binary computations:

- (A) $10_{10} + 15_{10}$ (B) $-12_{10} + 5_{10}$
 (C) $14_{10} - 12_{10}$ (D) $(-2_{10}) - (-6_{10})$

2.7 Representing Characters in Memory

As represented in introduction, all the input data given to the computer should be in understandable format. In general, 26 uppercase letters, 26 lowercase letters, 0 to 9 digits and special characters are used in a computer, which is called character set. All these character set are denoted through numbers only. All Characters in the character set needs a common encoding system. There are several encoding systems used for computer. They are

- BCD – Binary Coded Decimal
- EBCDIC – Extended Binary Coded Decimal Interchange Code
- ASCII – American Standard Code for Information Interchange
- Unicode
- ISCII - Indian Standard Code for Information Interchange

2.7.1 Binary Coded Decimal (BCD)

This encoding system is not in the practice right now. This is 2^6 bit encoding system. This can handle $2^6 = 64$ characters only.

2.7.2 American Standard Code for Information Interchange (ASCII)

This is the most popular encoding system recognized by United States. Most of the computers use this system. Remember this encoding system can handle English characters only. This can handle 2^7 bit which means 128 characters.

In this system, each character has individual number (Refer **Appendix**).

The new edition (version) ASCII -8, has 2^8 bits and can handle 256 characters are represented from 0 to 255 unique numbers.

The ASCII code equivalent to the uppercase letter 'A' is 65. The binary representation of ASCII (7 bit) value is 1000001. Also 01000001 in ASCII-8 bit.

2.7.3 Extended Binary Coded Decimal Interchange Code (EBCDIC)

This is similar to ASCII Code with 8 bit representation. This coding system is formulated by International Business Machine(IBM). The coding system can handle 256 characters. The input code in ASCII can be converted to EBCDIC system and vice - versa.

2.7.4 Indian Standard Code for Information Interchange (ISCII)

ISCII is the system of handling the character of Indian local languages. This as a 8-bit coding system. Therefore it can handle 2^8 characters. This system is formulated by the department of Electronics in India in the year 1986-88 and recognized by Bureau of Indian Standards (BIS). Now this coding system is integrated with Unicode.

2.7.5 Unicode

This coding system is used in most of the modern computers. The popular coding scheme after ASCII is Unicode. ASCII can represent only 256 characters. Therefore English and European Languages alone can be handled by ASCII. Particularly there was a situation, when the languages like Tamil, Malayalam, Kannada and Telugu could not be represented by ASCII. Hence, the Unicode was generated to handle all the coding system of Universal languages. This is 16 bit code and can handle 65536 characters.

Unicode scheme is denoted by hexadecimal numbers. The Unicode table of Tamil, Malayalam, Telugu and Kannada is shown in Table 2.6

Table 2.6

Unicode Table of Tamil								Unicode Table of Malayalam									
	0B8	0B9	0BA	0BB	0BC	0BD	0BE	0BF		0D0	0D1	0D2	0D3	0D4	0D5	0D6	0D7
0		ஐ 0B90		ர 0BB0	ீ 0BC0	ஓ 0BD0		ய 0BF0	0	ஃ 0D00	ஹ 0D10	௦ 0D20	௪ 0D30	ி 0D40		ஐ 0D60	ய 0D70
1				ற 0BB1	ு 0BC1			ள 0BF1	1	ஃ 0D01		ய 0D21	௪ 0D31	ு 0D41		ஐ 0D61	ற 0D71
2	ஃ 0B82	ஓ 0B92		ல 0BB2	ு 0BC2			த 0BF2	2	ஃ 0D02	௪ 0D12	ய 0D22	ல 0D32	ு 0D42		ஐ 0D62	ற 0D72
3	ஃ 0B83	ஓ 0B93	ண 0BA3	ள 0BB3				உ 0BF3	3	ஃ 0D03	ஓ 0D13	ள 0D23	ஐ 0D33	ு 0D43		ஐ 0D63	ற 0D73
4		ஓள 0B94	த 0BA4	ழ 0BB4				ம் 0BF4	4		ஓ 0D14	ற 0D24	ய 0D34	ு 0D44	உ 0D54		ஐ 0D74
5	அ 0B85	க 0B95		வ 0BB5				ஹ 0BF5	5	ஓ 0D05	க 0D15	ம 0D25	வ 0D35		ய 0D55		ஐ 0D75
6	ஆ 0B86			ய 0BB6	ெ 0BC6		௦ 0BE6	பு 0BF6	6	ஓ 0D06	வ 0D16	ப 0D26	ஸ 0D36	ெ 0D46	ஓ 0D56	௦ 0D66	ஹ 0D76
7	இ 0B87			ஷ 0BB7	ே 0BC7	ள 0BD7	க 0BE7	ஹ 0BF7	7	ஓ 0D07	ய 0D17	ய 0D27	ஷ 0D37	ே 0D47	ு 0D57	௦ 0D67	ஹ 0D77
8	ஈ 0B88		ந 0BA8	ஸ 0BB8	ை 0BC8		உ 0BE8	ஹ 0BF8	8	ஓ 0D08	ஹ 0D18	ம 0D28	ஸ 0D38	ெ 0D48	ய 0D58	௦ 0D68	ஹ 0D78
9	உ 0B89	ங 0B99	ன 0BA9	ஹ 0BB9			ந 0BE9	ஹ 0BF9	9	உ 0D09	௪ 0D19	ள 0D29	ஹ 0D39		ஓ 0D59	௦ 0D69	ஹ 0D79
A	ஊ 0B8A	ச 0B9A	ப 0BAA		ொ 0BCA		ச 0BEA	நீ 0BFA	A	ஓ 0D0A	ஹ 0D1A	ஹ 0D2A	ஹ 0D3A	ொ 0D4A	ய 0D5A	௦ 0D6A	ஹ 0D7A
B					ோ 0BCB		ஹ 0BEB		B	ஓ 0D0B	ஹ 0D1B	ஹ 0D2B	ஹ 0D3B	ோ 0D4B	ய 0D5B	௦ 0D6B	ஹ 0D7B
C		ஐ 0B9C			ெள 0BCC		சு 0BEC		C	ஓ 0D0C	ஹ 0D1C	ஹ 0D2C	ஹ 0D3C	ெ 0D4C	ய 0D5C	௦ 0D6C	ஹ 0D7C
D					ஃ 0BCD		எ 0BED		D		ய 0D1D	ப 0D2D	ஹ 0D3D	ஃ 0D4D	ய 0D5D	௦ 0D6D	ஹ 0D7D
E	எ 0B8E	ஞ 0B9E	ம 0BAE	ா 0BBE			அ 0BEE		E	ஓ 0D0E	ஹ 0D1E	ம 0D2E	ஹ 0D3E	ஃ 0D4E	ய 0D5E	௦ 0D6E	ஹ 0D7E
F	ஏ 0B8F	ஹ 0B9F	ய 0BAF	ி 0BBF			க 0BEF		F	ஓ 0D0F	ஹ 0D1F	ய 0D2F	ஹ 0D3F	ஹ 0D4F	ஹ 0D5F	௦ 0D6F	ஹ 0D7F

Table 2.6

Unicode Table of Telugu								Unicode Table of Kannada									
	0C0	0C1	0C2	0C3	0C4	0C5	0C6	0C7		0C8	0C9	0CA	0CB	0CC	0CD	0CE	0CF
0	ం	ఐ	ఠ	ర	ీ		ఋ		0	ం	ఐ	ఠ	ర	ీ		ఋ	
1	ఁ		డ	ఱ	ు		౞		1	ం		డ	ఱ	ు		౞	౞
2	ం	ఓ	ఢ	ల	ా		౞		2	ం	ఓ	ఢ	ల	ా		౞	౞
3	ం	ఓ	ణ	ళ	ృ		౞		3	ం	ఓ	ణ	ళ	ృ		౞	
4		ఙ	త	ఱ	ౄ				4		ఙ	త		ౄ			
5	అ	క	థ	వ		ం			5	అ	క	థ	వ		ం		
6	ఆ	ఖ	ద	ళ	ౌ	ం	ం		6	ఆ	ఖ	ద	ళ	ౌ	ం	ం	
7	ఇ	గ	ధ	ష	ౌ		ం		7	ఇ	గ	ధ	ష	ౌ		ం	
8	ఈ	ఘ	న	స	ౌ	ఙ	ం	ం	8	ఈ	ఘ	న	స	ౌ		ం	
9	ఉ	జ		హ		జ	ం	ం	9	ఉ	జ		హ			ం	
A	ఊ	చ	ప		ౌ	ఊ	ం	ం	A	ఊ	చ	ప		ౌ		ం	
B	ఋ	భ	ఫ		ౌ		ం	ం	B	ఋ	భ	ఫ		ౌ		ం	
C	ౠ	జ	బ		ౌ		ం	ం	C	ౠ	జ	బ		ౌ		ం	
D		ఝ	భ	ౌ	ౌ		ం	ం	D		ఝ	భ	ౌ	ౌ		ం	
E	ఎ	ఞ	మ	ౌ			ం	ం	E	ఎ	ఞ	మ	ౌ		ం	ం	
F	ప	ట	య	ీ			ం	ం	F	ప	ట	య	ీ			ం	



Appendix
American Standard Code for Information Interchange (ASCII)
(Few specific characters only)

Alphabets

Alphabets	Decimal number	Binary number (8 bit)	Octal number	Hexadecimal number
A	65	01000001	101	41
B	66	01000010	102	42
C	67	01000011	103	43
D	68	01000100	104	44
E	69	01000101	105	45
F	70	01000110	106	46
G	71	01000111	107	47
H	72	01001000	110	48
I	73	01001001	111	49
J	74	01001010	112	4A
K	75	01001011	113	4B
L	76	01001100	114	4C
M	77	01001101	115	4D
N	78	01001110	116	4E
O	79	01001111	117	4F
P	80	01010000	120	50
Q	81	01010001	121	51
R	82	01010010	122	52
S	83	01010011	123	53
T	84	01010100	124	54
U	85	01010101	125	55
V	86	01010110	126	56
W	87	01010111	127	57
X	88	01011000	130	58
Y	89	01011001	131	59
Z	90	01011010	132	5A
a	97	01100001	141	61
b	98	01100010	142	62
c	99	01100011	143	63
d	100	01100100	144	64
e	101	01100101	145	65





f	102	01100110	146	66
g	103	01100111	147	67
h	104	01101000	150	68
i	105	01101001	151	69
j	106	01101010	152	6A
k	107	01101011	153	6B
l	108	01101100	154	6C
m	109	01101101	155	6D
n	110	01101110	156	6E
o	111	01101111	157	6F
p	112	01110000	160	70
q	113	01110001	161	71
r	114	01110010	162	72
s	115	01110011	163	73
t	116	01110100	164	74
u	117	01110101	165	75
v	118	01110110	166	76
w	119	01110111	167	77
x	120	01111000	170	78
y	121	01111001	171	79
z	122	01111010	172	7A

Numerals

Alphabets	Decimal number	Binary number (8 bit)	Octal number	Hexadecimal number
0	48	00110000	60	30
1	49	00110001	61	31
2	50	00110010	62	32
3	51	00110011	63	33
4	52	00110100	64	34
5	53	00110101	65	35
6	54	00110110	66	36
7	55	00110111	67	37
8	56	00111000	70	38
9	57	00111001	71	39



Special Characters

Special symbols	Decimal number	Binary number (8 bit)	Octal number	Hexadecimal number
Blank	32	00100000	40	20
!	33	00100001	41	21
"	34	00100010	42	22
#	35	00100011	43	23
\$	36	00100100	44	24
%	37	00100101	45	25
&	38	00100110	46	26
'	39	00100111	47	27
(40	00101000	50	28
)	41	00101001	51	29
*	42	00101010	52	2A
+	43	00101011	53	2B
,	44	00101100	54	2C
-	45	00101101	55	2D
.	46	00101110	56	2E
/	47	00101111	57	2F
:	58	00111010	72	3A
;	59	00111011	73	3B
<	60	00111100	74	3C
=	61	00111101	75	3D
>	62	00111110	76	3E
?	63	00111111	77	3F
@	64	01000000	100	40
[91	01011011	133	5B
\	92	01011100	134	5C
]	93	01011101	135	5D
^	94	01011110	136	5E
_	95	01011111	137	5F
`	96	01100000	140	60
{	123	01111011	173	7B
	124	01111100	174	7C
}	125	01111101	175	7D
~	126	01111110	176	7E

Evaluation



SECTION - A

Choose the correct answer:

- Which refers to the number of bits processed by a computer's CPU?
A) Byte B) Nibble C) Word length D) Bit
- How many bytes does 1 KiloByte contain?
A) 1000 B) 8 C) 4 D) 1024
- Expansion for ASCII
A) American School Code for Information Interchange
B) American Standard Code for Information Interchange
C) All Standard Code for Information Interchange
D) American Society Code for Information Interchange
- 2^{50} is referred as
A) Kilo B) Tera C) Peta D) Zetta
- How many characters can be handled in Binary Coded Decimal System?
A) 64 B) 255 C) 256 D) 128
- For 1101_2 the equivalent Hexadecimal equivalent is?
A) F B) E C) D D) B
- What is the 1's complement of 00100110?
A) 00100110 B) 11011001 C) 11010001 D) 00101001
- Which amongst this is not an Octal number?
A) 645 B) 234 C) 876 D) 123

SECTION-B

Very Short Answers

- What is data?
- Write the 1's complement procedure.
- Convert $(46)_{10}$ into Binary number
- We cannot find 1's complement for $(28)_{10}$. State reason.
- List the encoding systems that represents characters in memory.

SECTION-C

Short Answers

- What is radix of a number system? Give example
- Write note on binary number system.
- Convert $(150)_{10}$ into Binary, then convert that Binary number to Octal
- Write short note on ISCII
- Add a) $-22_{10} + 15_{10}$ b) $20_{10} + 25_{10}$

SECTION - D

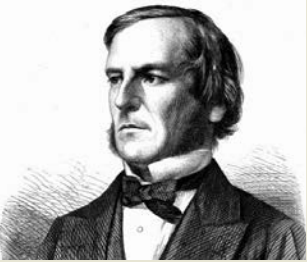
Explain in detail

- a) Write the procedure to convert fractional Decimal to Binary
b) Convert $(98.46)_{10}$ to Binary
- Find 1's Complement and 2's Complement for the following Decimal number
a) -98 b) -135
- a) Add $1101010_2 + 101101_2$ b) Subtract $1101011_2 - 111010_2$

Part - II - Boolean Algebra

2.8 Introduction

Boolean algebra is a mathematical discipline that is used for designing digital circuits in a digital computer. It describes the relation between inputs and outputs of a digital circuit. The name Boolean algebra has been given in honor of an English mathematician George Boole who proposed the basic principles of this algebra.



DO YOU KNOW? George Boole (1815-1864) was born to a lower class family and only received an elementary school education. Despite that, he taught himself highly advanced mathematics and different languages as a teenager without any assistance. He started teaching at age sixteen, and started his own school at age nineteen. By his mid-twenties, he had mastered most of the important mathematical principles in his day.

2.8.1 Binary valued quantities:

Every day we have to make logical decisions:

1. Should I carry Computer Science book every day? Yes / No
2. $8-10 = 10$ is this answer correct? Yes / No
3. Chennai is capital of India? Yes / No
4. What did I say yesterday?

The first three questions thrown above, the answer may be True (Yes) or False (No). But the fourth one, we cannot be answer as True or False. Thus, sentences which can be determined to be True or False are called “Logical Statement” or “Truth Functions”. The results True or False are called “Truth Values”. The truth values depicted by logical constant 1 and 0; 1 means True and 0 means False. The variable which can store these truth values are called “Logical variable” or “Binary valued variables” or “Boolean Variables” as these can store one of the two values of True or False.

2.8.2 Logical Operations:

Boolean algebra makes use of variables and operations (functions). The basic logical operations are AND, OR and NOT, which are symbolically represented by dot (.), plus (+), and by over bar / single apostrophe respectively. These symbols are also called as “Logical Operators”.

2.8.3 Truth Table:

A truth table represents all the possible values of logical variable or statements along with all the possible results of given combination of truth values.

2.8.4 AND operator

The AND operator is defined in Boolean algebra by the use of the dot (.) operator. It is similar to multiplication in ordinary algebra. The AND operator combines two or more input variables so that the output is true only if all the inputs are true. The truth table for a 2-input AND operator is shown as follows:

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The above 2-input AND operation is expressed as: $Y = A \cdot B$

2.8.5 OR operator

The plus sign is used to indicate the OR operator. The OR operator combines two or more input variables so that the output is true if at least one input is true. The truth table for a 2-input OR operator is shown as follows:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

The above 2-input OR operation is expressed as: $Y = A + B$

2.8.6 NOT operator

The NOT operator has one input and one output. The input is either true or false, and the output is always the opposite, that is, the NOT operator inverts the input. The truth table for a NOT operator where A is the input variable and Y is the output is shown below:

A	Y
0	1
1	0

The NOT operator is represented algebraically by the Boolean expression:
 $Y = \overline{A}$

Example:

Consider the Boolean equation:

$$D = A + (B \cdot C)$$

D is equal to 1 (true) if A is 1 or if $(B \cdot \underline{C})$ is 1, that is, B = 0 and C = 1.

Otherwise D is equal to 0 (false).

The basic logic functions AND, OR, and NOT can also be combined to make other logic operators such as NAND and NOR

2.8.7 NAND operator

The NAND is the combination of NOT and AND. The NAND is generated by inverting the output of an AND operator. The algebraic expression of the NAND function is:

$$Y = \overline{A \cdot B}$$

The NAND function truth table is shown below:

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

$$A \text{ NAND } B = \text{NOT } (A \text{ AND } B)$$

2.8.8 NOR operator

The NOR is the combination of NOT and OR. The NOR is generated by inverting the output of an OR operator. The algebraic expression of the NOR function is:

$$Y = \overline{A + B}$$

The NOR function truth table is shown below:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

$$A \text{ NOR } B = \text{NOT } (A \text{ OR } B)$$

2.9 Basic Logic Gates:

A gate is a basic electronic circuit which operates on one or more signals to produce an output signal. There are three fundamental gates namely AND, OR and NOT. The other logic gates like NAND, NOR, XOR and XNOR are derived gates which are derived from the fundamental gates. NAND and NOR gates are called Universal gates, because the fundamental logic gates can be realized through them.

2.9.1 AND Gate

The AND gate can have two or more input signals and produce an output signal.

The output is "true" only when both inputs are "true", otherwise, the output is "false". In other words the output will be 1 if and only if both inputs are 1; otherwise the output is 0. The output of the AND gate is represented by a variable say C, where A and B are two boolean variables. In boolean algebra, a variable can take either of the values '0' or '1'. The logical symbol of the AND gate is

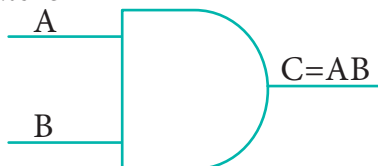


Fig. 2.4 Logic symbol of AND Gate

One way to symbolize the action of an AND gate is by writing the boolean function.

$$C = A \text{ AND } B$$

In boolean algebra the multiplication sign stands for the AND operation. Therefore, the output of the AND gate is

$$C = A \cdot B \text{ or}$$

$$\text{simply } C = AB$$

Read this as "C equals A AND B". Since there are two input variables here, the truth table has four entries, because there are four possible inputs : 00, 01, 10 and 11.

For instance if both inputs are 0,

$$\begin{aligned} C &= A \cdot B \\ &= 0 \cdot 0 \\ &= 0 \end{aligned}$$

The truth table for AND Gate is

Input		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.7 Truth Table for AND Gate

2.9.2 OR Gate

The OR gate gets its name from its behaviour like the logical inclusive "OR". The output is "true" if either or both of the inputs are "true". If both inputs are "false" then the output is "false". In other words the output will be 1 if and only if one or both inputs are 1; otherwise, the output is 0. The logical symbol of the OR gate is



Fig. 2.5 Logic symbol of OR Gate

The OR gate output is

$$C = A \text{ OR } B$$

We use the + sign to denote the OR function. Therefore,

$$C = A + B$$

Read this as "C equals A OR B".

For instance, if both the inputs are 1

$$C = A + B = 1 + 1 = 1$$

The truth table for OR gate is

Input		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Table 2.8 Truth Table for OR Gate

2.9.3 NOT Gate

The NOT gate, called a logical inverter, has only one input. It reverses the logical state. In other words the output C is always the complement of the input. The logical symbol of the NOT gate is

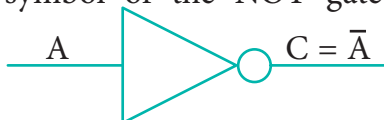


Fig. 2.6 Logic symbol of NOT Gate

The boolean function of NOT gate is

$$C = \text{NOT } A$$

In boolean algebra, the overbar stands for NOT operation. Therefore,

$$C = \bar{A}$$

Read this as "C equals NOT A" or "C equals the complement of A".

If A is 0,

$$C = \bar{0} = 1$$

On the otherhand, if A is 1,

$$C = \bar{1} = 0$$

The truth table for NOT gate is

Input	Output
A	C
1	0
0	1

Table 2.9 Truth Table for NOT Gate

2.9.4 NOR Gate

The NOR gate circuit is an OR gate followed by an inverter. Its output is "true" if both inputs are "false" Otherwise, the output is "false". In other words, the only way to get '1' as output is to have both inputs '0'. Otherwise the output is 0. The logic circuit of the NOR gate is

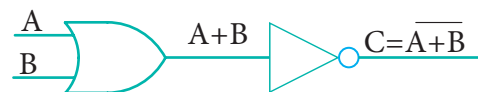


Fig. 2.7 Logic Circuit of NOR Gate



Fig. 2.8 Logic symbol of NOR Gate

The output of NOR gate is

$$C = (\bar{A} + \bar{B})$$

Read this as "C equals NOT of A OR B" or "C equals the complement of A OR B". For example if both the inputs are 0,

$$C = (\bar{0} + \bar{0}) = \bar{0} = 1$$

The truth table for NOR gate is

Input		Output
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Table 2.10 Truth Table for NOR Gate

2.9.5 Bubbled AND Gate

The Logic Circuit of Bubbled AND Gate

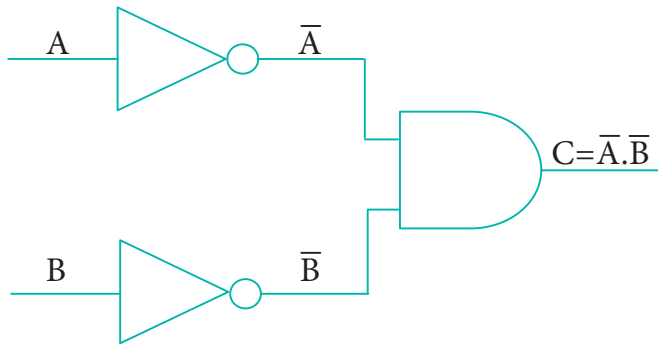


Fig. 2.9 Logic circuit of Bubbled AND Gate

In the above circuit, invertors on the input lines of the AND gate gives the output as

$$C = (\bar{A} \cdot \bar{B})$$

This circuit can be redrawn as the bubbles on the inputs, where the bubbles represent inversion.

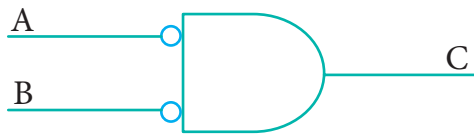


Fig. 2.10 Logic Symbol of Bubbled AND Gate

We refer this as bubbled AND gate. Let us analyse this logic circuit for all input possibilities.

$$\text{If } A = 0 \text{ and } B = 0 \quad C = (\bar{0} \cdot \bar{0}) = 1 \cdot 1 = 1$$

$$\text{If } A = 0 \text{ and } B = 1 \quad C = (\bar{0} \cdot \bar{1}) = 1 \cdot 0 = 0$$

$$\text{If } A = 1 \text{ and } B = 0 \quad C = (\bar{1} \cdot \bar{0}) = 0 \cdot 1 = 0$$

$$\text{If } A = 1 \text{ and } B = 1 \quad C = (\bar{1} \cdot \bar{1}) = 0 \cdot 0 = 0$$

Here the truth table is

Input		Output
A	B	C
0	0	1
0	1	0

1	0	0
1	1	0

You can see that, a bubbled AND gate produces the same output as a NOR gate. So, You can replace each NOR gate by a bubbled AND gate. In other words the circuits are interchangeable.

Therefore

$$(\overline{A + B}) = \bar{A} \cdot \bar{B}$$

Which establishes the De Morgan's first theorem.

2.9.6 NAND Gate

The NAND gate operates an AND gate followed by a NOT gate. It acts in the manner of the logical operation "AND" followed by inversion. The output is "false" if both inputs are "true", otherwise, the output is "true". In otherwords the output of the NAND gate is 0 if and only if both the inputs are 1, otherwise the output is 1. The logic circuit of NAND gate is

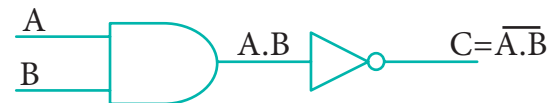


Fig. 2.11 Logic Circuit of NAND Gate

The logical symbol of NAND gate is



Fig. 2.12 Logic Symbol of NAND Gate

The output of the NAND gate is

$$C = (\bar{A} \cdot \bar{B})$$

Read this as "C" equals NOT of A AND B" or "C" equals the complement of A AND B".

For example if both the inputs are 1

$$C = (\overline{1 \cdot 1}) = \overline{1} = 0$$

The truth table for NAND gate is

Input		Output
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Table 2.11 Truth Table for NAND Gate

2.9.7 Bubbled OR Gate

The logic circuit of bubbled OR gate is

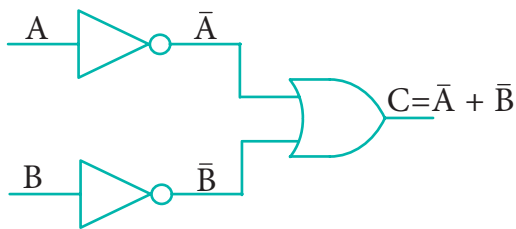


Fig. 2.13 Logic Circuit of Bubbled OR Gate

The output of this circuit can be written as $C = \bar{A} + \bar{B}$

The above circuit can be redrawn as the bubbles on the input, where the bubbles represents the inversion.

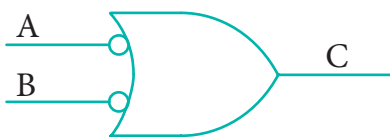


Fig. 2.14 Logic Symbol of Bubbled OR Gate

We refer this as bubbled OR gate. The truth table for the bubbled OR is

Input		Output
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Table 2.12 Truth Table for Bubbled OR Gate

If we compare the truth tables of the bubbled OR gate with NAND gate, they are identical. So the circuits are interchangeable.

Therefore,

$$(\overline{A \cdot B}) = \bar{A} + \bar{B}$$

Which establishes the De Morgan's second theorem.

2.9.8 XOR Gate

The XOR (exclusive - OR) gate acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true". The output is "false" if both inputs are "false" or if both inputs are "true." Another way of looking at this circuit is to observe that the output is 1 if the inputs are different, but 0 if the inputs are the same.

The logic circuit of XOR gate is

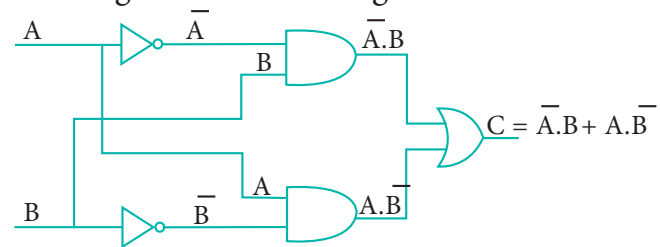


Fig. 2.15 Logic Circuit of XOR Gate

The output of the XOR gate is

The truth table for XOR gate is

Input		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.13 Truth Table for XOR Gate

In boolean algebra, exclusive - OR operator \oplus or "encircled plus".

Hence $C = A \oplus B$

The logical symbol of XOR gate is



Fig. 2.16 Logic Symbol of XOR Gate

2.9.9 XNOR Gate

The XNOR (exclusive - NOR) gate is a combination XOR gate followed by an inverter. Its output is "true" if the inputs are the same, and "false" if the inputs are different. In simple words, the output is 1 if the input are the same, otherwise the output is 0. The logic circuit of XNOR gate is

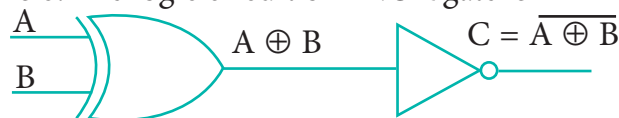


Fig. 2.17 Logic Circuit of XNOR Gate

The output of the XNOR is NOT of XOR

$$\begin{aligned} C &= \overline{A \oplus B} \\ &= \overline{A} \cdot B + A \cdot \overline{B} \\ &= AB + \overline{A} \cdot \overline{B} \end{aligned}$$

(Using De Morgan's Theorem)

In boolean algebra, \odot or "included dot" stands for the XNOR.

Therefore, $C = A \odot B$

The logical symbol is



Fig. 2.18 Logic Symbol of XNOR Gate

The truth table for XNOR Gate is

Input		Output
A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

Table 2.14 Truth Table for XNOR Gate

Using combination of logic gates, complex operations can be performed. In theory, there is no limit to the number of gates that can be arranged together in a single device. But in practice, there is a limit to the number of gates that can be packed into a given physical space. Arrays of logic gates are found in digital integrated circuits.

Theorems of Boolean Algebra

Identity

$$\begin{aligned} A + 0 &= A \\ A \cdot 1 &= A \end{aligned}$$

Complement

$$\begin{aligned} A + \overline{A} &= 1 \\ A \cdot \overline{A} &= 0 \end{aligned}$$

Commutative

$$\begin{aligned} A + B &= B + A \\ A \cdot B &= B \cdot A \end{aligned}$$

Associative

$$\begin{aligned} A + (B + C) &= (A + B) + C \\ A \cdot (B \cdot C) &= (A \cdot B) \cdot C \end{aligned}$$

Distributive

$$\begin{aligned} A \cdot (B + C) &= A \cdot B + A \cdot C \\ A + (B \cdot C) &= (A + B) \cdot (A + C) \end{aligned}$$

Null Element

$$\begin{aligned} A + 1 &= 1 \\ A \cdot 0 &= 0 \end{aligned}$$

Involution

$$\overline{(\overline{A})} = A$$

Idempotence

$$\begin{aligned} A + A &= A \\ A \cdot A &= A \end{aligned}$$

Absorption

$$\begin{aligned} A + (A \cdot B) &= A \\ A \cdot (A + B) &= A \end{aligned}$$

3rd Distributive


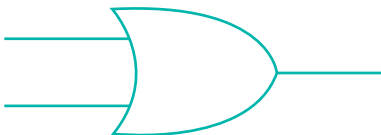
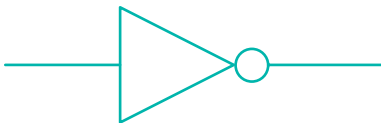


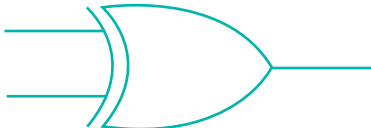

$$A + \overline{A} \cdot B = A + B$$

De Morgan's

$$\begin{aligned} \overline{A + B} &= \overline{A} \cdot \overline{B} \\ \overline{A \cdot B} &= \overline{A} + \overline{B} \end{aligned}$$



Table 2. 15
Logic Gates and their corresponding Truth Tables

Logical Gates	Symbol	Truth Table		
AND		A	B	AB
		0	0	0
		0	1	0
		1	0	0
		1	1	1
OR		A	B	A + B
		0	0	0
		0	1	1
		1	0	1
		1	1	1
NOT		A	\overline{A}	
		0	1	
		1	0	
NAND		A	B	$\overline{A B}$
		0	0	1
		0	1	1
		1	0	1
		1	1	0
NOR		A	B	$\overline{A + B}$
		0	0	1
		0	1	0
		1	0	0
		1	1	0
XOR		A	B	$A \oplus B$
		0	0	0
		0	1	1
		1	0	1
		1	1	0
XNOR		A	B	$\overline{A \oplus B}$
		0	0	1
		0	1	0
		1	0	0
		1	1	1



Evaluation



SECTION – A

Choose the correct answer

- Which is a basic electronic circuit which operates on one or more signals?
(A) Boolean algebra (B) Gate
(C) Fundamental gates (D) Derived gates
- Which gate is called as the logical inverter?
(A) AND (B) OR
(C) NOT (D) XNOR
- $A + A = ?$
(A) A (B) O
(C) 1 (D) A
- NOR is a combination of ?
(A) NOT(OR) (B) NOT(AND)
(C) NOT(NOT) (D) NOT(NOR)
- NAND is called as Gate
(A) Fundamental Gate (B) Derived Gate
(C) Logical Gate (D) Universal gate



SECTION-B

Very Short Answers

- What is Boolean Algebra?
- Write a short note on NAND Gate.
- Draw the truth table for XOR gate.
- Write the associative laws?
- What are derived gates?

SECTION-C

Short Answers

- Write the truth table of fundamental gates.
- Write a short note on XNOR gate.
- Reason out why the NAND and NOR are called universal gates?
- Give the truth table of XOR gate.
- Write the De Morgan's law.

SECTION - D

Explain in detail

- Explain the fundamental gates with expression and truth table.
- How AND and OR can be realized using NAND and NOR gate.
- Explain the Derived gates with expression and truth table.