2

Data Representation and Boolean Algebra

Key concepts

- **Number systems**
  - Decimal, binary, octal, hexadecimal
- **Number conversions**
- **Binary arithmetic**
  - Addition, subtraction
- **Data representation**
  - Representation of integers and floating point numbers
  - Character representation: ASCII, EBCDIC, ISCII, UNICODE
  - Representation of audio, image and video data
- **Introduction to Boolean algebra**
  - Logic operators and logic gates
- **Basic postulates**
- **Basic theorems**
- **Circuit designing for simple Boolean expressions**
- **Universal gates**

Computer is a machine that can handle different types of data items. We feed data such as numbers, characters, images, videos and sounds to a computer for processing. We know that computer is an electronic device functioning on the basis of two electric states - ON and OFF. All electronic circuits have two states - open and closed. The two-state operation is called binary operation. Hence, the data given to computer should also be in binary form. In this chapter we will discuss various methods for representing data such as numbers, characters, images, videos and sounds.

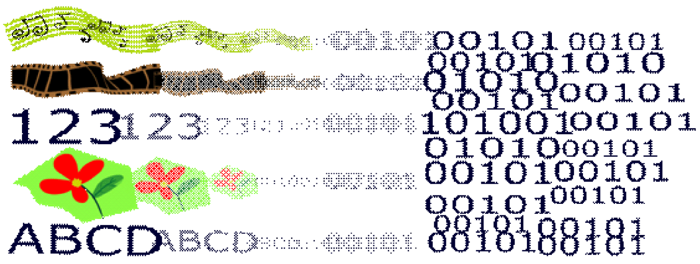


Fig. 2.1: External and internal form of data

Data representation is the method used internally to represent data in a computer**.**

Before discussing data representation of numbers, let us see what a number system is.

2.1 Number systems

A number is a mathematical object used to count, label and measure. A number system is a systematic way to represent numbers. The number system we use in our day to day life is the decimal number system that uses 10 symbols or digits. The number 289 is pronounced as two hundred and eighty nine and it consists of the symbols 2, 8 and 9. Similarly there are other number systems. Each has its own symbols and method for constructing a number. A number system has a unique base, which depends upon the number of symbols. The number of symbols used in a number system is called **base** or **radix** of a number system.

Let us discuss some of the number systems.

2.1.1 Decimal number system

The decimal number system involves ten symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9 to form a number. Since there are ten symbols in this number system, its base is 10. Therefore, the decimal number system is also known as base-10 number system.

Consider two decimal numbers 743 and 347

743 $\rightarrow$ seven hundred + four tens+ three ones ($7\times10^2 + 4\times10^1 + 3\times10^0$)

347 $\rightarrow$ three hundreds + four tens + seven ones ($3\times10^2 + 4 \times10^1 + 7\times10^0$)

Here, place value (weight) of 7 in first number 743 is $10^2 = 100$. But weight of 7 in second number 347 is $10^0=1$. The weight of a digit depends on its relative position. Such a number system is known as ***positional number system***. All positional number systems have a base and the place value of a digit is some power of this base.

Place value of each decimal digit is power of 10 (10^0, 10^1, 10^2, ...). Consider a decimal number 5876.

This number can be written in expanded form as

Weight	10^3	10^2	10^1	10^0
Decimal Number	5	8	7	6

$$
\begin{aligned}
&= 5\times10^3 + 8\times10^2 + 7\times10^1 + 6\times10^0 \\
&= 5\times1000 + 8\times100 + 7\times10 + 6\times1 \\
&= 5000 + 800 + 70 + 6 \\
&= 5876
\end{aligned}
$$

In the above example, the digit 5 has the maximum place value, 10^3=1000 and 6 has the minimum place value, 10^0=1. The digit with most weight is called Most Significant

Digit (MSD) and the digit with least weight is called Least Significant Digit (LSD). So in the above number MSD is 5 and LSD is 6.

Left most digit of a number is MSD and right most digit of a number is LSD

For fractional numbers weights are negative powers of 10 (10^{-1}, 10^{-2}, 10^{3}, ...) for the digits to the right of decimal point. Consider another example 249.367

Weight	10^2	10^1	10^0	10^{-1}	10^{-2}	10^{-3}
Decimal Number	2	4	9	3	6	7

MSD (.) LSD

$= 2\times10^2 + 4\times10^1 + 9\times10^0 + 3\times10^{-1} + 6\times10^{-2} + 7\times10^{-3}$
$= 2\times100 + 4\times10 + 9\times1 + 3\times0.1 + 6\times0.01 + 7\times0.001$
$= 200 + 40 + 9 + 0.3 + 0.06 + 0.007$
$= 249.367$

So far we have discussed a number system which uses 10 symbols. Now let us see the construction of other number systems with different bases.

2.1.2 Binary number system

A number system which uses only two symbols 0 and 1 to form a number is called binary number system. Bi means two. Base of this number system is 2. So it is also called base-2 number system. We use the subscript 2 to indicate that the number is in binary.

e.g. $(1101)_2$, $(101010)_2$, $(1101.11)_2$

Each digit of a binary number is called bit. A **bit** stands for **binary digit**.

The binary number system is also a positional number system where place value of each binary digit is power of 2. Consider an example $(1101)_2$. This binary number can be written in expanded form as shown below:

Weight	2^3	2^2	2^1	2^0
Binary Number	1	1	0	1

MSB LSB

$= 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$
$= 1\times8 + 1\times4 + 0\times2 + 1\times1$
$= 8 + 4 + 0 + 1$
$= 13$

The right most bit in a binary number is called Least significant Bit (**LSB**). The leftmost bit in a binary number is called Most significant Bit (**MSB**).

The binary number 1101 is equivalent to the decimal number 13. The number 1101 also exists in the decimal number system. But it is interpreted as one thousand one hundred and one. To avoid this confusion, base must be specified in all number systems other than decimal number system. The general format is

$$\textbf{(Number)}_{\textbf{base}}$$

This notation helps to differentiate numbers of different bases. So a binary number must be represented with base 2 as $(1101)_2$ and it is read as "one one zero one to the base two".

If no base is given in a number, it will be considered as decimal. In other words, specifying the base is not compulsory in decimal number.

For fractional numbers, weights are negative powers of 2 (2^{-1}, 2^{-2}, 2^{-3}, ...) for the digits to the right of the binary point. Consider an example $(111.011)_2$

Weight	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
Binary Number	1	1	1	0	1	1

MSB (.) LSB

$$= 1\times2^2 + 1\times2^1 + 1\times2^0 + 0\times2^{-1} + 1\times2^{-2} + 1\times2^{-3}$$

$$= 1\times4 + 1\times2 + 1\times1 + 0\times\frac{1}{2} + 1\times\frac{1}{4} + 1\times\frac{1}{8}$$

$$= 4 + 2 + 1 + 0 + 0.25 + 0.125$$

$$= 7.375$$

Importance of binary numbers in computers

We have seen that binary number system is based on two digits 1 and 0. The electric state ON can be represented by 1 and the OFF state by 0 as in Figure 2.2. Because of this, computer uses binary number system as the basic number system for data representation.

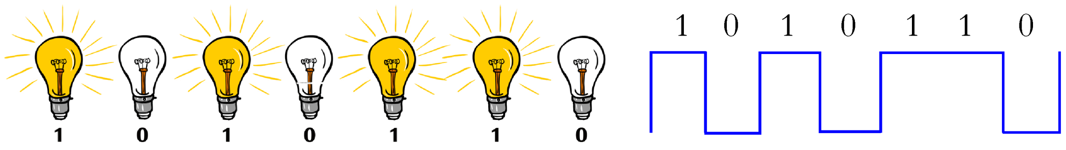


Fig. 2.2 : Digital representation of ON and OFF

2.1.3 Octal number system

A number system which uses eight symbols 0, 1, 2, 3, 4, 5, 6 and 7 to form a number is called octal number system. Octa means eight, hence this number system is called octal. Base of this number system is 8 and hence it is also called base-8 number system. Consider an example $(236)_8$. Weight of each digit is power of 8 (8^0, 8^1, 8^2, 8^3, ...). The number $(236)_8$ can be written in expanded form as

Weight	8^2	8^1	8^0
Octal Number	2	3	6

$$= 2\times8^2 + 3\times8^1 + 6\times8^0$$
$$= 2\times64 + 3\times8 + 6\times1$$
$$= 128 + 24 + 6$$
$$= 158$$

For fractional numbers weights are negative powers of 8, i.e. (8^{-1}, 8^{-2}, 8^{-3}, …) for the digits to the right of the octal point. Consider an example $(172.4)_8$

Weight	8^2	8^1	8^0	8^{-1}
Octal Number	1	7	2	4

$$= 1\times8^2 + 7\times8^1 + 2\times8^0 + 4\times8^{-1}$$
$$= 64 + 56 + 2 + 4\times\frac{1}{8}$$
$$= 122 + 0.5$$
$$= 122.5$$

2.1.4 Hexadecimal number system

A number system which uses 16 symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F to form a number is called hexadecimal number system. Base of this number system is 16 as there are sixteen symbols in this number system. Hence this number system is also called base-16 number system.

In this system, the symbols A, B, C, D, E and F are used to represent the decimal numbers 10, 11, 12, 13, 14 and 15 respectively. The hexadecimal digit and their equivalent decimal numbers are shown below.

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Consider a hexadecimal number $(12AF)_{16}$. Weights of each digit is power of 16 (16^0, 16^1, 16^2, ...).

This number can be written in expanded form as shown below:

Weight	16^3	16^2	16^1	16^0
Hexadecimal Number	1	2	A	F

$$= 1\times16^3 + 2\times16^2 + 10\times16^1 + 15\times16^0$$
$$= 1\times4096 + 2\times256 + 10\times16 + 15\times1$$
$$= 4096 + 512 + 160 + 15$$
$$= 4783$$

For fractional numbers, weights are some negative power of 16 (16^{-1}, 16^{-2}, 16^{-3}, ...) for the digits to the right of the hexadecimal point. Consider an example $(2D.4)_{16}$

Weight	16^1	16^0	16^{-1}
Hexadecimal	2	D	4

$$= 2\times16^1 + 13\times16^0 + 4\times\frac{1}{16}$$
$$= 32 + 13 + 0.25$$
$$= 45.25$$

Table 2.1 shows the base and symbols used in different number systems:

Number System	Base	Symbols used
Binary	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Table 2.1 : Number systems with base and symbols

Importance of octal and hexadecimal number systems

As we have discussed, digital hardware uses the binary number system for its operations and data. Representing numbers and operations in binary form requires too many bits and needs lot of effort. With octal, the bits are grouped in threes (because $2^3 = 8$) and with hexadecimal, the bits are grouped in four (because $2^4 = 16$) and these groups are replaced with the respective octal or hexadecimal symbol. This conversion processes of binary numbers to octal and hexadecimal number systems and vice versa are very easy. This short-hand notation is widely used in the design and operations of electronic circuits.

Check yourself

1. Number of symbols used in a number system is called _____.
2. Pick invalid numbers from the following
   i) $(10101)_8$ ii) $(123)_4$ iii) $(768)_8$ iv) $(ABC)_{16}$
3. Define the term 'bit'.
4. Find MSD in the decimal number 7854.25.
5. The base of hexadecimal number system is ________.

2.2 Number conversions

After having learnt the various number systems, let us now discuss how to convert the numbers of one base to the equivalent numbers in other bases. There are different types of number conversions like decimal to binary, binary to decimal, decimal to octal etc. This section discusses how to convert one number system to another.

2.2.1 Decimal to binary conversion

The method of converting decimal number to binary number is by repeated division. In this method the decimal number is successively divided by 2 and the remainders are recorded. The binary equivalent is obtained by grouping all the remainders, with the last remainder being the Most Significant Bit (MSB) and first remainder being the Least Significant Bit (LSB). In all these cases the remainders will be either 0 or 1 (binary digit).

Examples:

Find binary equivalent of decimal number 25.

2	25	Remainders	
2	12	1	LSB
2	6	0	
2	3	0	
2	1	1	
	0	1	MSB

$(25)_{10} = (11001)_2$

Find binary equivalent of $(80)_{10}$.

2	80	Remainders	
2	40	0	LSB
2	20	0	
2	10	0	
2	5	0	
2	2	1	
2	1	0	
	0	1	MSB

$(80)_{10} = (1010000)_2$

Hint: Binary equivalent of an odd decimal number ends with 1 and binary of even decimal number ends with zero.

Converting decimal fraction to binary

To convert a fractional decimal number to binary, we use the method of repeated multiplication by 2. At first the decimal fraction is multiplied by 2. The integer part of the answer will be the MSB of binary fraction. Again the fractional part of the answer is multiplied by 2 to obtain the next significant bit of binary fraction. The procedure is continued till the fractional part of product is zero or a desired precision is obtained.

Example: Convert 0.75 to binary.

	$0.75 \times 2 = 1.50$
1	$.50 \times 2 = 1.00$
1	$.00$

$(0.75)_{10} = (0.11)_2$

Example: Convert 0.625 to binary.

	$0.625 \times 2 = 1.25$
1	$.25 \times 2 = 0.50$
0	$.50 \times 2 = 1.00$
1	$.00$

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

Example: Convert 15.25 to binary.

Convert 15 to binary

		Remainders
2	15	
2	7	1
2	3	1
2	1	1
	0	1

Convert 0.25 to binary

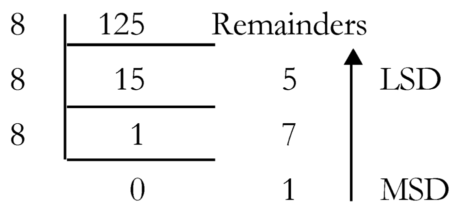
	$0.25 \times 2 = 0.50$
0	$.50 \times 2 = 1.00$
1	$.00$

$(15.25)_{10} = (1111.01)_2$

2.2.2 Decimal to octal conversion

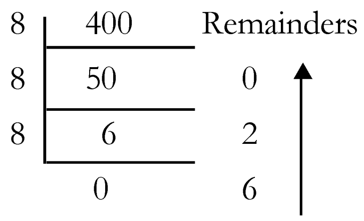
The method of converting decimal number to octal number is also by repeated division. In this method the number is successively divided by 8 and the remainders are recorded. The octal equivalent is obtained by grouping all the remainders, with the last remainder being the MSD and first remainder being the LSD. Remainders will be 0, 1, 2, 3, 4, 5, 6 or 7.

Example: Find octal equivalent of decimal number 125.



$$(125)_{10} = (175)_8$$

Example: Find octal equivalent of $(400)_{10}$.

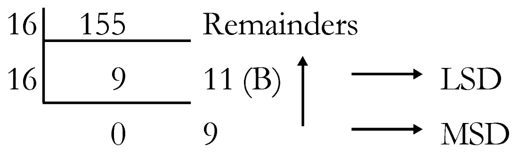


$$(400)_{10} = (620)_8$$

2.2.3 Decimal to hexadecimal conversion

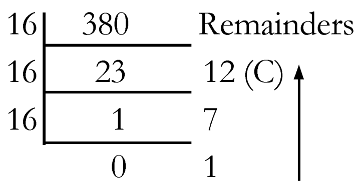
The method of converting decimal number to hexadecimal number is also by repeated division . In this method, the number is successively divided by 16 and the remainders are recorded. The hexadecimal equivalent is obtained by grouping all the remainders, with the last remainder being the Most Significant Digit (MSD) and first remainder being the Least Significant Digit(LSD). Remainders will be 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E or F.

Example: Find hexadecimal equivalent of decimal number 155.



$$(155)_{10} = (9B)_{16}$$

Example: Find hexadecimal equivalent of 380.



$$(380)_{10} = (17C)_{16}$$

2.2.4 Binary to decimal conversion

A binary number can be converted into its decimal equivalent by summing up the product of each bit and its weight. Weights are some power of 2 ($2^0, 2^1, 2^2, 2^3$, ...).

Example: Convert $(11011)_2$ to decimal.

Weight	2^4	2^3	2^2	2^1	2^0
Bit	1	1	0	1	1

$$\begin{aligned}(11011)_2 &= 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0\\ &= 16 + 8 + 2 + 1\\ &= 27\end{aligned}$$

$\boxed{(11011)_2 = (27)_{10}}$

Example: Convert $(1100010)_2$ to decimal.

Weight	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bit	1	1	0	0	0	1	0

$$\begin{aligned}(1100010)_2 &= 1\times2^6 + 1\times2^5 + 0\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0\\ &= 64 + 32 + 2\\ &= 98\end{aligned}$$

$\boxed{(1100010)_2 = (98)_{10}}$

Table 2.2 may help us to find powers of 2.

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

Table 2.2 : Powers of 2

Converting binary fraction to decimal

A binary fraction number can be converted into its decimal equivalent by summing up the product of each bit and its weight. Weights of binary fractions are negative powers of 2 ($2^{-1}, 2^{-2}, 2^{-3}$, ...) for the digits after the binary point.

Example: Convert $(0.101)_2$ to decimal.

Weight	2^{-1}	2^{-2}	2^{-3}
Bit	1	0	1

$$\begin{aligned}(0.101)_2 &= 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}\\ &= 0.5 + 0 + 0.125\\ &= 0.625\end{aligned}$$

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

Example: Convert $(1010.11)_2$ to decimal.

Weight	2^3	2^2	2^1	2^0
Bit	1	0	1	0

$$\begin{aligned}(1010)_2 &= 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0\\ &= 8 + 0 + 2 + 0\\ &= 10\end{aligned}$$

$(1010)_2 = (10)_{10}$

$(0.11)_2 = 1\times2^{-1} + 1\times2^{-2}$

$= 0.5 + 0.25$

$= 0.75$ $(0.11)_2 = (0.75)_{10}$

Weight	2^{-1}	2^{-2}
Bit	1	1

$\mathbf{(1010.11)_2 = (10.75)_{10}}$

Table 2.3 shows some negative powers of 2.

2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
0.5	0.25	0.125	0.0625	0.03125

Table 2.3 : Negative powers of 2

2.2.5 Octal to decimal conversion

An octal number can be converted into its decimal equivalent by summing up the product of each octal digit and its weight. Weights are some powers of 8 (8^0, 8^1, 8^2, 8^3, ...).

Example: Convert $(157)_8$ to decimal.

$(157)_8 = 1\times8^2 + 5\times8^1 + 7\times8^0$

$= 64 + 40 + 7$

$= 111$

Weight	8^2	8^1	8^0
Octal digit	1	5	7

$\mathbf{(157)_8 = (111)_{10}}$

Example: Convert $(1005)_8$ to decimal.

$(1005)_8 = 1\times8^3 + 0\times8^2 + 0\times8^1 + 5\times8^0$

$= 512+5$

$= 517$

Weight	8^3	8^2	8^1	8^0
Octal digit	1	0	0	5

$\mathbf{(1005)_8 = (517)_{10}}$

2.2.6 Hexadecimal to decimal conversion

An hexadecimal number can be converted into its decimal equivalent by summing up the product of each hexadecimal digit and its weight. Weights are powers of 16 (16^0, 16^1, 16^2, ...).

Example: Convert $(AB)_{16}$ to decimal.

$(AB)_{16} = 10\times16^1+11\times16^0$

$= 160 + 11$

$= 171$

Weight	16^1	16^0
Hexadecimal digit	A	B

A = 10 B = 11

$\mathbf{(AB)_{16} = (171)_{10}}$

Example: Convert $(2D5)_{16}$ to decimal.

$(2D5)_{16}$ $= 2\times16^2+13\times16^1+5\times16^0$

$= 512+208+5$

$= 725$

Weight	16^2	16^1	16^0
Hexadecimal digit	2	D	5

D = 13

$\boxed{(2D5)_{16} = (725)_{10}}$

2.2.7 Octal to binary conversion

An octal number can be converted into binary by converting each octal digit to its 3 bit binary equivalent. Eight possible octal digits and their binary equivalents are listed in Table 2.4.

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

Table 2.4 : Binary equivalent of octal digit

Example: Convert $(437)_8$ to binary.

3-bit binary equivalents of each octal digit are

$$\begin{array}{ccc} 4 & 3 & 7 \\ \downarrow & \downarrow & \downarrow \\ 100 & 011 & 111 \end{array}$$

$\boxed{(437)_8=(100011111)_2}$

Example: Convert $(7201)_8$ to binary.

3-bit binary equivalents of each octal digits are

$$\begin{array}{cccc} 7 & 2 & 0 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 111 & 010 & 000 & 001 \end{array}$$

$\boxed{(7201)_8 = (111010000001)_2}$

2.2.8 Hexadecimal to binary conversion

A hexadecimal number can be converted into binary by converting each hexadecimal digit to its 4 bit binary equivalent. Sixteen possible hexadecimal digits and their binary equivalents are listed in Table 2.5.

Example: Convert $(AB)_{16}$ to binary.

4-bit binary equivalents of each hexadecimal digit are

$$\begin{array}{cc} A & B \\ \downarrow & \downarrow \\ 1010 & 1011 \end{array}$$

$$\boxed{(AB)_{16} = (10101011)_2}$$

Example: Convert $(2F15)_{16}$ to binary.

4-bit binary equivalents of each hexadecimal digit are

$$\begin{array}{cccc} 2 & F & 1 & 5 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 0010 & 1111 & 0001 & 0101 \end{array}$$

$$\boxed{(2F15)_{16} = (10111100010101)_2}$$

Hexa decimal	Binary equivalent
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Table 2.5 : Binary equivalent of hexadecimal digits

2.2.9 Binary to octal conversion

A binary number can be converted into its octal equivalent by grouping binary digits to group of 3 bits and then each group is converted to its octal equivalent. Start grouping from right to left.

Example: Convert $(101100111)_2$ to octal.

We can group above binary number 101100111 from right as shown below.

$$\begin{array}{ccc} 101 & 100 & 111 \\ \downarrow & \downarrow & \downarrow \\ 5 & 4 & 7 \end{array}$$

$$\boxed{(101100111)_2 = (547)_8}$$

Example: Convert $(10011000011)_2$ to octal.

We can group above binary number 10011000011 from right as shown below.

$$\begin{array}{cccc} 010 & 011 & 000 & 011 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 2 & 3 & 0 & 3 \end{array}$$

After grouping, if the left most group has no 3 bits, then add leading zeros to form 3 bit binary.

$$\boxed{(10011000011)_2 = (2303)_8}$$

2.2.10 Binary to hexadecimal conversion

A binary number can be converted into its hexadecimal equivalent by grouping binary digits to group of 4 bits and then each group is converted to its hexadecimal equivalent. Start grouping from right to left.

Example: Convert $(101100111010)_2$ to hexadecimal.

We can group the given binary number 101100111010 from right as shown below:

$$\begin{array}{ccc} 1011 & 0011 & 1010 \\ \downarrow & \downarrow & \downarrow \\ \text{B} & 3 & \text{A} \end{array}$$

$$\boxed{\mathbf{(101100111010)_2 = (B3A)_{16}}}$$

Example: Convert $(110111100001100)_2$ to hexadecimal.

We can group the given binary number 110111100001100 from right as shown below:

After grouping, if the left most group has no 4 bits, then add leading zeros to form 4 bit binary.

$$\begin{array}{cccc} 0110 & 1111 & 0000 & 1100 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 6 & \text{F} & 0 & \text{C} \end{array}$$

$$\boxed{\mathbf{(110111100001100)_2 = (6F0C)_{16}}}$$

2.2.11 Octal to hexadecimal conversion

Conversion of an octal number to hexadecimal number is a two step process. Octal number is first converted into binary. This binary equivalent is then converted into hexadecimal.

Example: Convert $(457)_8$ to hexadecimal equivalent.

First convert $(457)_8$ into binary.

$$\begin{array}{lccc} (457)_8 = & 4 & 5 & 7 \\ & \downarrow & \downarrow & \downarrow \\ & 100 & 101 & 111 \end{array}$$

$$= (100101111)_2$$

Then convert $(100101111)_2$ into hexadecimal as follows:

$$\begin{array}{lccc} (100101111)_2 & = 0001 & 0010 & 1111 \\ & \downarrow & \downarrow & \downarrow \\ & = \ 1 & 2 & \text{F} \end{array}$$

$$= (12F)_{16}$$

$$\boxed{\mathbf{(457)_8 = (12F)_{16}}}$$

2.2.12 Hexadecimal to octal conversion

Conversion of an hexadecimal to octal number is also a two step process. Hexadecimal number is first converted into binary. This binary equivalent is then converted into octal.

Example: Convert $(A2D)_{16}$ into octal equivalent.

First convert $(A2D)_{16}$ into binary.

$$(A2D)_{16} = \begin{array}{ccc} A & 2 & D \\ \downarrow & \downarrow & \downarrow \\ 1010 & 0010 & 1101 \end{array}$$

$$= (101000101101)_2$$

Then convert $(101000101101)_2$ into octal as follows:

$$(101000101101)_2 = \begin{array}{cccc} 101 & 000 & 101 & 101 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 5 & 0 & 5 & 5 \end{array}$$

$$= (5055)_8$$

$$\boxed{\mathbf{(A2D)_{16} = (5055)_8}}$$

Table 2.6 shows procedures for various number conversions.

Conversion	Procedure
Decimal to Binary	Repeated division by 2 and grouping the remainders
Decimal to Octal	Repeated division by 8 and grouping the remainders
Decimal to Hexadecimal	Repeated division by 16 and grouping the remainders
Binary to Decimal	Multiply binary digit by place value(power of 2) and find their sum
Octal to Decimal	Multiply octal digit by place value (power of 8) and find their sum
Hexadecimal to Decimal	Multiply hexadecimal digit by place value (power of 16) and find their sum
Octal to Binary	Converting each octal digit to its 3 bit binary equivalent
Hexadecimal to Binary	Converting each hexadecimal digit to its 4 bit binary equivalent
Binary to Octal	Grouping binary digits to group of 3 bits from right to left
Binary to Hexadecimal	Grouping binary digits to group of 4 bits from right to left
Octal to Hexadecimal	Convert octal to binary and then binary to hexadecimal
Hexadecimal to Octal	Convert hexadecimal to binary and then binary to octal

Table 2.6 : Procedure for number conversions

Check yourself

1 Convert the decimal number 31 to binary.

2 Find decimal equivalent of $(10001)_2$

3 If $(x)_8 = (101011)_2$, then find x.

4 Fill in the blanks:

a) (________)$_2$ = $(AB)_{16}$

b) (___D__)$_{16}$ = (1010_____1000)$_2$

c) 0.25_{10} = (_______)$_2$

5 Find the largest number in the list

(i) $(1001)_2$ (ii) $(A)_{16}$ (iii) $(10)_8$ (iv) $(11)_{10}$

2.3 Binary arithmetic

As in the case of decimal number system, arithmetic operations are performed in binary number system. When we give instruction to add two decimal numbers, the computer actually adds their binary equivalents. Let us see how binary addition and subtraction are carried out.

2.3.1 Binary addition

The rules for adding two bits are as follows:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Note that a carry bit 1 is created only when two ones are added. If three ones are added (i.e. 1+1+1), then the sum bit is 1 with a carry bit 1.

Example: Find sum of binary numbers 1011 and 1001.

$$\begin{array}{r} 1\,0\,1\,1\ + \\ \underline{1\,0\,0\,1} \\ 1\,0\,1\,0\,0 \end{array}$$

Example: Find sum of binary numbers 110111 and 10011.

$$\begin{array}{r} 1\,1\,0\,1\,1\,1\ + \\ \underline{1\,0\,0\,1\,1\,0} \\ 1\,0\,1\,1\,1\,0\,1 \end{array}$$

2.3.2 Binary subtraction

The rules for subtracting a binary digit from another digit are as follows.

A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Note that when 1 is subtracted from 0 the difference is 1, but 1 is borrowed from immediate left bit of first number. The above rules can be used only when a small binary number is subtracted from a large binary number.

Example: Subtract $(10101)_2$ from $(11111)_2$.

```
1 1 1 1 1 –
1 0 1 0 1
0 1 0  1 0
```

Example: Subtract $(10111)_2$ from $(101000)_2$.

```
1 0 1 0 0 0 –
  1 0 1 1 1
  1 0 0 0 1
```

2.4 Data representation

Computer uses a fixed number of bits to represent a piece of data which could be a number, a character, image, sound, video etc. Data representation is the method used internally to represent data in a computer. Let us see how various types of data can be represented in computer memory.

2.4.1 Representation of numbers

Numbers can be classified into integer numbers and floating point numbers. Integers are whole numbers or numbers without any fractional part. A floating point number or a real number is a number with fractional part. These two numbers are treated differently in computer memory. Let us see how integers are represented.

a. Representation of integers

There are three methods for representing an integer number in computer memory. They are

i) Sign and magnitude representation

ii) 1's complement representation

iii) 2's complement representation

The following data representation methods are based on 8 bit word length.

A **word** is basically a fixed-sized group of bits that are handled as a unit by a processor. Number of bits in a word is called **word length**. The word length is the choice of computer designer and some popular word lengths are 8, 16, 32 and 64.

i. Sign and magnitude representation

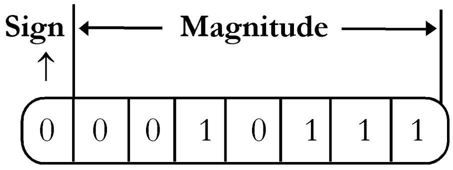
In this method, first bit from left (MSB) is used for representing sign of integer and remaining 7-bits are used for representing magnitude of integer. For negative integers sign bit is 1 and for positive integers sign bit is 0. Magnitude is represented as 7-bit binary equivalent of the integer.

Example: Represent + 23 in sign and magnitude form.

Number is positive, so first bit (MSB) is 0.

7 bit binary equivalent of 23 = $(0010111)_2$

So + 23 can be represented as $(00010111)_2$

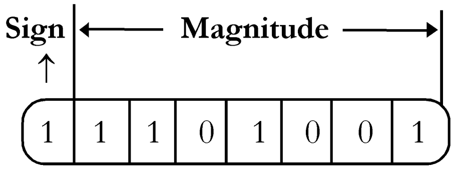


Example: Represent -105 in sign and magnitude form.

Number is negative, so first bit(MSB) is 1

7 bit binary equivalent of 105 = $(1101001)_2$

So -105 can be represented as $(11101001)_2$



Note: In this method an 8 bit word can represent 2^8-1 =255 numbers (i.e. -127 to +127). Similarly, a 16 bit word can represent 2^{16}-1 = 65535 numbers (i.e. -32767 to +32767). So, an *n*-bit word can represent 2^n-1 numbers i.e., $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$. The integer 0 can be represented in two ways: +0 = 00000000 and –0 = 10000000.

ii. 1's complement representation

In this method, first find binary equivalent of absolute value of integer. If number of digits in binary equivalent is less than 8, provide zero(s) at the left to make it 8-bit form. 1's complement of a binary number is obtained by replacing every 0 with 1 and every 1 with 0. Some binary numbers and the corresponding 1's compliments are given below:

Binary Number	1's Complement
11001	00110
10101	01010

If the number is negative it is represented as 1's complement of 8-bit form binary. If the number is positive, the 8-bit form binary equivalent itself is the 1's complement representation.

Example: Represent -119 in 1's complement form.

Binary of 119 in 8-bit form = $(01110111)_2$

-119 in 1's complement form = $(10001000)_2$

Example: Represent +119 in 1's complement form.

Binary of 119 in 8-bit form = $(01110111)_2$

+119 in 1's complement form = $(01110111)_2$

(No need to find 1's complement, since the number is positive)

Note: In this representation if first bit (MSB) is 0 then number is positive and if MSB is 1 then number is negative. So 8 bit word can represent integers from -127 (represented as 10000000) to +127 (represented as 01111111). Here also integer 0 can be represented in two ways: +0 = 00000000 and –0 = 11111111. An *n*-bit word can represent 2^n-1 numbers i.e. -(2^{n-1}-1) to +(2^{n-1}-1).

iii. 2's complement representation

In this method, first find binary equivalent of absolute value of integer and write it in 8-bit form. If the number is negative, it is represented as 2's complement of 8-bit form binary. If the number is positive, 8-bit form binary itself is the representation. 2's complement of a binary number is calculated by adding 1 to its 1's complement.

For example, let us find the 2's complement of $(10101)_2$.

1's complement of $(10101)_2$ = $(01010)_2$

So 2's complement of $(10101)_2$ = 01010 +
1
= $(01011)_2$

Example: Represent -38 in 2's complement form.

Binary of 38 in 8-bit form = $(00100110)_2$

-38 in 2's complement form = 11011001+
1
= $(11011010)_2$

Example: Represent +38 in 2's complement form.

Binary of 38 in 8-bit form = $(00100110)_2$

+38 in 2's complement form = $(00100110)_2$ (No need to find 2's complement)

Note: In this representation if first bit (MSB) is 0 then number is positive and if MSB is 1 then number is negative. Here integer 0 has only one way of representation and is 00000000. So an 8 bit word can represent integers from -128 (represented as 10000000) to +127(represented as 01111111). It is the most common integer representation. An *n*-bit word can represent 2^n numbers -(2^{n-1}) to + (2^{n-1}-1). Table 2.7 shows the comparison of different representation methods of integers in 8-bit word length.

Features	Sign & Magnitude	1's Complement	2's Complement	Remarks
Range	-127 to +127	-127 to +127	-128 to +127	Range is more in 2's complement
Total Numbers	255	255	256	
Representation of integer 0	Two ways of representation	Two ways of representation	Only one way of representation	In 2's complement there is no ambiguity in 0 representation
Representation of positive integers	Binary equivalent of integer in 8 bit form	Binary equivalent of integer in 8 bit form	Binary equivalent of integer in 8 bit form	All three forms are same.
Representation of negative integers	Sign bit 1 and magnitude is represented in 7 bit binary form	Find 1's complement of 8 bit form binary	Find 2's complement of 8 bit form binary	For all negative numbers MSB is 1

Table 2.7 : Comparison for representation of integers in 8-bit word length

Subtraction using complements

We have discussed how to subtract a binary number from another binary number. But to design and implement an electronic circuit for this method of subtraction is really complex and difficult. Circuitry for binary addition is simpler. So it is better if we can subtract through addition. For that we use the concept of complements. There are two methods of subtraction using complements.

Subtraction using 1's complement

The steps for subtracting a smaller binary number from a larger binary number are:

Step 1: Add 0s to the left of smaller number, if necessary, to make two numbers with same number of bits.

Step 2: Find 1's complement of subtrahend (Number to be subtracted, here small number)

Step 3: Add the complement with minuend (Number from which subtracting, here larger number)

Step 4: Add the carry 1 to the sum to get the answer.

Example: Subtract 100_2 from 1010_2 using 1's complement.

At first find 1's complement of 0100 and it is 1011

To compare the three types of representations let us consider the following table. For clarity and easy illustration, 4-bits are used to represent the numbers in this table .

Number	Sign & Magnitude	1's Complement	2's Complement
-8	Not possible	Not possible	1000
-7	1111	1000	1001
-6	1110	1001	1010
-5	1101	1010	1011
-4	1100	1011	1100
-3	1011	1100	1101
-2	1010	1101	1110
-1	1001	1110	1111
0	1000 or 0000	0000 or 1111	0000
1	0001	0001	0001
2	0010	0010	0010
3	0011	0011	0011
4	0100	0100	0100
5	0101	0101	0101
6	0110	0110	0110
7	0111	0111	0111

From this table, it is clear that the MSB of a binary number indicates the sign of the corresponding decimal number irrespective of the representation. That is, if the MSB is 1, the number is negative and if it is 0, the number is positive. The table also shows that only 2's complement method can represent the maximum numbers for a given number of bits. This fact reveals that, a number below -7 and above +7 cannot be represented using 4-bits in sign & magnitude form and 1's complement form. So we go for 8-bit representation. Similarly in 2's complement method, if we want to handle numbers outside the range -8 to +7, eight bits are required.

In 8-bits implementation, the numbers from -128 to +127 can be represented in 2's complement method. The range will be -127 to +127 for the other two methods. For the numbers outside this range, we use 16 bits and so on for all the representations.

```
Add it with larger number, i.e.          1010   +
                                         1011
                                       1 0101
              MSB                        0101   +    MSB is removed and
                                            1        added with the result
              The result is              0110
```

Subtraction using 2's complement

To subtract a smaller binary number from a larger binary number the following are the steps.

Step 1: Add 0s to the left of smaller number, if necessary, to make the two numbers have the same number of bits.

Step 2: Find 2's complement of subtrahend (Number to be subtracted, here the smaller number).

Step 3: Add the 2's complement with minuend (Number from which subtracting, here the larger number).

Step 3: Ignore the carry.

Example: Subtract $(100)_2$ from $(1010)_2$ using 2's complement.

```
2's complement of 0100                   1100
Add it with larger number, i.e.          1010   +
                                         1100
   Ignore the carry to                 1 0110
   get the result
              The result is               110
```

b. Representation of floating point numbers

A floating point number / real number consists of an integer part and a fractional part. A real number can be written in a special notation called the floating point notation. Any number in this notation contains two parts, ***mantissa*** and ***exponent*****.**

For example, 25.45 can be written as 0.2545×10^2, where 0.2545 is the mantissa and the power 2 is the exponent. (In normalised floating point notation mantissa is between 0.1 and 1). Similarly -0.0035 can be written as $-0.35\text{x}10^{-2}$, where -0.35 is mantissa and -2 is exponent.

Let us see how a real number is represented in 32 bit word length computer. Here 24 bits are used for storing mantissa (among these the first bit is for sign) and 8 bits are used for storing exponent (first bit for sign) as in Figure 2.3. Assume that decimal point is to the right of the sign bit of mantissa. No separate space is reserved for storing decimal point.

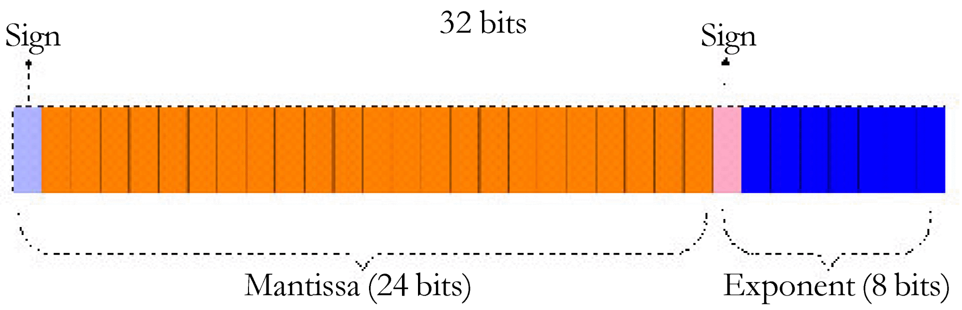


Fig 2.3: Representation of floating point numbers

Consider the real number 25.45 mentioned earlier, that can be written as 0.2545×10^2, where 0.2545 is the mantissa and 2 is the exponent. These numbers are converted into binary and stored in respective locations. Various standards are followed for representing mantissa and exponent. When word length changes, bits used for storing mantissa and exponents will change accordingly.

In real numbers, binary point keeps track of mantissa part and exponent part. Since the value of mantissa and exponent varies from number to number the binary point is not fixed. In other words it floats and hence such a representation is called floating point representation.

2.4.2 Representation of characters

We have discussed methods for representing numbers in computer memory. Similarly there are different methods to represent characters. Some of them are discussed below.

a. ASCII

The code called ASCII (pronounced "AS-key"), which stands for American Standard Code for Information Interchange, uses 7 bits to represent each character in computer memory. The ASCII representation has been adopted as a standard by the U.S. government and is widely accepted. A unique integer number is assigned to each character. This number called ASCII code of that character is converted into binary for storing in memory. For example, ASCII code of A is 65, its binary equivalent in 7-bit is 1000001. Since there are exactly 128 unique combinations of 7 bits, this 7-bit code can represent only128 characters.

Another version is ASCII-8, also called extended ASCII, which uses 8 bits for each character, can represent 256 different characters. For example, the letter A is represented by 01000001, B by 01000010 and so on. ASCII code is enough to represent all of the standard keyboard characters.

b. EBCDIC

It stands for Extended Binary Coded Decimal Interchange Code. This is similar to ASCII and is an 8 bit code used in computers manufactured by International Business Machine (IBM). It is capable of encoding 256 characters. If ASCII coded data is to be used in a computer which uses EBCDIC representation, it is necessary to transform ASCII code to EBCDIC code. Similarly if EBCDIC coded data is to be used in a ASCII computer, EBCDIC code has to be transformed to ASCII.

c. ISCII

ISCII stands for Indian Standard Code for Information Interchange or Indian Script Code for Information Interchange. It is an encoding scheme for representing various writing systems of India. ISCII uses 8-bits for data representation. It was evolved by a standardisation committee under the Department of Electronics during 1986-88, and adopted by the Bureau of Indian Standards (BIS). Nowadays ISCII has been replaced by Unicode.

d. Unicode

Using 8-bit ASCII we can represent only 256 characters. This cannot represent all characters of written languages of the world and other symbols. Unicode is developed to resolve this problem. It aims to provide a standard character encoding scheme, which is universal and efficient. It provides a unique number for every character, no matter what the language and platform be.

Unicode originally used 16 bits which can represent up to 65,536 characters. It is maintained by a non-profit organisation called the Unicode Consortium. The Consortium first published the version 1.0.0 in 1991 and continues to develop standards based on that original work. Nowadays Unicode uses more than 16 bits and hence it can represent more characters. Unicode can represent characters in almost all written languages of the world.

2.4.3 Representation of audio, image and video

In the previous sections we have discussed different data representation techniques and standards used for the computer representation of numbers and characters. While we attempt to solve real life problems with the aid of a digital computer, in most cases we may have to represent and process data other than numbers and characters. This may include audio data, images and videos. We can see that like numbers and characters, the audio, image and video data also carry information. In this section we will see different file formats for storing sound, image and video.

Digital audio, image and video file formats

Multimedia data such as audio, image and video are stored in different types of files. The variety of file formats is due to the fact that there are quite a few approaches to compressing the data and a number of different ways of packaging the data. For example an image is most popularly stored in Joint Picture Experts Group (JPEG) file format. An image file consists of two parts - header information and image data. Information such as name of the file, size, modified data, file format, etc. are stored in the header part. The intensity value of all pixels is stored in the data part of the file.

The data can be stored uncompressed or compressed to reduce the file size. Normally, the image data is stored in compressed form. Let us understand what compression is. Take a simple example of a pure black image of size 400×400 pixels. We can repeat the information black, black, ..., black in all 16,0000 (400×400) pixels. This is the uncompressed form, while in the compressed form black is stored only once and information to repeat it 1,60,000 times is also stored. Numerous such techniques are used to achieve compression. Depending on the application, images are stored in various file formats such as bitmap file format (BMP), Tagged Image File Format (TIFF), Graphics Interchange Format (GIF), Portable (Public) Network Graphic (PNG).

What we said about the header file information and compression is also applicable for audio and video files. Digital audio data can be stored in different file formats like WAV, MP3, MIDI, AIFF, etc. An audio file describes a format, sometimes referred to as the 'container format', for storing digital audio data. For example WAV file format typically contains uncompressed sound and MP3 files typically contain compressed audio data. The synthesised music data is stored in MIDI(Musical Instrument Digital Interface) files. Similarly video is also stored in different files such as AVI (Audio Video Interleave) - a file format designed to store both audio and video data in a standard package that allows synchronous audio with video playback, MP3, JPEG-2, WMV, etc.

Check yourself

1. Which is the MSB of representation of -80 in the sign and magnitude method?
2. Write 28.756 in mantissa exponent form.
3. ASCII stands for ______________.
4. Represent -60 in 1's complement form.
5. Define Unicode.
6. List any two image file formats.

2.5 Introduction to Boolean algebra

In many situations in our life we face questions that require 'Yes' or 'No' answers. Similarly much of our thinking process involves answering questions with 'Yes' or 'No'. The way of finding truth by answering such two-valued questions is known as human reasoning or logical reasoning. These values can be expressed as 'True' or 'False' and numerically 1 or 0. These values are known as binary values or Boolean values. Boolean algebra is the algebra of logic which is a part of mathematical algebra that deals with the operations on variables that represent the values 1 and 0. The name Boolean algebra is given to honour the British mathematician George Boole, as he was the person who established the link between logic and mathematics. His revolutionary paper 'An Investigation of the laws of thought' led to the development of Boolean algebra.

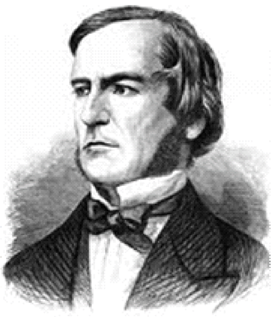


Fig. 2.4: George Boole (1815 - 1864)

2.5.1 Binary valued quantities

Let us consider the following:

1. Should I take an umbrella?
2. Will you give me your pen?
3. George Boole was a British mathematician.
4. Kerala is the biggest state in India.
5. Why were you absent yesterday?
6. What is your opinion about Boolean algebra?

1st and 2nd sentences are questions which can be answered as YES or NO. These cases are called binary decisions and the results are called binary values. The 3rd statement is TRUE and 4th statement is FALSE. But 5th and 6th sentences cannot be answered like the cases above. The sentences which can be determined to be TRUE or FALSE are called ***logical statements*** or truth functions and the results TRUE or FALSE are called binary values or logical constants. The **logical constants** are represented by 1 and 0, where 1 stands for TRUE and 0 for FALSE. The variables which can store (hold) logical constants 1 and 0 are called logical variables or ***Boolean variables***.

2.5.2 Boolean operators and logic gates

We have already seen that data fed to a computer must be converted into a combination of 1s and 0s. All data, information and operations are represented inside the computer using 0s and 1s. The operations performed on these Boolean values are called **Boolean operations**. As we know, operators are required to perform these operations. These operators are called Boolean operators or logical operators. There are three basic logical operators in Boolean algebra. These operators and their operations are as follows:

OR	→ Logical Addition
AND	→ Logical Multiplication
NOT	→ Logical Negation

The first two operators require two operands and the third requires only one operand. Here the operands are always Boolean variable or constants and the result will always be either True (1) or False (0).

Computers perform these operations with some electronic circuits, called logic circuits. A logic circuit is made up of individual units called gates, where a gate represents a Boolean operation. A **Logic gate** is a physical device that can perform logical operations on one or more logical inputs and produce a single logical output. Logic gates are primarily implemented using diodes or transistors acting as electronic switches. There are three basic logic gates and they represent the three basic Boolean operations. These gates are OR, AND and NOT.

a. The OR operator and OR gate

Let us consider a real life situation. When do you use an umbrella? When it rains, isn't it? And of course, if it is too sunny. We can combine these two situations using a compound statement like "If it is raining or if it is sunny, we use an umbrella". Note down the use of ***or*** in this statement. The interpretation of this statement can be shown as in Table 2.8. The logical reasoning of the use of umbrella in our example very much resembles the Boolean OR operation.

Raining	Sunny	Need Umbrella
No	No	No
No	Yes	Yes
Yes	No	Yes
Yes	Yes	Yes

Table 2.8: Logical OR operation

The OR operator performs logical addition and the symbol used for this operation is + (plus). The expression A + B is read as A OR B. Table 2.9 is the truth table that represents the OR operation. Assume that the variables A and B are the inputs (operands) and A + B is the output (result). It is clear from the truth table that, if any one of the inputs is 1 (True), the output will be 1 (True).

Truth Table is a table that shows Boolean operations and their results. It lists all possible inputs for the given operation and their corresponding output. Usually these operations consist of operand variables and operators. The operands and the operators together are called Boolean expression. Truth Table represents all possible values of the operands and the corresponding results (values) of the operation. A Boolean expression with m operands (variables) and n operators require 2^m rows and $m + n$ columns.

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Table 2.9 : Truth table of OR operation

While designing logic circuits, the logic gate used to implement logical OR operation is called logical **OR gate**. Figure 2.5 shows the OR gate symbol in Boolean algebra.

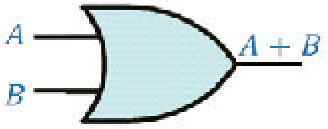


Fig. 2.5 : Logical OR gate

The working of this gate can be illustrated with an electronic circuit. Figure 2.6 illustrates the schematic circuit of parallel switches which shows the idea of an OR gate. Here A and B are two switches and Y is a bulb. Each switch and the bulb can take either close (ON) or open (OFF) state. Now let us relate the operation of the above circuit with the functioning of OR gate. Assume that OFF represents the logical LOW state (say 0) and ON represents the logical HIGH (say 1) state. If we consider the state of switches A and B as input to the OR gate and state of bulb as output of OR gate, then the truth table shown in Table 2.9 will describe the operation of an OR gate. Thus the Boolean expression for OR gate can be written as: $Y = A + B$

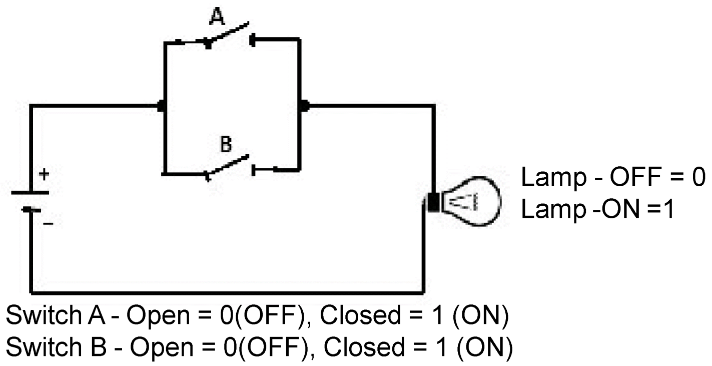

A
B
+
-
Lamp - OFF = 0
Lamp -ON =1
Switch A - Open = 0(OFF), Closed = 1 (ON)
Switch B - Open = 0(OFF), Closed = 1 (ON)

Fig. 2.6 : Circuit with two switches and a bulb for parallel connection

An OR gate can take more than two inputs. Let us see what will be the truth table, Boolean expression and logical symbol for the three input OR gate.

The truth table and the gate symbol shows that, the Boolean expression for the OR gate with three inputs is Y= A + B + C. Figure 2.7 shows the representation of OR gate with three inputs. From the truth tables 2.9 and 2.10, we can see that the output of OR gate is 1 if any input is 1; and output is 0 if and only if all inputs are 0.

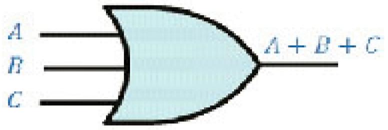


Fig 2.7 : OR gate with three inputs

A	B	C	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

Table 2.10 : Truth table for OR gate with 3 inputs

b. The AND operator and AND gate

We will discuss another situation to understand the concept of AND Boolean operation. Suppose you are away from home and it is lunch time. You can have your food only if two conditions are satisfied – (i) there should be a hotel and (ii) you should have enough money. Here also, we can make a compound statement like "If there is a hotel and if we have money, we can have food". Note the use of ***and*** in this statement. Table 2.11 shows the logical reasoning of getting food and it very much resembles the Boolean AND operation.

Hotel	Money	Take Food
No	No	No
No	Yes	No
Yes	No	No
Yes	Yes	Yes

Table 2.11 : Logical AND operation

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.12 : Truth table of AND operation

The AND operator performs logical multiplication and the symbol used for this operation is **.** (dot). The expression A **.** B is read as A AND B. Table 2.12 is the truth table that represents the AND operation. Assume that the variables A and B are the inputs (operands) and A **.** B is the output (result).

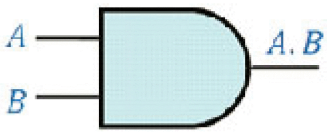


Fig. 2.8 : Logical AND gate

While designing logic circuits, the logic gate used to implement logical AND operation is called logical **AND gate**. Figure 2.8 shows the AND gate symbol in Boolean algebra.

The working of this gate can be illustrated with an electronic circuit shown in Figure 2.9. This schematic circuit has two serial switches which illustrates the idea of an AND gate. Here A and B are two switches and Y is a bulb. Each switch and the bulb can take either close (ON) or open (OFF) state. Now let us relate the operation of the above circuit with the functioning of AND gate. Assume that OFF represents the logical LOW state (say 0) and ON represents the logical HIGH state (say 1). If we consider the state of switches A and B as input to the AND gate and state of bulb as output of AND gate, then the Boolean expression for AND gate can be written as: Y = A . B

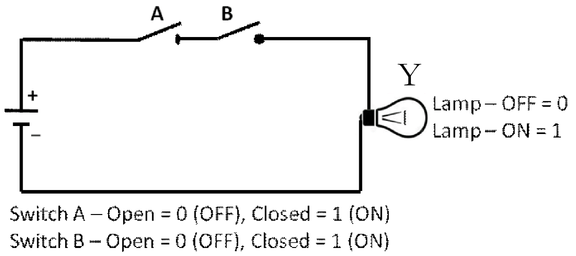

A
B
+
-
Y
Lamp – OFF = 0
Lamp – ON = 1
Switch A – Open = 0 (OFF), Closed = 1 (ON)
Switch B – Open = 0 (OFF), Closed = 1 (ON)

Fig. 2.9 : Circuit with two switches and a bulb for serial connection

An AND gate can take more than two inputs. Let us see what will be the truth table, Boolean expression and logical symbol for three input AND gate. The truth table and the gate symbol shows that the Boolean expression for the AND gate with three inputs is Y = A . B . C

A	B	C	A.B.C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 2.13 : Truth table for AND gate with 3 inputs

Figure 2.10 shows the representation of AND gate with three inputs. From Truth Tables 2.12 and 2.13, we can see that the output of AND gate is 0 if any input is 0; and output is 1 if and only if all inputs are 1.

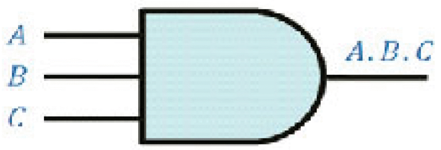


Fig. 2.10 : AND gate with three inputs

c. The NOT operator and NOT gate

Let us discuss another case to familiarise the Boolean NOT operation. Suppose you jog everyday in the morning. Can you do it every day? If it rains, can you jog in the morning? Table 2.14 shows all the possibilities of this situation. It is quite similar to Boolean NOT operation.

Raining	Jogging
No	Yes
Yes	No

Table 2.14: Logical NOT

It is a unary operator and hence it requires only one operand. The NOT operator performs logical negation and the symbol used for this operation is - (over-bar).

The expression $\overline{A}$ is read as A bar . It is also expressed as A^{I} and read as A dash. Table 2.15 is the truth table that represents the NOT operation. Assume that the variable A is the input (operand) and $\overline{A}$ is the output (result). It is clear from the truth table that, the output will be the opposite value of the input. The logic gate used to implement NOT operation is NOT gate. Figure 2.11 shows the NOT gate symbol.

A	$\overline{A}$
0	1
1	0

Table 2.15 : Truth table of NOT operation

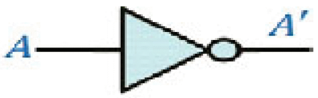


Fig. 2.11 : NOT gate

A NOT gate is also called ***inverter***. It has only one input and one output. The input is always changed into its opposite state. If input is 0 , the NOT gate will give its complement or opposite which is 1. If the input is 1, then the NOT gate will complement it to 0.

Check yourself

1. Define the term Boolean variable.
2. A logic circuit is made up of individual units called ________.
3. Name the logical operator/gate which gives high output if and only if all the inputs are high.
4. Define the term truth table.
5. An AND operation performs logical _________ and an OR operation performs logical ________.
6. Draw the logic symbol of OR gate.

2.6 Basic postulates of Boolean algebra

Boolean algebra being a system of mathematics, consists of certain fundamental laws. These fundamental laws are called postulates. They do not have proof, but are made to build solid framework for scientific principles. On the other hand, there are some theorems in Boolean algebra which can be proved based on these postulates and laws.

Postulate 1: Principles of 0 and 1

If $A \neq 0$, then $A = 1$ and if $A \neq 1$, then $A=0$

Postulate 2: OR Operation (Logical Addition)

$0 + 0 = 0 \qquad 0 + 1 = 1 \qquad 1 + 0 = 1 \qquad 1 + 1 = 1$

Postulate 3: AND Operation (Logical Multiplication)

$0 . 0 = 0 \qquad 0 . 1 = 0 \qquad 1 . 0 = 0 \qquad 1 . 1 = 1$

Postulate 4: NOT Operation (Logical Negation or Compliment Rule)

$\overline{0} = 1 \qquad \overline{1} = 0$

Principle of Duality

When Boolean variables and/or values are combined with Boolean operators, Boolean expressions are formed. X + Y and $\overline{A}$ + 1 are examples of Boolean expressions. The postulates 2, 3 and 4 are all Boolean statements. Consider the statements in postulate 2. If we change the value 0 by 1 and 1 by 0, and the operator OR (+) by AND (.), we will get the statements in postulate 3. Similarly, if we change the value 0 by 1 and 1 by 0, and the operator AND (.) by OR (+) in statements of postulate 3, we will get the statements of postulate 2. This concept is known as principle of duality.

The principle of duality states that for a Boolean statement, there exists its dual form, which can be derived by

(i) changing each OR sign (+) to AND sign (.)

(ii) changing each AND sign (.) to OR sign (+)

(iii) replacing each 0 by 1 and each 1 by 0

2.7 Basic theorems of Boolean algebra

There are some standard and accepted rules in every theory. The set of rules are known as ***axioms*** of the theory. A conclusion can be derived from a set of presumptions by using these axioms or postulates. This conclusion is called law or theorem. Theorems of Boolean algebra provide tools for simplification and manipulation of Boolean expressions. Let us discuss some of these laws or theorems. These laws or theorems can be proved using truth tables and Boolean laws that are already proved.

2.7.1 Identity law

If X is a Boolean variable, the law states that:

(i)	$0 + X = X$	(ii)	$1 + X = 1$
(iii)	$0 . X = 0$	(iv)	$1 . X = X$

Statements (i) and (ii) are known as additive identity law; and statements (iii) and (iv) are called multiplicative identity. Also note that, statement (iv) is the dual of (i) and vice versa. Similarly, statements (ii) and (iii) are dual forms. The truth tables shown in Tables 2.16(a), 2.16(b), 2.17(a) and 2.17 (b) prove these laws.

0	X	0 + X
0	0	0
0	1	1

Table 2.16 (a) : Additive Identity law

1	X	1 + X
1	0	1
1	1	1

Table 2.16 (b) : Additive Identity law

Table 2.16 (a) shows that columns 2 and 3 are the same and it is proved that 0 + X = X. Similarly, columns 1 and 3 of table 12.16 (b) are the same and hence the statement 1+X=1 is true.

0	X	0 . X
0	0	0
0	1	0

Table 2.17(a) : Multiplicative Identity law

1	X	1 . X
1	0	0
1	1	1

Table 2.17(b) : Multiplicative Identity law

Table 2.17 (a) shows that columns 1 and 3 are the same and it is proved that 0 . X = 0. Similarly, columns 2 and 3 of Table 2.17 (b) are the same and hence the statement 1. X=X is true.

2.7.2 Idempotent law

The idempotent law states that: (i) X + X = X

and (ii) X . X = X

If the value of X is 0, the statements are true, because 0 + 0 = 0 (*Postulate 2*) and 0 . 0 = 0 (*Postulate 3*). Similarly the statements will be true when the value of X is 1. Truth Tables 2.18 (a) and 2.18 (b) shows the proof of these laws. Also note that the statements are dual to each other.

X	X	X + X
0	0	0
1	1	1

Table 2.18 (a) : Idempotent law

X	X	X . X
0	0	0
1	1	1

Table 2.18 (b) : Idempotent law

2.7.3 Involution law

This law states that: $\overline{\overline{X}} = X$

Let X = 0, then $\overline{X}$ = 1 (*Postulate 4*); and if we take its compliment, $\overline{\overline{X}} = \overline{1} = 0$, which is same as X. The statement will also be true, when the value of X is 1.

Columns 1 and 3 of Table 2.19 show that $\overline{\overline{X}} = X$.

X	$\overline{X}$	$\overline{\overline{X}}$
0	1	0
1	0	1

Table 2.19 : Involution law

2.7.4 Complimentary law

The complimentary law states that: (i) $X + \overline{X} = 1$

and (ii) $X . \overline{X} = 0$

If the value of X is 0, then $\overline{X}$ becomes 1. Hence, $X + \overline{X}$ becomes 0 + 1, which results into 1 (*Postulate 2*). Similarly when X is 1, $\overline{X}$ will be 0. The truth tables 2.20 (a) and 2.20 (b) show the proof of these laws taking all the possibilities. Also note that the statements are dual to each other.

X	$\overline{X}$	$X + \overline{X}$
0	1	1
1	0	1

Table 2.20 (a) : Complimentary law

X	$\overline{X}$	$X . \overline{X}$
0	1	0
1	0	0

Table 2.20 (b) : Complimentary law

2.7.5 Commutative law

Commutative law allows to change the position of variables in OR and AND operations. If X and Y are two variables, the law states that:

(i) $X + Y = Y + X$

and (ii) $X . Y = Y . X$

The truth table shown in Tables 2.21 (a) and 2.21 (b) prove these statements.

X	Y	X + Y	Y + X
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Table 2.21 (a) : Commutative law

X	Y	X.Y	Y.X
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Table 2.21 (b) : Commutative law

The law ensures that order of the operands for OR and AND operations does not affect the output in each case.

2.7.6 Associative law

In the case of three operands for OR and AND operations, associative law allows grouping of operands differently. If X, Y and Z are three variables, the law states that:

(i) $X + (Y + Z) = (X + Y) + Z$

and (ii) $X . (Y . Z) = (X . Y) . Z$

The truth tables shown in Tables 2.22(a) and 2.22(b) prove these statements.

X	Y	Z	X + Y	Y + X	X+(Y + Z)	(X+Y)+Z
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Table 2.22(a) : Associative law 1

In Table 2.22(a), columns 6 and 7 show that X + (Y + Z) = (X + Y) + Z. Columns 6 and 7 of Table 2.22(b) show the validity of the experience X . (Y . Z) = (X . Y) . Z

X	Y	Z	X . Y	Y . Z	X . (Y . Z)	(X . Y) . Z
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

Table 2.22(b) : Associative law 2

Associative law ensures that the order and combination of variables in OR (logical addition) or AND (logical multiplication) operations do not affect the final output.

2.7.7 Distributive law

Distributive law states that Boolean expression can be expanded by multiplying terms as in ordinary algebra. It also supports expansion of addition operation over multiplication. If X, Y and Z are variables, the law states that:

(i) X . (Y + Z) = X . Y + X . Z

and (ii) X + Y . Z = (X + Y) . (X + Z)

The following truth tables prove these statements:

X	Y	Z	Y + Z	X.(Y+Z)	X.Y	X.Z	X.Y + X.Z
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Table 2.23(a) : Distribution of Multiplication over Addition

Columns 5 and 8 of Table 2.23(a) show that $X . (Y + Z) = X . Y + X . Z$

X	Y	Z	Y . Z	X + Y . Z	X+Y	X+Z	(X+Y) . (X+Z)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Table 2.23(b) : Distribution of Addition over Multiplication

Columns 5 and 8 of Table 2.23(b) show that $X + Y . Z = (X + Y) . (X + Z)$

We are familiar with the first statement in ordinary algebra. To remember the second statement of this law, find the dual form of the first.

2.7.8 Absorption law

Absorption law is a kind of distributive law in which two variables are used and the result will be one of them. If X and Y are variables, the absorption law states that:

(i) $X + (X . Y) = X$

and (ii) $X . (X + Y) = X$

The truth tables shown Tables 2.24(a) and 2.24(b) prove the validity of the statements of absorption law.

X	Y	X . Y	X+(X.Y)
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Table 2.24 (a) : Absorption law

X	Y	X+Y	X.(X+Y)
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Table 2.24 (b) : Absorption law

Columns 1 and 4 of Table 2.24(a) and columns 1 and 4 of Table 2.24(b) show that the laws are true.

Table 2.25 depicts all the Boolean laws we have discussed so far.

No.	Boolean Law	Statement 1	Statement 2
1	Additive Identity	$0 + X = X$	$1 + X = 1$
2	Multiplicative Identity	$0 . X = 0$	$1 . X = X$
3	Idempotent Law	$X + X = X$	$X . X = X$
4	Involution Law	$\overline{\overline{X}} = X$	
5	Complimentary Law	$X + \overline{X} = 1$	$X . \overline{X} = 0$
6	Commutative Law	$X + Y = Y + X$	$X . Y = Y . X$
7	Associative Law	$X + (Y + Z) = (X + Y) + Z$	$X . (Y . Z) = (X . Y) . Z$
8	Distributive Law	$X . (Y + Z) = X . Y + X . Z$	$X + (Y . Z) = (X+Y) . (X+Z)$
9	Absorption Law	$X + (X . Y) = X$	$X . (X + Y) = X$

Table 2.25 : Boolean laws

The laws we discussed have been proved using truth tables. Some of them can be proved by applying some other laws. This method of proof is called algebraic proof. Let us see some of them.

i. To prove that X . (X + Y) = X – Absorption law

LHS $= X . (X + Y)$

$= X . X + X . Y$ *(Distribution of multiplication over addition)*

$= X + X . Y$ *(Idempotent law)*

$= X . (1 + Y)$ *(Distribution of multiplication over addition)*

$= X . 1$	*(Additive identity)*
$= X$	*(Multiplicative identity)*
$= \text{RHS}$	

ii. To prove that $X + (X . Y) = X$ – Absorption law

LHS	$= X + (X . Y)$	
	$= X . 1 + X . Y$	*(Multiplicative identity)*
	$= X . (1 + Y)$	*(Distribution of multiplication over addition)*
	$= X . 1$	*(Additive identity)*
	$= X$	*(Multiplicative identity)*
	$= \text{RHS}$	

iii. To prove that $X + (Y . Z) = (X+Y) . (X+Z)$ – Distributive Law

Let us take the expression on the RHS of this statement.

$(X+Y) . (X+Z)$

$= (X+Y) . X + (X+Y) . Z$	*(Distribution of multiplication over addition)*
$= X . (X+Y) + Z . (X+Y)$	*(Commutative law)*
$= X . X + X . Y + Z . X + Z . Y$	*(Distribution of multiplication over addition)*
$= X + X . Y + Z . X + Z . Y$	*(Idempotent law)*
$= X . 1 + X . Y + Z . X + Z . Y$	*(Multiplicative identity)*
$= X . (1 + Y) + Z . X + Z . Y$	*(Distribution of multiplication over addition)*
$= X . 1 + Z . X + Z . Y$	*(Additive identity)*
$= X . (1 + Z) + Z . Y$	*(Distribution of multiplication over addition)*
$= X . 1 + Z . Y$	*(Additive identity)*
$= X + Y . Z$	*(Multiplicative identity and Commutative law)*
$= \text{LHS}$	

The expression obtained is the LHS of the given statement. Thus the theorem is proved.

2.8 De Morgan's theorems

Augustus De Morgan (1806 – 1871), a famous logician and mathematician of University College, London proposed two theorems to simplify complicated Boolean expressions. These theorems are known as De Morgan's theorems. The two theorems are:

(i) $\overline{X+Y} = \overline{X} \cdot \overline{Y}$

(ii) $\overline{X.Y} = \overline{X} + \overline{Y}$

Literally these theorems can be stated as

(i) "the complement of sum of Boolean variables is equal to product of their individual complements" and

(ii) "the complement of product of Boolean variables is equal to sum of their individual complements".

Algebraic proof of the first theorem

We have to prove that, $\overline{X+Y} = \overline{X} \cdot \overline{Y}$

Let us assume that, $Z = X + Y$ ______ (1)

Then, $\overline{Z} = \overline{X+Y}$ ______ (2)

We know that, by complimentary law, the equations (3) and (4) are true.

$Z + \overline{Z} = 1$ ______(3)

$Z \,.\, \overline{Z} = 0$ ______(4)

Substituting expressions (1) in (3) and (2) in (4), we will get equations (5) and (6).

$(X + Y) + (\overline{X+Y}) = 1$ ______(5)

$(X + Y) \,.\, (\overline{X+Y}) = 0$ ______(6)

For the time being let us assume that De Morgan's first theorem is true. If so, $(\overline{X+Y})$ in equations (5) and (6) can be substituted with $(\overline{X} \cdot \overline{Y})$. Thus equations (5) and (6) can be modified as follows:

$(X+ Y) + (\overline{X} \,.\, \overline{Y}) = 1$ ______(7)

$(X + Y) \,.\, (\overline{X}.\overline{Y}) = 0$ ______(8)

Now we will prove equations (7) and (8) separately. If they are correct, we can conclude that the assumptions we made to form those equations are also correct. That is, if equations (7) and (8) are true, De Morgan's theorem is also true.

Consider the LHS of equation (7),

$(X + Y) + (\overline{X}.\overline{Y}) = (X + Y + \overline{X}) \,.\, (X + Y + \overline{Y})$ *(Distributive Law)*

$= (X + \overline{X} + Y) \,.\, (X + Y + \overline{Y})$ *(Associative Law)*

$= (1 + Y) \,.\, (X + 1)$ *(Complimentary Law)*

$= 1 . 1$ *(Additive Identity)*

$= 1$

$= \text{RHS}$

Now, let us consider the LHS of equation (8),

$(X + Y) . (\overline{X} . \overline{Y}) = (X . \overline{X} . \overline{Y}) + (Y . \overline{X} . \overline{Y})$ *(Distributive Law)*

$= (X . \overline{X} . \overline{Y}) + (Y . \overline{Y} . \overline{X})$ *(Associative Law)*

$= (0 . \overline{Y}) + (0 . \overline{X})$ *(Complimentary Law)*

$= 0 + 0$ *(Multiplicative Identity)*

$= 0$

$= \text{RHS}$

We have algebraically proved equations (7) and (8), which mean that De Morgan's first theorem is proved. The theorem can also be proved using truth table, but it is left to you as an exercise.

Algebraic proof of the second theorem

We have to prove that, $\overline{X . Y} = \overline{X} + \overline{Y}$

Let us assume that, $Z = X . Y$ ______ (1)

Then, $\overline{Z} = \overline{X . Y}$ ______ (2)

We know that, by complimentary laws the equations (3) and (4) are true.

$Z + \overline{Z} = 1$ ______ (3)

$Z . \overline{Z} = 0$ ______ (4)

Substituting expressions (1) in (3) and (2) in (4), we will get the expressions (5) and (6).

$(X . Y) + (\overline{X . Y}) = 1$ ______ (5)

$(X . Y) . (\overline{X . Y}) = 0$ ______ (6)

For the time being let us assume that De Morgan's second theorem is true. If so, $(\overline{X . Y})$ in equations 5 and 6 can be substituted with $(\overline{X} + \overline{Y})$. Thus equations (5) and (6) can be modified as follows:

$(X . Y) + (\overline{X} + \overline{Y}) = 1$ ______ (7)

$(X . Y) . (\overline{X} + \overline{Y}) = 0$ ______ (8)

Now we will prove equations (7) and (8) separately. If they are correct, we can conclude that the assumptions we made to form those equations are also correct. That is, if equations (7) and (8) are true, De Morgan's theorem is also true.

Consider the LHS of equation (7),

$$\begin{array}{lll}
(X\,.\,Y) + (\overline{X} + \overline{Y}) & = (\overline{X} + \overline{Y}) + (X\,.\,Y) & \textit{(Commutative Law)} \\
 & = (\overline{X} + \overline{Y} + X)\,.\,(\overline{X} + \overline{Y} + Y) & \textit{(Distributive Law)} \\
 & = (\overline{X} + X + \overline{Y})\,.\,(\overline{X} + \overline{Y} + Y) & \textit{(Associative Law)} \\
 & = (1 + \overline{Y})\,.\,(\overline{X} + 1) & \textit{(Complimentary Law)} \\
 & = 1\,.\,1 & \textit{(Additive Identity)} \\
 & = 1 & \\
 & = \text{RHS} &
\end{array}$$

Now, let us consider the LHS of equation (8),

$$\begin{array}{lll}
(X\,.\,Y)\,.\,(\overline{X} + \overline{Y}) & = (X\,.\,Y\,.\,\overline{X}) + (X\,.\,Y\,.\,\overline{Y}) & \textit{(Distributive Law)} \\
 & = (X\,.\,\overline{X}\,.\,Y) + (X\,.\,Y\,.\,\overline{Y}) & \textit{(Associative Law)} \\
 & = (0\,.\,Y) + (X\,.\,0) & \textit{(Complimentary Law)} \\
 & = 0 + 0 & \textit{(Multiplicative Identity)} \\
 & = 0 & \\
 & = \text{RHS} &
\end{array}$$

We have algebraically proved equations (7) and (8), which mean that De Morgan's second theorem is proved. The theorem can also be proved using truth table, but it is left to you as an exercise.

We can extend Demorgan's theorem for any number of variables as shown below:

$$\overline{A + B + C + D + ...} = \overline{A}\,.\,\overline{B}\,.\,\overline{C}\,.\,\overline{D}\,.\,.......$$

$$\overline{A.B.C.D.....} = \overline{A} + \overline{B} + \overline{C} + \overline{D} + ...$$

Although the identities above represent De Morgan's theorem, the transformation is more easily performed by following the steps given below:

(i) Complement the entire function

(ii) Change all the ANDs (.) to ORs (+) and all the ORs (+) to ANDs (.)

(iii) Complement each of the individual variables.

This process is called *demorganisation* and simply demorganisation is '*Break the line, change the sign*'.

Check yourself

1. Find the dual of Boolean expression A.B+B.C=1
2. Name the law which states that A+A=A.

   (a) Commutative law (b) Idempotent Law (c) Absorption Law
3. State De Morgan's theorems.

2.9 Circuit designing for simple Boolean expressions

By using basic gates, circuit diagrams can be designed for Boolean expressions. We have seen that the Boolean expressions A.B is represented using an AND gate, A+B is represented using an OR gate and $\overline{A}$ is represented using a NOT gate. Let us see how a circuit is designed for other Boolean expressions.

Consider a boolean expression $\overline{A}+B$, which is an OR operation with two input and first input is inverted. So circuit diagram can be drawn as shown in Figure 2.12.

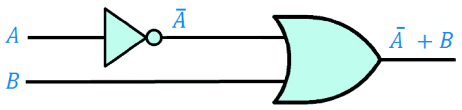


Fig 2.12 : f (A, B) = $\overline{A}+B$

Example: Construct a logical circuit for Boolean expression $f(X, Y) = X.Y + \overline{Y}$

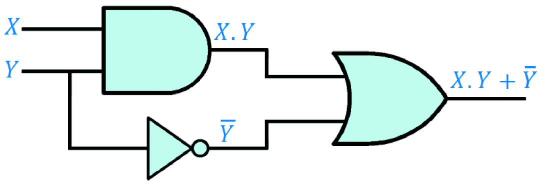


Fig 2.13 : f (X, Y) = X.Y + $\overline{Y}$

Example: Construct a logical expression for $f(a, b) = (a + b) . (\overline{a} + \overline{b})$

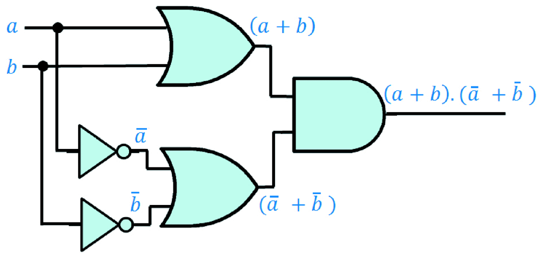


Fig 2.14 : f (a, b) = (a + b) . $(\overline{a} + \overline{b})$

Example: Construct logical circuit for the Boolean expression $\overline{a}.b + a.\overline{b}$

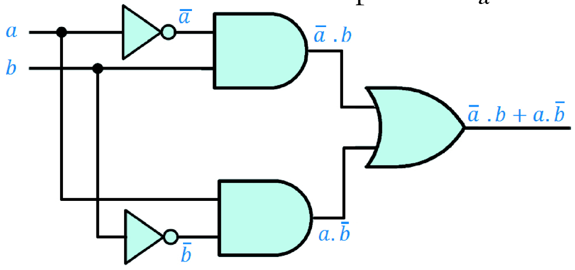


Fig 2.15 : f (a, b) = $\overline{a}.b + a.\overline{b}$

2.10 Universal gates

The NAND and NOR gates are called universal gates. A universal gate is a gate which can implement any Boolean function without using any other gate type. In practice, this is advantageous, since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in most of IC digital logic families.

2.10.1 NAND gate

This is an AND gate with its output inverted by a NOT gate. The logical circuit arrangement is shown in Figure 2.16.

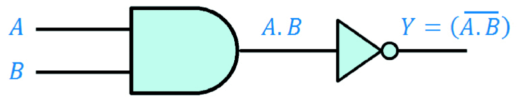


Fig. 2.16 : Circuit realisation of NAND gate

Note that A and B are the inputs of AND gate and its output is (A.B). The output of AND gate is inverted by an inverter (NOT gate) to get the resultant output Y as $(\overline{A.B})$. So the logical expression for a NAND gate is $(\overline{A.B})$. From the truth table shown as Table 2.26, we can see that output of a NAND gate is 1 if any one of the input is 0. It produces output 0 if and only if all inputs are 1. This is the inverse operation of an AND gate. So we can say that ***a NAND gate is an inverted AND gate.***

A	B	Y= $(\overline{A.B})$
0	0	1
0	1	1
1	0	1
1	1	0

Table 2.26 : NAND truth table

The logical symbol of NAND gate is shown in Figure 2.17. Note that the NAND symbol is an AND symbol with a small bubble at the output. The bubble is sometimes called an invert bubble.

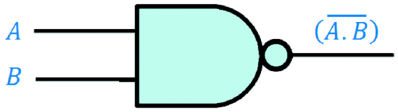


Fig. 2.17 : NAND gate

2.10.2 NOR gate

This is an OR gate with its output inverted by a NOT gate. The logical circuit arrangement is shown in Figure 2.18.

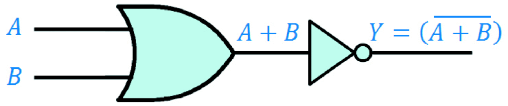


Fig. 2.18 : Circuit realisation of NOR gate

Note that A and B are the input of OR gate and its output is (A+B). The output of OR gate is inverted by an inverter (NOT gate) to get resultant output as $(\overline{A+B})$. So the logical expression for a NOR gate is $(\overline{A+B})$. Let us see the truth table of the two input NOR gate.

From the truth table shown on as Table 2.27, we can see that output of a NOR gate is 1 if and only if all inputs are 0. If any one of the inputs is 1 it produces an output 0. This is the inverse operation of an OR gate. So we can say that ***a NOR gate is an inverted OR gate.***

A	B	Y= $\overline{(A+B)}$
0	0	1
0	1	0
1	0	0
1	1	0

Table 2.27 : NOR truth table

The logical symbol of NOR gate is shown in Figure 2.19. Note that the NOR symbol is an OR symbol with a small bubble at the output.

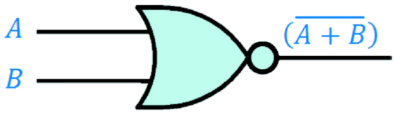


Fig. 2.19 : NOR gate

2.10.3 Implementation of basic gates using NAND and NOR

We can design all basic gates (AND, OR and NOT) using NAND or NOR gate alone. Let us see the implementation of basic gates using NAND gate.

NOT gate using NAND gate

We can implement a NOT gate (inverter) using a NAND by applying the same signal to both inputs of a NAND gate as shown in Figure 2.20.

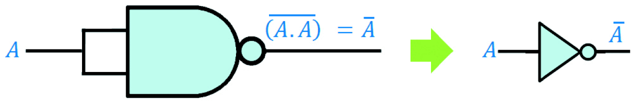


Fig. 2.20 : NOT gate using NAND gate

Proof:

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

= $\bar{A}$ Since A.A = A

The truth table shown as Table 2.28 is the proof for obtaining NOT gate using NAND gate.

A	AA	$\overline{(A.A)}$	$\bar{A}$
0	0	1	1
1	1	0	0

Table 2.28 : Proof using truth table

AND gate using NAND gate

We can implement an AND gate by using a NAND gate followed by another NAND gate to invert the output as shown in Figure 2.21.

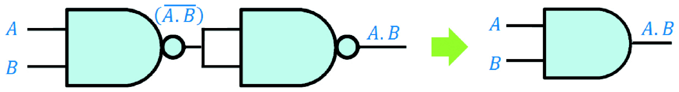


Fig. 2.21 : AND gate using NAND gate

Proof

We know that A NAND B $= (\overline{A.B})$

(A NAND B) NAND (A NAND B) $= (\overline{A.B})$ NAND $(\overline{A.B})$

$= \overline{((\overline{AB}).(\overline{AB}))}$

$= (\,\overline{\overline{(AB)}}\,)$ Since A.A = A

$=$ A.B Since $\overline{\overline{(A)}} = A$

Table 2.29 shows the proof for obtaining AND gate using NAND gate with the help of the truth table.

A	B	A.B	$(\overline{A.B})$	$(\overline{A.B}).(\overline{A.B})$	$\overline{((\overline{AB}).(\overline{AB}))}$
0	0	0	1	1	0
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	0	1

Table 2.29 : Proof using truth table

OR gate using NAND gate

The OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate inverters as shown in Figure 2.22.

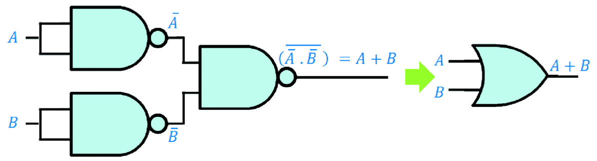


Fig. 2.22 : OR gate using NAND gate

Proof:

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

$= \overline{A}$

Similarly, B NAND B $= \overline{B}$

Therefore, (A NAND A) NAND (B NAND B) $= \overline{A}$ NAND $\overline{B}$

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

$= \overline{\overline{A}} + \overline{\overline{B}}$ Since $(\overline{A.B}) = \overline{A} + \overline{B}$

$= A + B$ Since $\overline{\overline{(A)}} = A$

Table 2.30 shows the proof for obtaining OR gate using NAND gate with the help of truth table.

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

Table 2.30 : Proof using truth table

Thus, the NAND gate is a universal gate since it can implement AND, OR and NOT operations. Now let us see implementation of basic gates by using another universal gate, the NOR gate.

NOT gate using NOR gate

We can implement a NOT gate (inverter) using a NOR by applying the same signal to both inputs of a NOR gate as shown in Figure 2.23.

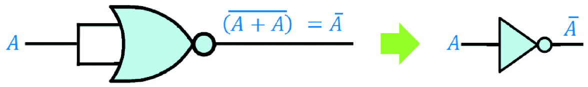


Fig 2.23 : NOT gate using NOR gate

Proof:

A NOR A $= \overline{(A+A)}$

$= \overline{A}$ Since A+A=A

Table 2.31 shows the proof for obtaining NOT gate using NOR gate with the help of truth table.

A	A+A	$\overline{(A+A)}$	$\overline{A}$
0	0	1	1
1	1	0	0

Table 2.31 : Proof using truth table

OR gate using NOR gate

We can implement an OR gate by using a NOR gate followed by another NOR gate to invert the output as shown in Figure 2.24.

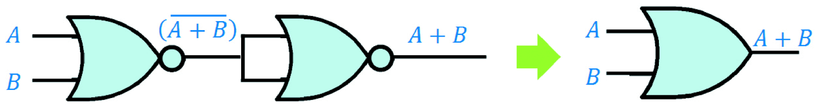


Fig 2.24: OR gate using NOR gate

Proof:

We know that A NOR B $= \overline{(A+B)}$

(A NOR B) NOR (A NOR B) $= \overline{(A+B)}$ NOR $\overline{(A+B)}$

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

$= ((\overline{\overline{A+B}}))$ Since A+A=A

$= A+B$ Since $(\overline{\overline{A}})=A$

Table 2.32 shows the proof of obtaining OR gate using NOR gate with the help of the truth table.

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

Table 2.32 : Proof using truth table

AND gate using NOR gate

The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters as shown in Figure 2.25.

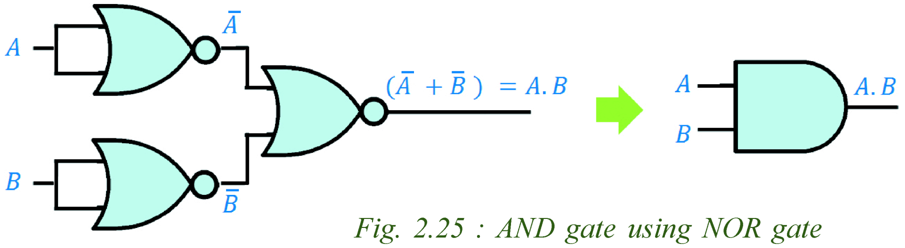


Fig. 2.25 : AND gate using NOR gate

Proof

A NOR A $= (\overline{A+A}) = \overline{A}$

Similarly, B NOR B $= (\overline{B+B}) = \overline{B}$

Therefore, (A NOR A) NOR (B NOR B) $= \overline{A}$ NOR $\overline{B}$

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

$= (\overline{\overline{A}}).(\overline{\overline{B}})$ Since $(\overline{A+B}) = \overline{A}.\overline{B}$

$= A.B$ Since $\overline{\overline{A}} = A$

Thus, the NOR gate is also a universal gate since it can implement the AND, OR and NOT operations. Table 2.33 represents the proof for obtaining AND gate using NOR gate with the help of truth table.

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

Table 2.33 : Proof using truth table

Check yourself

1. Draw logic circuits for the Boolean expression $X+\overline{Y}$.
2. Which gates are called universal gates?
3. ______ gate produces low (0) output if any one of the input is high(1)
   (a) OR (b) AND (c) NAND (d) NOR
4. A NAND B = ____________ .
   (a) A+B (b) A.B (c) $(\overline{A+B})$ (d) $(\overline{A.B})$

Let us sum up

Different methods of data representation were discussed in this chapter. Before discussing data representation of numbers, we introduced different number systems and their conversions. After the discussion of integer and floating point number representation we have mentioned different methods for character, sound, image and sound data representation. We have also discussed the concept of Boolean algebra in detail including logical operators, logic gates and laws of Boolean algebra. We concluded the chapter by introducing methods for designing basic logic circuits and by discussing the importance of universal gates in circuit designing.

Learning outcomes

After the completion of this chapter the learner will be able to

- explain the characteristics of different number systems.
- convert one number system to another.
- perform binary arithmetic.
- represent numbers and characters in computer memory.
- list the formats of sound, image and video file formats.
- identify the concept of Boolean algebra.
- explain the working of logical operators and logic gates with the help of examples.
- state and prove basic postulates and laws of Boolean algebra.
- design circuits for simple Boolean expressions.
- implement basic gates using universal gates.

Sample questions

Very short answer type

1. What is the place value of 9 in $(296)_{10}$?
2. Find octal equivalent of the decimal number 55.
3. Find missing terms in the following series

   a) 101_2, 1010_2, 1111_2, ______, ______

   b) 15_8, 16_8, 17_8, ______, ______.

   c) 18_{16}, $1A_{16}$, $1C_{16}$, ______, ______.
4. If $(X)_2 - (1010)_2 = (1000)_2$ then find X.
5. Name the coding system that can represent almost all the characters used in the human languages in the world.
6. Find out the logical statement(s) from the following .

   a) Why are you late?

   b) Will you come with me to market ?

   c) India is my country.

   d) Go to class room.
7. List three basic logic gates.
8. Which gate is called inverter?
9. List two complimentarity Laws.
10. The Boolean expression $\overline{(A+B)}$ represents ______ gate.

    a) AND b) NOR c) OR d) NAND

Short answers type

1. Define the term data representation.
2. What do you mean by a number system? List any four number systems.
3. Convert the following numbers into the other three number systems:

   a) $(125)_8$ b) 98 c) $(101110)_2$ d) $(A2B)_{16}$
4. Find the equivalents of the given numbers in the other three number systems.

   a) $(7F.1)_{16}$ b) $(207.13)_8$ c) 93.25 d) $(10111011.1101)_2$
5. If $(X)_2 = (Y)_8 = (Z)_{16} = (28)_{10}$ Then find X, Y and Z.
6. Arrange the following numbers in descending order

   a) $(101)_{16}$ b) $(110)_{10}$ c) $(111000)_2$ d) $(251)_8$
7. Find X , if $(X)_2 = (10111)_2 + (11011)_2 - (11100)_2$

8. What are the methods of representing integers in computer memory?
9. Represent the following numbers in sign and magnitude method, 1's complement method and 2's complement method

   a) -19 b) +49 c) -97 d) -127
10. Find out the integer which is represented as $(10011001)_2$ in sign and magnitude method.
11. Explain the method of representing a floating point number in 32 bit computer.
12. What are the methods of representing characters in computer memory?
13. Briefly explain the significance of Unicode in character representation.
14. Match the following:

A		B	
i)	If any input is 1 output is 1	a)	NAND
ii)	If an input is 0 output is 0	b)	OR
iii)	If any input is 0 output is 1	c)	NOR
iv)	If any input is 1 output is 0	d)	AND

15. Find dual of following Boolean expressions

   a) $X.Y+Z$ b) $A.C+A.1+A.C$ c) $(A+0).(A.1.\overline{A})$
16. Find complement of following Boolean expressions

   a) $\overline{A}\ \overline{B}$ b) $\overline{A.B}+\overline{C.D}$
17. Construct logic circuit for the following Boolean expression.

   (i) $\overline{a}\overline{b}+c$ (ii) $ab+\overline{a}b+\overline{ab}$ (iii) $(a+\overline{b}).(\overline{a}+\overline{b})$
18. Why are NAND and NOR gates called universal gates? Justify with an example.

Long answer type

1. Briefly explain different methods for representing numbers in computer memory.
2. Briefly explain different methods for representing characters in computer memory.
3. What are the file formats for storing image, sound and video data?
4. Give logic symbol, Boolean expression and truth table for three input AND gate.
5. Prove that NOR gate is a universal gate by implementing all the basic gates.