Ministry of high Education and Scientific Research
Southern Technical University
Technological institute of Basra
Department of Computer Networks and Software Techniques

# Learning package

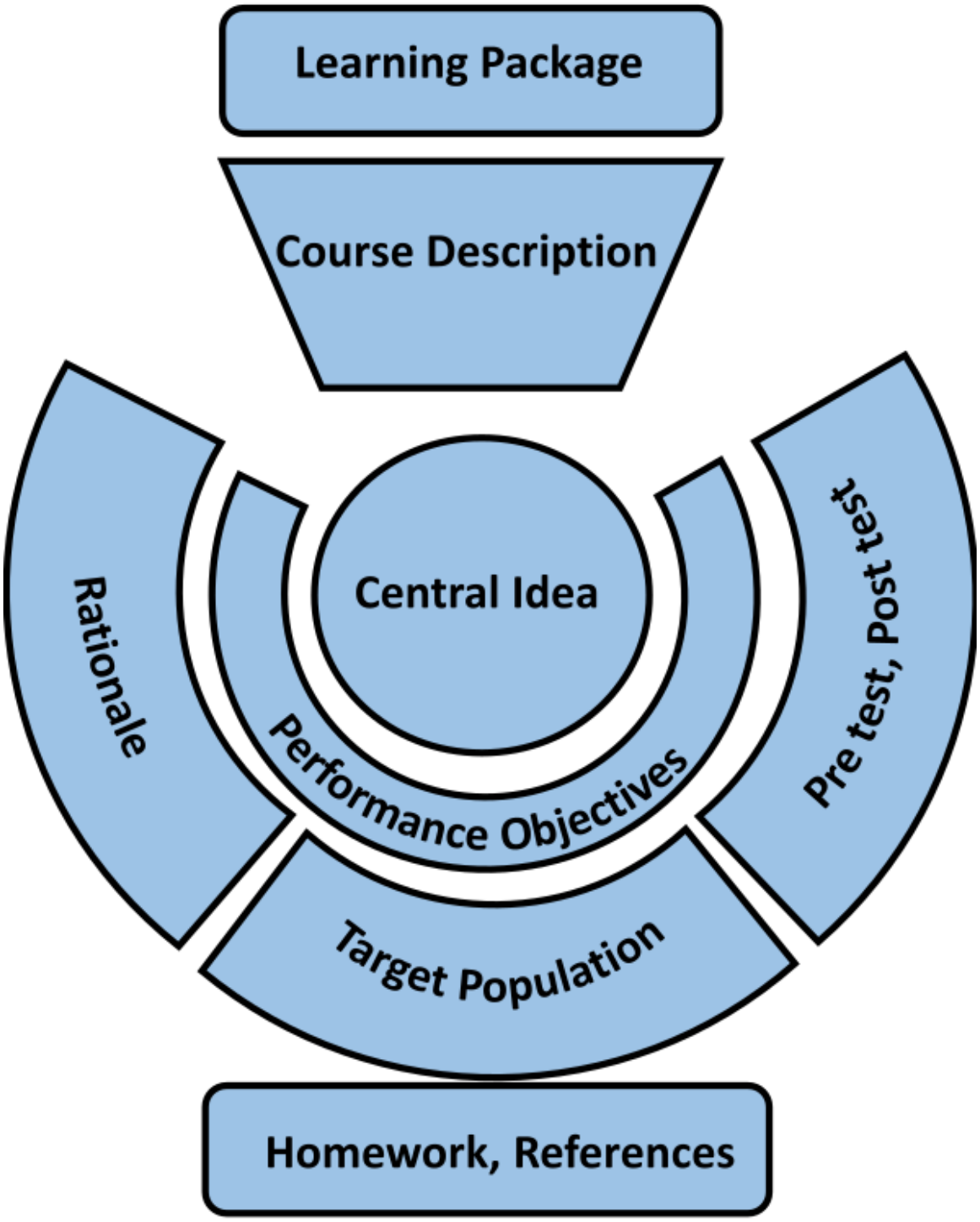
# Logic Design

For

First year students

By

**Ethar Abduljabbar Hadi**
**Assistant Lecturer**
**Dep. Of Computer Networks and Software Techniques**
**2025**


Learning Package
Course Description
Rationale
Central Idea
Performance Objectives
Pre test, Post test
Target Population
Homework, References

# Course Description

| |
|---|
| Course Name: |
| Logic Design |
| Course Code: |
| |
| Semester / Year: |
| First Semester/ First year |
| Description Preparation Date: |
| 9/5/2025 |
| Available Attendance Forms: |
| Attendance only |
| Number of Credit Hours (Total) / Number of Units (Total) |
| 60 hours (theoretical + practical) at a rate of 4 hours per week (2 theoretical + 2 practical) |
| Course administrator's name (mention all, if more than one name) |
| Name: Ethar Abduljabbar Hadi<br>Email: ethar.hadi@stu.edu.iq |
| Course Objectives |
| 1. Teaching the student the components of the computer<br>2. Teaching the student the types of numerical systems and how to convert between them.<br>3. Teaching the student about logic gates and how to design logical circuits<br>4. Teaching the student how to simplify logical circuits using the Boolean algebra method and the Karnaugh map method.<br>5. Teaching the student how to design the logical circuits found in the calculator in their simple and complex types.<br>6. Teaching the student how to examine and represent these logical circuits |

using special programs for that.

7. Developing Basic Understanding of digital circuits: Enabling students to understand the fundamental principles of digital circuits, including basic electronic components such as adders, digital gates, and comparator.
8. Developing Practical Skills: Providing hands-on training through laboratory experiments, allowing students to acquire the skills necessary to build and test electronic circuits.
9. Enhancing Critical Thinking: Encouraging students to engage in critical and analytical thinking when solving problems related to electronics.

Teaching and Learning Strategies

1. Cooperative concept planning education strategy.
2. Brainstorming Teaching Strategy.
3. Note-taking Sequence Strategy.

Course Structure

| Weeks | Hours | Required Learning Outcomes | Unit or subject name | Learning method | Evaluation method |
|---|---|---|---|---|---|
| 1<br>2<br><br>3<br>4<br><br>5<br>6<br>7<br>8<br>9<br>10 | 4hours<br>4hours<br><br>4hours<br>4hours<br><br>4hours<br>4hours<br>4hours<br>4hours<br>4hours<br>4hours | **1.**Understanding digital logic circuits Applicatio<br>**2.**Developing Critical Thinking and Problem-Solving Skills through Circuit Analysis and Fault Detection.<br>**4.**Analyzing logic Circuits | **1-** Number Systems<br>**2-** Conversion among Number Systems<br>**3.** Binary operations<br>**4.** SubtractionUsing Complement<br>**5.** Logic Gates<br>**6.** Boolean Algebra<br>**7.** De Morgan's Theorem<br>**8.** Standard Forms<br>**9.**Karnaugh Map<br>**10.**Digital Binary Adders | **1.**Conducting laboratory experime to build and test digital circuits. This enhances theoretical understanding and develops practic skills.<br>**2.**Seeking feedback from instructors and peers to identify strengths and weaknesses. | Daily, Weekly, Mid term Exams, and Final Term Exam. |

| | | | | | |
|---|---|---|---|---|---|
| 11<br>12<br>13<br>14 | 4hours<br>4hours<br>4hours<br>4hours | | **11.** Digital Binary Subtractors<br>**12.** Flip-Flops<br>**13.** Flip-Flops<br>**14-15.** Shift Registers | **3.**Reviewing concept periodically and applying them to new problems to reinforce memory and understanding.<br><br>**4.**Using educational software and interactive applications to bette understand concepts such as **circuit simulations**<br>**5.**Encouraging self-research on new topics in electronics and exploring recent developments | |

## Course Evaluation

Distributing the score out of 100 as follows:
20 points for Midterm Theoretical Exams.
20 points for Midterm Practical Exams.
10 points for Daily Exams, homework and Assessment.
50 points for the Final Exam.

## Learning and Teaching Resources

| | |
|---|---|
| Required textbooks (curricular books, if any) | Holdsworth, Brian, and Clive Woods. Digital logi design. Elsevier, 2002. |
| Main references (sources) | Alam, Mansaf, and Bashir Alam. Digital Logic Design. PHI Learning Pvt. Ltd., 2015. |
| Recommended books and references (scientific journals, reports…) | Dally, William James, and R. Curtis Harting. Digit design: a systems approach. Cambridge Univers Press, 2012. |
| Electronic References, Websites | https://www.geeksforgeeks.org/digital-logic/digital-electronics-logic-design-tutorials/<br>https://www.electronics-tutorials.ws/_1.html<br>https://www.geeksforgeeks.org/digital-logic/flip-flop-types-their-conversion-and-applications/#sr-flip-flop |

**Ministry of high Education and Scientific Research**
**Southern Technical University**
**Technological institute of Basra**
**Department of Computer Networks and Software Techniques**

# Learning package

**In**

# Numbers Systems

**For**

Students of First Year

**By**
**Ethar Abduljabbar Hadi**
**Assistant Lecturer**
**Dep. Of Computer Networks and Software Techniques**
**2025**

# 1/ Overview

## 1 / A –Target population :-

For First year students
Technological institute of Basra
Dep. Of Computer Networks and Software Techniques

## 1 / B –Rationale :-

Understanding number systems is crucial for gaining comprehensive knowledge of how data is represented, processed, and transmitted in digital systems, and enabling students to efficiently design, configure, troubleshoot, and secure computer systems and networks., which is why I have created this unit .

## 1 / C –Central Idea :-

1 – Types of numbers systems

2 – Representation of numbers systems

3 –Conversion numbers systems .

4 – Binary code

5 -The operations on Binary

6- Representation methods.

## 1 / D – Performance Objectives

After studying the first unit, the student will be able to:-

1. Know the types of Number systems
2. Representations of numbers systems
3. Convert between the types of numbers of systems
4. Know about Binary code.
5. Understand the operations on Binary.
6. Understand the representation methods.

## 2/ Pretest :

# Why do we need to study the number system?

# 3/ Numbers Systems :-

The numeric system we use daily is the decimal system, but this system is not convenient for machines since the information is handled codified in the shape of on or off bits; this way codifying takes us to the necessity of knowing the positional calculation which will allow us to express a number in any base where we need it.

## 1.Radix number systems

A base of a number system or radix defines the range of values that a digit may have.

## A-Decimal Number System :-

This system is composed of 10 numbers or symbols, these 10 symbols are:

0 1 2 3 4 5 6 7 8 9

These symbols are called digits.

The decimal system, also called base 10 system, because it has 10 digits which is a naturally result of the fact that man has 10 fingers.

## B- Binary Number System

In the binary system or base 2, there can be only two values for each digit of a number, either a "0" or "1".

## C- Octal Number System

This system is composed of 8 numbers or symbols:

**0 1 2 3 4 5 6 7**

This is a base -8 system.

For counting after 7

10 ,11,12,13,14,15,16,17,

20,21,22,23,24,25,26,27,

30,31,…….37,

40,……..47,

50,…….57,

60,61,62,63,64,65,66,67,

70,71,72,73,74,75,76,77,

100,101,102,……107

110,111,112,……117.

# D- Hexa- Decimal System

This system is composed of 16 numbers or symbols (digit):

**0 1 2 3 4 5 6 7 8 9 A B C D E F**

Where “A” stands for 10, “B” for 11 and so on.

It is a base – 16 systems

**For counting after F**

10 ,11,12,13,14,15,16,17,18,19,1A,1B,1C,1D,1E,1F,

20,21,22,23,24,25,26,27,28,29,2A,2B,……..2F,

30,31,……..,3A,…….3F

.

.

90,91,92,……..99,9A,9B,……9F,

A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,AA,AB,,AC,AD,AE,AF,

B0,B1,…….B9,BA,…….BF,

# 2. Conversion among radices

## A.Convert from Decimal to Any Base

Let's express a **decimal number 1341 in binary notation**. Note that the desired base is 2, so we repeatedly divide the given decimal number by **2**.

```
          Quotient  Remainder
-----------------------------
1341/2  =   670         1 ----------------------+
 670/2  =   335         0 ---------------------+ |
 335/2  =   167         1 ------------------+ | |
 167/2  =    83         1 ----------------+ | | |
  83/2  =    41         1 --------------+ | | | |
  41/2  =    20         1 ------------+ | | | | |
  20/2  =    10         0 ----------+ | | | | | |
  10/2  =     5         0 --------+ | | | | | | |
   5/2  =     2         1 ------+ | | | | | | | |
   2/2  =     1         0 ----+ | | | | | | | | |
   1/2  =     0         1 --+ | | | | | | | | | |(Stop when the
                            | | | | | | | | | | quotient is 0)
                            1 0 1 0 0 1 1 1 1 0 1  (BIN; Base 2)
```

Let's express the same decimal number 1341 in octal notation.

```
          Quotient  Remainder
-----------------------------
1341/8  =   167         5 --------+
 167/8  =    20         7 ------+ |
  20/8  =     2         4 ----+ | |
   2/8  =     0         2 --+ | | |  (Stop when the quotient is 0)
                            | | | |
                            2 4 7 5  (OCT; Base 8)
```

Let's express the same decimal number 1341 in hexadecimal notation.

```
          Quotient  Remainder
-----------------------------
1341/16 =    83        13 ------+
  83/16 =     5         3 ----+ |
   5/16 =     0         5 --+ | |  (Stop when the quotient is 0)
                            | | |
                            5 3 D  (HEX; Base 16)
```

In conclusion, the easiest way to convert fixed point numbers to any base is to convert each part separately. We begin by separating the number into its integer and fractional part. The integer part is converted using the remainder method, by using a successive division of the number by the base until a zero is obtained. At each division, the reminder is kept and then the new number in the base r is obtained by reading the remainder from the last remainder upwards.

The conversion of the fractional part can be obtained by successively multiplying the fraction with the base. If we iterate this process on the remaining fraction, then we will obtain successive significant digit. These methods form the basis of the multiplication methods of converting fractions between bases

***Example.*** Convert the decimal number 3315 to hexadecimal notation. What about the hexadecimal equivalent of
the decimal number 3315.3

Solution:

```
             Quotient  Remainder
-----------------------------
3315/16 =   207        3 ------+
 207/16 =    12       15 ----+ |
  12/16 =     0       12 --+ | |  (Stop when the quotient is 0)
                           | | |
                           C F 3  (HEX; Base 16)
```

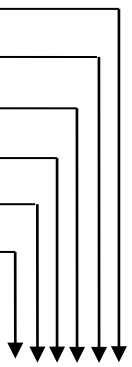
```
                                                    (HEX; Base 16)
                     Product  Integer Part   0.4 C C C ...
--------------------------------             | | | |
0.3*16           =   4.8        4          ----+ | | | |
0.8*16           =  12.8       12          ------+ | | |
0.8*16           =  12.8       12          --------+ | |
0.8*16           =  12.8       12          ----------+ |
                 :                          ----------------------+
                 :
     Thus, 3315.3 (DEC) -->
       CF3.4CCC... (HEX)
```

***Example:*** convert the following decimal numbers to the equivalent binary numbers (36 , 39.5).

| | | Q | R | |
|---|---|---|---|---|
| 36/2 | = | 18 | 0 | |
| 18/2 | = | 9 | 0 | |
| 9/2 | = | 4 | 1 | |
| 4/2 | = | 2 | 0 | |
| 2/2 | = | 1 | 0 | |
| 1/2 | = | 0 | 1 | |

36 (base 10) = 100100

| | Q | R |
|---|---|---|
| 39/2 = | 19 | 1 |
| 19/2 = | 9 | 1 |
| 9/2 = | 4 | 1 |
| 4/2 = | 2 | 0 |
| 2/2 = | 1 | 0 |
| 1/2 = | 0 | 1 |



39.5 (base 10) == 100111

10

0.5 x 2 = 1

0 x 2 = 0

The binary equivalent of $(39.5)_{10}$ is $(100111.10)_2$

***Example****:* convert the following decimal number to equivalent octal number $(266)_{10}$ & $(20.75)_{10}$

266/8 = 33 r =2

33/8 = 4 r =1

4/8 = 0 r =4

$(266)_{10}$ = 412

20/8 = 2 r =4 0.75 x 8 = 6.0

2/8 = 0 r =2

The equivalent octal number is $(24.6)_8$

# B. Convert From Any Base to Decimal

Let's think more carefully what a decimal number means. For example, 1234 means that there are four boxes (digits); and there are 4 one's in the right-most box (least significant digit), 3 ten's in the next box, 2 hundred's in the next box, and finally 1 thousand's in the left-most box (most significant digit). The total is 1234:

```
Original Number:       1      2      3     4
                       |      |      |     |
How Many Tokens:       1      2      3     4
Digit/Token Value:  1000    100     10     1
Value:              1000 + 200  + 30  + 4   = 1234
```

or simply, 1*1000 + 2*100 + 3*10 + 4*1 = 1234
Thus, each digit has a value: 10^0=1 for the least significant digit, increasing to 10^1=10, 10^2=100, 10^3=1000, and so forth.

Likewise, the least significant digit in a hexadecimal number has a value of 16^0=1 for the least significant digit, increasing to 16^1=16 for the next digit, 16^2=256 for the next, 16^3=4096

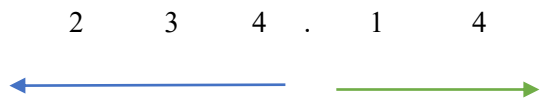
***Example***. Convert 11001. 0101 expressed in a Binary notation to decimal.

| 1 | 1 | 0 | 0 | 1 | . | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 1 | 0 | | -1 | -2 | -3 | -4 |
| 2^4 | 2^3 | 2^2 | 2^1 | 2^0 | | 2^-1 | 2^-2 | 2^-3 | 2^-4 |

$= 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 + 0*2^{-1} + 1*2^{-2} + 0*2^{-3} + 1*2^{-4}$
$= 1*16 + 1*8 + 0*4 + 0*2 + 1*1 + 0*\frac{1}{2} + 1*\frac{1}{4} + 0*1/8 + 1*1/16$

$= 16 + 8 + 1 + 1/4 + 1/16$
$= 25 + 0.25 + 0.0625$
$= 25.312$

***Example.*** Convert 234.14 expressed in an octal notation to decimal.




$$
\begin{array}{lccccc}
 & 8^{\wedge}2 & 8^{\wedge}1 & 8^{\wedge}0 & 8^{\wedge}-1 & 8^{\wedge}-2 \\
= & 2*8^{2} & + 3*8^{1} & + 4*8^{0} & +1*8^{-1} & + 4*8^{-2} \\
= & 2*64 & + 3*8 & + 4*1 & +1/8 & +4/64
\end{array}
$$

$$= 128 + 24 + 4 + 0.125 + 0.062 = 156.1875$$

***Examples:***

**1) Decimal :-**

$(124)_{10} = 4 \times 10^{0} + 2 \times 10^{1} + 1 \times 10^{2}$

$(252.512)_{10} = 2 \times 10^{0} + 5 \times 10^{1} + 2 \times 10^{2} + 5 \times 10^{-1} + 1 \times 10^{-2} + 2 \times 10^{-3}$

**2)Binary :-**

$(1011101)_{2} = 1 \times 2^{0} + 0 \times 2^{1} + 1 \times 2^{2} + 1 \times 2^{3} + 1 \times 2^{4} + 0 \times 2^{5} + 1 \times 2^{6}$

$= (93)_{10}$

$(101.11)_{2} = 1 \times 2^{0} + 0 \times 2^{1} + 1 \times 2^{2} + 1 \times 2^{-1} + 1 \times 2^{-2}$

$= (5.75)_{10}$

**3) Octal :-**

$(537)_8 = 7 \times 8^0 + 3 \times 8^1 + 5 \times 8^2$

$=(351)_{10}$

**4) Hexa- Decimal :-**

$(A01B)_{16} = 11 \times 16^0 + 1 \times 16^1 + 0 \times 16^2 + 10 \times 16^3$

$=(40987)_{10}$

## C.Relationship between Binary - Octal and Binary-hexadecimal

There is a direct correspondence between the binary system and the octal system, with three binary digits corresponding to one octal digit. Likewise, four binary digits translate directly into one hexadecimal digit.

## Octal-to-Binary:-

The conversion from octal to binary is performed by converting each octal digit to its 3-bit binary equivalent. The eight possible digits are converted as indicated in the following table:

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

***Example:*** convert the following octal number to it's equivalent binary number $(472)_8$

| 4 | 7 | 2 |
|---|---|---|
| 100 | 111 | 010 |

The equivalent binary number is $(100111010)_2$

## Binary-to-octal :-

1. group into 3's starting at least significant symbol **(if the number of bits is not evenly divisible by 3, then add 0's at the most significant end)**
2. write 1 octal digit for each group

***Example:***

| 100 | 010 | 111 | (binary) |
|---|---|---|---|
| 4 | 2 | 7 | (octal) |

| 10 | 101 | 110 | (binary) |
|---|---|---|---|
| 2 | 5 | 6 | (octal) |

***Example:-***

convert $(177)_{10}$ to its 8-bit binary equivalent by first converting to octal.

Solution:-

177/8 = 22 + reminder of 1
22/8 = 2 + reminder of 6
2/8 = 0 + reminder of 2

$(177)_{10} = (261)_8$

| 2 | 6 | 1 |
|---|---|---|
| 010 | 110 | 001 |

$(177)_{10} = (261)_8 = (010110001)_2$

**EXAMPLE 1**: convert (1010111100) binary to octal

| Binary | 001 | 010 | 111 | 100 |
|---|---|---|---|---|
| Octal | 1 | 2 | 7 | 4 |

The Octal number is (1274)

**EXAMPLE 2**: convert (101111010110) binary to octal

| Binary | 101 | 111 | 010 | 110 |
|---|---|---|---|---|
| Octal | 5 | 7 | 2 | 6 |

## Hexadecimal -to-Binary :-

Each hexadecimal digit is converted to it's 4-bit binary equivalent as show in the table below :

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

***Example:***

3 9 C 8
0011 1001 1100 1000

## Binary-to-hexadecimal :-

This conversion is just the reverse of the Hexadecimal-to-Binary conversion process.

***Example:***

1001 1110 0111 0000
9 E 7 0

1 1111 1010 0011
1 F A 3

**EXAMPLE 1**: convert (01011110101101010010) binary to hexadecimal

| Binary | 0101 | 1110 | 1011 | 0101 | 0010 |
|---|---|---|---|---|---|
| hexa | 5 | E | B | 5 | 2 |

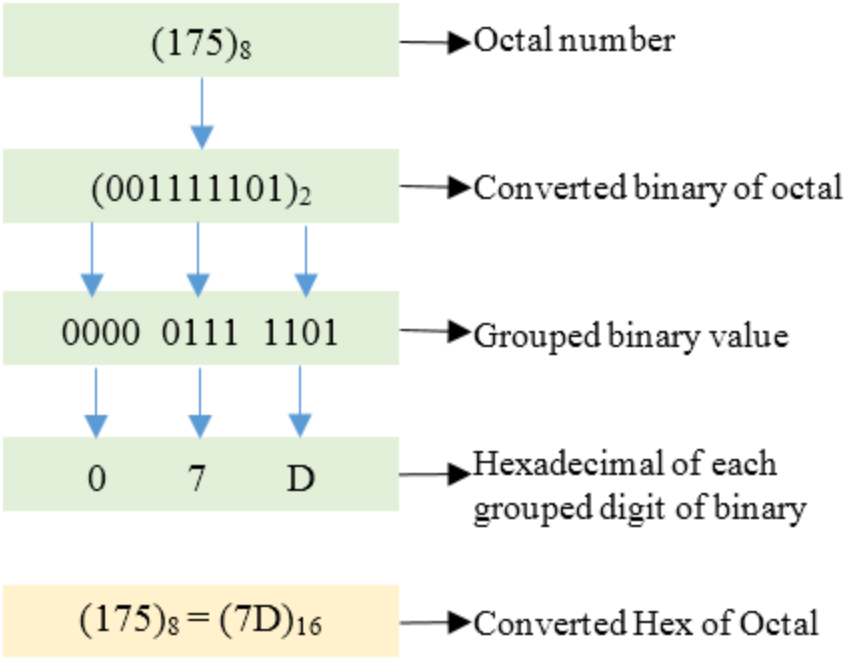
The conversion methods can be used to convert a number from any base to any other base, but it may not be very intuitive to convert something like 513.03 to base 7. As an aid in performing an unnatural conversion, we can convert to the more familiar base 10 form as an intermediate step, and then continue the conversion from base 10 to the target base.

***Example :*** Convert the Octal number (752) to Hexadecimal number .

*Step1 :*
Octal to Binary Conversion
7 5 2
(111 101 010)
So the binary equivalent 111101010

*Step2 :*
Binary to Hexadecimal Conversion
0001 1110 1010
1 F A

***Example :***


$(175)_8$
Octal number
$(001111101)_2$
Converted binary of octal
0000 0111 1101
Grouped binary value
0 7 D
Hexadecimal of each grouped digit of binary
$(175)_8 = (7D)_{16}$
Converted Hex of Octal

# Binary Codes

Binary codes are one of the important concepts in digital electronics. A binary code is a type of digital code consisting of two digits, 0 and 1. Binary codes act as the primary language in any digital computing system. Binary codes can represent different types of information such as numbers, letters, images, videos, etc.

All digital system can understand and manipulate information expressed in binary language only. In the case of binary codes, each digit is called a **binary digit** or **bit**.

Binary codes represents information using 0 and 1. In a digital system, the binary codes are organized into segments like **bits** or **bytes**. A bit is either a binary 0 or 1. When 8 bits are grouped together, then it is called a byte. Each byte represents a piece of information in a digital system.

## Types of Binary Codes

Binary codes can be classified into the following major types

- Weighted Binary Codes
- Non-weighted Binary Codes
- Alphanumeric Code
- Binary Coded Decimal (BCD)
- Error Detecting Code
- Error Correction Code

## Weighted Binary Codes

Weighted binary codes are a type of binary code in which each bit position has a specific weight associated with its positional value. For example, let a 4-bit weighted binary code 1011. The value of the code is,

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$
$$1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$$
$$8 + 0 + 2 + 1 = 11$$

It is clear that the rightmost bit has a positional weight of $2^0 = 1$, whereas the leftmost bit has a positional weight of $2^3 = 8$.

**Examples of weighted binary codes** are 8421 BCD code, 5211 code, 2421 code, etc.

## Binary-Coded-Decimal (BCD) code :-

In computing and electronic systems, binary-coded decimal (BCD) is an encoding for decimal numbers in which each digit is represented by its own binary sequence. Its main virtue is that it allows easy conversion to decimal digits for printing or display and faster decimal calculations. Its drawbacks are the increased complexity of circuits needed to implement mathematical operations and a relatively inefficient encoding. It occupies more space than a pure binary representation.In BCD, a digit is usually represented by four bits which, in general, represent the values/digits/characters 0-9

To BCD-encode a decimal number using the common encoding, each decimal digit is stored in a four-bit **nibble.**

| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| BCD | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 |

The position weights of the BCD code are 8, 4, 2, 1.

***Example**:* The BCD of decimal 874

| 8 | 7 | 4 |
|---|---|---|
| 1000 | 0111 | 0100 |

***Example**:* the BCD encoding for the number 127 would be:

0001 0010 0111

It is very important to understand the difference between the conversion of a decimal number to binary and the binary coding of a decimal number. In each case, the final result is a series of bits. The bits obtained from conversion are binary digit. Bits obtained from coding are combinations of 1's and 0's arranged according to the rule of the code used.
e.g. the binary conversion of 13 is 1101; the BCD coding of 13 is 00010011
The main advantage of binary coded decimal is that it allows easy conversion between decimal (base-10) and binary (base-2) form. However, the disadvantage is that BCD code is wasteful as the states between 1010 (decimal 10), and 1111 (decimal 15) are not used. Nevertheless, binary coded decimal has many important applications especially using digital displays.

## ***Advantages of BCD Codes***

- BCD codes are very similar to the decimal system.
- We need to remember the binary equivalent of the decimal numbers 0 to 9 only.

## ***Disadvantages of BCD Codes***

- The addition and subtraction of BCD codes follow different rules.
- BCD arithmetic is a little more complicated.

- BCD needs more number of bits than binary to represent the decimal number. e.g. the binary conversion of 13 is 1101; the BCD coding of 13 is 00010011. So, BCD is less efficient than binary.

## Non-Weighted Binary Codes

In digital electronics, the type of digital or binary codes in which each bit position does not have a specific weight associated with it is known as a non-weighted binary code.

In non-weighted binary codes, the value of the bit does not depend on the position within the number. Each bit position has an equal positional value.

Examples of non-weighted binary codes include **Excess-3** code and **Gray code**.

## Excess-3-code :-

It is performed in the same manner as BCD except that 3 is added to each decimal digit before encoding it in binary. Thus, the code of decimal 0 is 0011, that of 6 is 1001, etc. The following table shows this code.

| **Decimal** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Ex-3code** | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 |

## Gray code :-

Gray codes are a type of non-weighted code. They are not arithmetic codes, which means there are no specific weights assigned to the bit position.

Gray codes have a very special feature that, only one bit will change each time the decimal number is incremented (see the figure below). As only one bit changes at a time, gray codes are also known unit distance code.

The following table shows this code.

| **Decimal** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Gray-code** | 0000 | 0001 | 0011 | 0010 | 0110 | 0111 | 0101 | 0100 | 1100 | 1101 |

# Operations in Binary

## 1. Addition in Binary

Now that we know binary numbers, we will learn how to add them. Binary addition is much like your normal everyday addition (decimal addition), except that it carries on a value of 2 instead of a value of 10.

For example: in decimal addition, if you add 8 + 2 you get ten, which you write as 10; in the sum this gives a digit 0 and a carry of 1. Something similar happens in binary addition when you add 1 and 1; the result is two (as always), but since two is written as 10 in binary, we get, after summing 1 + 1 in binary, a digit 0 and a carry of 1. Therefore in binary:

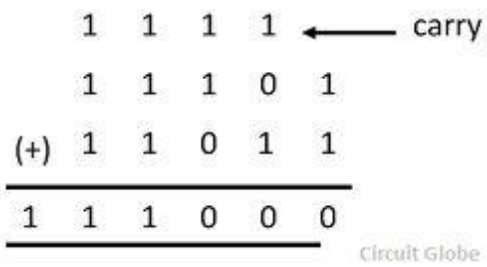
| No. of state | A | + | B | Carry | Sum |
|---|---|---|---|---|---|
| 0 | 0 | + | 0 | 0 | 0 |
| 1 | 0 | + | 1 | 0 | 1 |
| 2 | 1 | + | 0 | 0 | 1 |
| 3 | 1 | + | 1 | 1 | 0 |

Problem: 100101 + 10101 = ?.
Answer: 100101 + 10101 = 111010.

Explanation:

```
                    1       1
      1     0       0     1     0     1
+           1       0     1     0     1
-------------------------------------------------------------------
    1     1     1     0     1     0
```

***Example*** :




***Example :***

00011010 + 00001100 = 00100110

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 1 | | | | | *carries* |
| | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | = $26_{(\text{base } 10)}$ |
| + | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | = $12_{(\text{base } 10)}$ |
| | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | = $38_{(\text{base } 10)}$ |

00010011 + 00111110 = 01010001

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 1 | 1 | 1 | 1 | | | *carries* |
| | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | = $19_{(\text{base } 10)}$ |
| + | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | = $62_{(\text{base } 10)}$ |
| | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | = $81_{(\text{base } 10)}$ |

***Example :***

1011.01+11.011= 1110.101

```
   11  1
  1011.01
+   11.011
-----------
  1110.101
```

## 2. Subtraction in Binary

Subtraction and Borrow, these two words will be used very frequently for the binary subtraction. There are four rules of binary subtraction.

| No. of state | A - B | Borrow | Subtract |
|---|---|---|---|
| 0 | 0 - 0 | 0 | 0 |
| 1 | 0 - 1 | 1 | 0 |
| 2 | 1 - 0 | 0 | 1 |
| 3 | 1 - 1 | 0 | 0 |

***Example :***

0011010-001100=00001110

```
          0 10
            0 10                    Borrow
  0  0  1  1  0  1  0               =26
  0  0  0  1  1  0  0               =12

-----------------------------------------
  0  0  0  1  1  1  0               = 14
```

***Example :***

```
          0  10                     Borrow
    1  1  0   0                     =12
-   1  0  1   0                     =10

-----------------------------------------
       0  0  1  0                   =2
```

***Example:***

```
                1  10  10
             0  10 0   0  10          borrow
    1  0  1  0  1  .  1  0  1
-      1  0  1  1  .  1  1
-----------------------------------
       1  0  0  1  .  1  1  1
```

```
  0  0 1 0  0
  1  1 1 1  1
  10101.101
- 1011.11
-----------
  1001.111
```

## 3. Multiplications in Binary :

Binary multiplication is similar to decimal multiplication. It is simpler than decimal multiplication because only 0s and 1s are involved. There are four rules of binary multiplication.

| No. of state | A × B | Multiplication |
|---|---|---|
| 0 | 0 × 0 | 0 |
| 1 | 0 × 1 | 0 |
| 2 | 1 × 0 | 0 |
| 3 | 1 × 1 | 1 |

Example:

0011010 x 001100 = 100111000

$$
\begin{array}{rl}
 & 0011010 = 26_{10} \\
\times & 0001100 = 12_{10} \\
\hline
 & 0000000 \\
 & 0000000 \\
 & 0011010 \\
 & 0011010 \\
\hline
 & 0100111000 = 312_{10}
\end{array}
$$

## 4. Division in Binary

The binary division operation is similar to the base 10 decimal system, except the base 2. The division is probably one of the most challenging operations of the basic arithmetic operations. There are different ways to solve division problems using binary operations. Long division is one of them and the easiest and the most efficient way.

### Binary Division Rules

The binary division is much easier than the decimal division when you remember the following division rules. The main rules of the binary division include:

- 1÷1 = 1
- 1÷0 = Meaningless
- 0÷1 = 0
- 0÷0 = Meaningless

Similar to the decimal number system, the binary division is similar, which follows the four-step process:

- Divide
- Multiply
- Subtract
- Bring down

**Important Note:** Binary division follows the long division method to find the resultant in an easy way.

**Comparison with Decimal Value**

$(01111100)_2 = (1111100)_2 = 124_{10}$

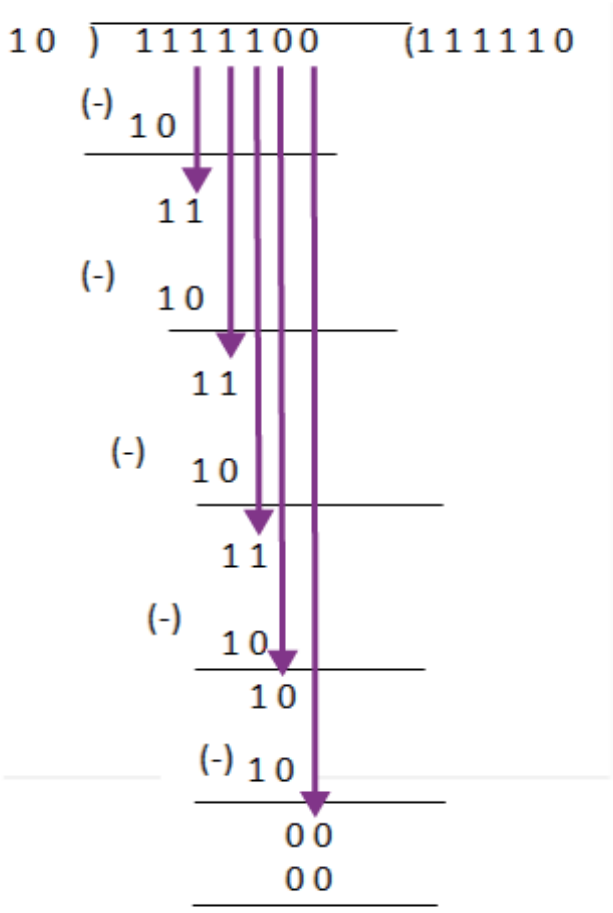
$(0010)_2 = (10)_2 = 2_{10}$
You will get the resultant value as 62 when you divide 124 by 2.
So the binary equivalent of 62 is $(111110)_2$
$(111110)_2 = 62_{10}$
Both the binary and the decimal system produce the same result.
**Binary Division Examples**

***Example 1.***
**Question:** Solve 01111100 ÷ 0010
**Solution:**
Given
01111100 ÷ 0010
Here the dividend is 01111100, and the divisor is 0010

Remove the zero's in the **Most Significant Bit** in both the dividend and divisor, that doesn't change the value of the number.

So the dividend becomes 1111100, and the divisor becomes 10.

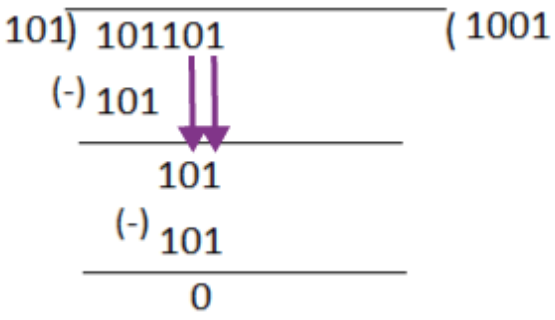
Now, use the long division method.




So, 01111100 ÷ 0010 = 111110

***Example 2:*** Solve using the long division method: 101101 ÷ 101

**Solution:**



# Representation methods

## 1.Signed Magnitude Representation

The signed magnitude (also referred to as sign and magnitude) representation is most familiar to us as the base 10 number system. A plus or minus sign to the left of a number indicates whether the number is positive or negative as in +12 or -12. In the binary signed magnitude representation, the leftmost bit is used for the sign, which takes on a value of 0 or 1 for '+' or '-', respectively. The remaining bits contain the absolute magnitude.

Consider representing (+12) and (-12) in an eight-bit format:

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

B7 : for the sign of number

If the number is (+) => B7=0

If the number is (-) => B7=1

B0-B6: Is for the magnitude

$(+12)_{10} = (00001100)_2$

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

$(-12)_{10} = (10001100)_2$

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

The negative number is formed by simply changing the sign bit in the positive number from 0 to 1. Notice that there are both positive and negative representations for zero: +0= 00000000 and -0= 10000000.

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 2.One's Complement Representation

The one's complement operation is trivial to perform: convert all of the 1's in the number to 0's, and all of the 0's to 1's. We can observe that in the one's complement representation the leftmost bit is 0 for positive numbers and 1 for negative numbers, as it is for the signed magnitude representation. This negation, changing 1's to 0's and changing 0's to 1's, is known as complementing the bits. Consider again representing (+12)10 and (-12)10 in an eight-bit format, now using the one's complement representation:

(+12) in decimal = (00001100) in Binary

(-12) in decimal = (11110011) in Binary

**Note again that there are representations for both +0 and -0, which are 00000000 and 11111111, respectively**. As a result, there are only 28 - 1 = 255 different numbers that can be represented even though there are 28 different bit patterns.

The one's complement representation is not commonly used. This is at least partly due to the difficulty in making comparisons when there are two representations for 0. There is also additional complexity involved in adding numbers.

## 3. Two's Complement Representation

The two's complement is formed in a way similar to forming the one's complement: complement all of the bits in the number, but then add 1, and if that addition results in a carry-out from the most significant bit of the number, discard the carry-out.

Examination of the fifth column of Table above shows that in the two's complement representation, the leftmost bit is again 0 for positive numbers and is 1 for negative numbers. However, this number format does not have the unfortunate characteristic of signed-magnitude and one's complement representations: it has only one representation for zero. To see that this is true, consider forming the negative of (+0)10, which has the bit pattern: (+0)10 = (00000000)2

Forming the one's complement of (00000000)2 produces (11111111)2 and adding

1 to it yields (00000000)2, thus (-0)10 = (00000000)2. The carry out of the leftmost position is discarded in two's complement addition (except when detecting an overflow condition). Since there is only one representation for 0, and since all bit patterns are

valid, there are 2^8 = 256 different numbers that can be represented.

Starting with $(+12)_{10} = (00001100)_2$,

1's complement, or negate the number, producing $(11110011)_2$

Now add one

producing $(11110100)_2$, and

thus $(-12)_{10} = (11110100)_2$:

$$(+12)_{10} = (00001100)_2$$

$$(-12)_{10} = (11110100)_2$$

There is an equal number of positive and negative numbers provided zero is considered to be a positive number, which is reasonable because its sign bit is 0. The positive numbers start at 0, but the negative numbers start at -1, and so the magnitude of the most negative number is one greater than the magnitude of the most positive number. The positive number with the largest magnitude is +127, and the negative number with the largest magnitude is -128. There is thus no positive number that can be represented that corresponds to the negative of -128. If we try to form the two's

complement negative of -128, then we will arrive at a negative number, as shown below:

$(-128)_{10} = (10000000)_2$
$(-128)_{10} = (01111111$
$(-128)_{10} + (+0000001)2$
$(-128)_{10}$ ——————)2
$(-128)10 = (10000000)_2$

| | | |
|---|---|---|
| (0) | = | (0 0 0 0 0 0 0 0 ) |
| 1's complement | = | ( 1 1 1 1 1 1 1 1) |
| Add 1 | = | +( 0 0 0 0 0 0 0 1) |
| The 2's complement | = | (0 0 0 0 0 0 0 0 ) |

The two's complement representation is the representation most commonly used in conventional computers.

# Subtraction using Complements

## -Binary Subtraction Using 2's Complement

**What is a 2 's Complement?**

To implement this method for subtracting two binary numbers, the first step is to find the 2's complement of the number to be subtracted from another number. To get the 2's complement, first of all, 1's complement is found, and then 1 is added. The addition is the required 2's complement.

Suppose we need to find the 2's complement of the binary number 10010. First, find 1's complement. To find this, replace all 1 to 0 and all 0 to 1. Therefore, 1's complement of 10010 will be 01101. Add 1 to this, and we will get the 2's complement, i.e. 01110.

To learn how to subtract binary numbers using 2's complement, which is the subtraction of a smaller number from a larger number using 2's complement subtraction, the following steps are to be followed:

- **Step 1:** Determine the 2's complement of the smaller number
- **Step 2:** Add this to the larger number.
- **Step 3:** Omit the carry. Note that there is always a carry in this case.

The following example illustrates the above-mentioned steps:

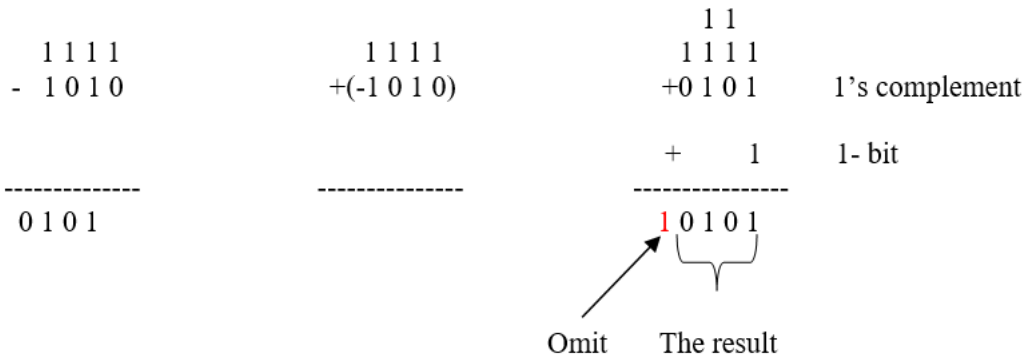
***Exampe:*** Subtract (1010)2 from (1111)2 using 2's complement method.

Ans:

**Step 1**: 2's complement of (1010)2 is (0110)2.

**Step 2**: Add (0110)2 to (1111)2.

This is shown below:




## Solved Examples

**Q 1. 10110 - 11010**

**Ans:** 11010 has a 2s complement of (00101+1) or 00110.
Add the 2's complement to the minuend (10110+00110) or 11100.
Now taking its complement;
The solution is (00011+1)= - (00100)

**Q 2. 10110-01111**

**Ans:** 01111's 2s complement is 10001.
The minuend plus the complement of two (10110-10001) equals 100111.
The response is 00111.

**Q 3. 0100-11101**

**Ans:** 11101's 2s complement is 00011

The minuend plus the complement of two (10100- 00011) equals 10111.
Since there is no carry here, the response is 01001.

**Q 4. 110101 - 101001**

**Ans:** 101001's complement in 2 is 010111
(110101-010111) Add the minuend and the 2's complement to get 1001100.
Carry, the result's leftmost bit is a 1 and is ignored.
The response is 001100.

## 4/ Post test :-

1. Suppose you are in base 7 Numeric System, write the basic digit of this system and suggest a code for it .
2. convert $(423)_{10}$ to hexadecimal.

## key answer

**posttest:-**

**1. 0 1 2 3 4 5**

**2. 1A7**

# 5/ HomeWorks: -

1- Convert $(641)_8$ to decimal (Ans. 369).
2- Convert $(146)_{10}$ to octal then from octal to binary (Ans. 222 and 010010010).
3- Convert $(10011101)_2$ to octal (Ans. 235).
4- Write the next three numbers in this octal counting sequence: 624, 625, 626, ….., ….., ……
5- Convert $(975)_{10}$ to binary by first converting to octal (Ans. 1111001111).
6- Convert binary 1010111011 to decimal by first converting to octal (Ans. 699).
7-Convert $(24CE)_{16}$ to decimal (Ans. 9422).
8-Convert $(3117)_{10}$ to hex, then from hex to binary (Ans. C2D and 110000101101)
9-Convert $(1001011110110101)_2$ to hex (Ans 97B5).
10-Write the next four numbers in this hex counting sequence: E9A, E9B, E9C, E9D, ….., ……., ……, …….
11-Convert $(3527)_8$ to hex (Ans. $(757)_{16}$).
12- **1001 – 0100** .**Ans:** 0101
13- 0100 – 1011 . **Ans:** 1011
14- 0110 – 0100 . **Ans:** 0010
15-10110- 11101 . **Ans:** 00111
16- 110-101 .**Ans:** 001
17. By using Signed Magnitude Representation method, represent (+81) and (-81

**Ministry of high Education and Scientific Research**
**Southern Technical University**
**Technological institute of Basra**
**Department of Computer Networks and Software Techniques**

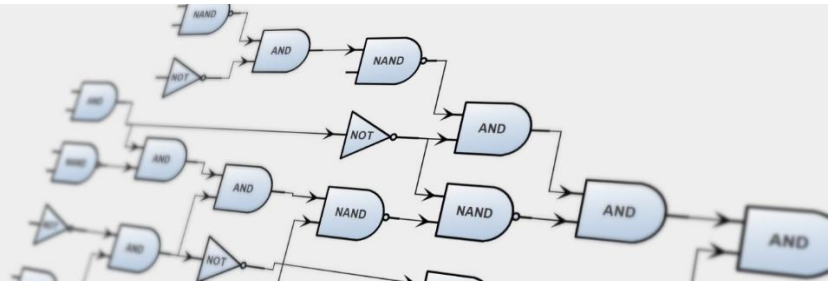
# Learning package

**In**

# Logic Gates

**For**

Students of First Year




**By**
**Ethar Abduljabbar Hadi**
**Assistant Lecturer**
**Dep. Of Computer Networks and Software Techniques**
**2025**

# 1/ Overview

## 1 / A –Target population :-

For First year students
Technological institute of Basra
Dep. Of Computer Networks and Software Techniques

## 1 / B –Rationale :-

The objective of studying logic gates is to understand the fundamental building blocks of digital systems. Logic gates are the foundation of digital electronics.

## 1 / C –Central Idea :-

1- Understand Binary Logic.

2- Understand Logic Gates(AND, OR, NOT, NAND, NOR, XOR, and XNOR).

3- Different inputs Logic Circuit.

4- Interpret and Create Truth Tables.

5- Translate logical expressions into truth tables and vice versa

## 1 / D – Performance Objectives

After studying this unit, the student will be able to:-

1- Know the types of logic gates

2- Build simple and complex logic circuits

3- Represent logic circuits in Boolean algebra .

## 2/ Pretest :

**What is the relation between the logic gates and electronics circuits?**

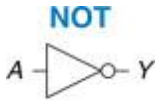
# 3/ Logic gates :-

A 'Logic Gate' is a type of simple digital circuit that takes binary inputs and produces binary output. It is used in digital systems to perform operations on binary variables.

Logic gates are simple digital circuits that take one or more binary inputs and produce a binary output. Logic gates are drawn with a symbol showing the input (or inputs) and the output. Inputs are usually drawn on the left (or top) and outputs on the right (or bottom). Digital designers typically use letters near the beginning of the alphabet for gate inputs and the letter Y for the gate output. The relationship between the inputs and the output can be described with a truth table or a Boolean equation. A truth table lists inputs on the left and the corresponding output on the right. It has one row for each possible combination of input. A Boolean equation is a mathematical expression using binary variables.

## 1.NOT Gate

A NOT gate has one input, A, and one output, Y, as shown in Figure bellow. The NOT gate's output is the inverse of its input. If A is FALSE, then Y is TRUE. If A is TRUE, then Y is FALSE. This relationship is summarized by the truth table and Boolean equation in the figure. The line over A in the Boolean equation is pronounced NOT, so $Y=\overline{A}$ is read "Y equals NOT A." The NOT gate is also called an inverter.




$Y=\overline{A}$

| A | Y |
|---|---|
| 0 | 1 |
| 1 | 0 |

## 2. AND Gate

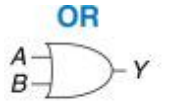
Two-input logic gates are more interesting. The AND gate shown in Figure below produces a TRUE output, Y**, if and only if both A and B are TRUE. Otherwise, the output is FALSE.** The Boolean equation for an AND gate can be written in several ways: Y = A • B, Y = AB, read "Y equals A and B.


$Y=AB$

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## 3. OR Gate

The OR gate shown in Figure below produces a TRUE output, Y, **if either A or B (or both) are TRUE**. The Boolean equation for an OR gate is written as Y = A + B . Digital designers normally use the + notation, Y = A + B is pronounced "Y equals A or B."

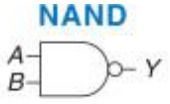



$Y = A + B$

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

## 4. NAND Gate

Any gate can be followed by a bubble to invert its operation. The NAND gate performs **NOT AND**. Its output is TRUE unless both inputs are TRUE as shown in figure bellow:

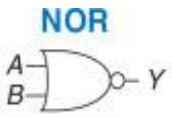



$Y = \overline{AB}$

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## 5.NOR Gate

The NOR gate performs **NOT OR**. Its output is TRUE if neither A nor B is TRUE as shown in figure below:




$Y = \overline{A + B}$

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

## 6.EX-OR Gate

XOR (exclusive OR, pronounced "ex-OR") is TRUE if A or B, but not both, are TRUE. The XOR operation is indicated by $\oplus$, a plus sign with a circle around it.


$Y = A \oplus B$

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

An N-input XOR gate is sometimes called a parity gate and produces a TRUE output if an **odd number of inputs are TRUE**.

As with two-input gates, the input combinations in the truth table are listed in counting order

## 7.XNOR Gate

Figure below shows the symbol and Boolean equation for a two-input XNOR (pronounced ex-NOR) gate that performs the inverse of an XOR.

The XNOR output is TRUE if both inputs are FALSE or both inputs are TRUE. The two-input XNOR gate is sometimes called an equality gate because its output is TRUE when the inputs are equal.


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

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Summery :**

- AND gate: the output is 1 if all inputs are 1; otherwise, the output is 0.
- OR gate: the output is 1 if at least one input is 1; otherwise, the output is 0.
- XOR gate: the output is 0 if both inputs are same; otherwise, the output is 1.
- NAND gate: the output is 1 if at lease one input is 0;

otherwise, the output is 0.

- NOR gate: the output is 1 if both inputs are 0; otherwise, the output is 0.
- NOT gate or inverter: the output is 1 if the input is 0 and the output is 0 if the input is 1.

## Logic gate symbols

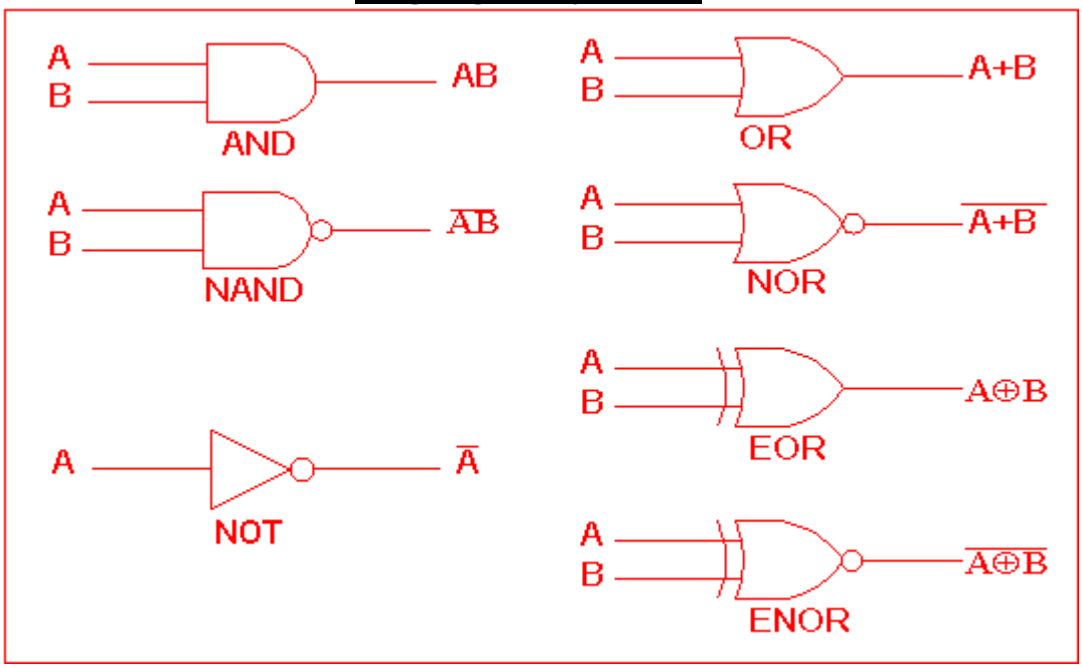



Table 2 is a summary truth table of the input/output combinations for the NOT gate together with all possible input/output combinations for the other gate functions. Also note that a truth table with 'n' inputs has 2n rows. You can compare the outputs of different gates.

## Logic gates representation using the Truth table

| NOT gate | |
|---|---|
| A | $\overline{A}$ |
| 0 | 1 |
| 1 | 0 |

| INPUTS | | OUTPUTS | | | | | |
|---|---|---|---|---|---|---|---|
| A | B | AND | NAND | OR | NOR | EXOR | EXNOR |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |

# Multiple-Input Gates

Many Boolean functions of three or more inputs exist. The most common are AND, OR, XOR, NAND, NOR, and XNOR. An N-input AND gate produces a TRUE output when all N inputs are TRUE. An N-input OR gate produces a TRUE output when at least one input is TRUE.

## Truth tables :-

Many logic circuits have more than one input and one or more outputs. A truth table shows how the logic circuit's output responds to the various combinations of logic states at the inputs. The formal for two, three, and four input with one output truth tables are shown below :

| No.of State in Decimal | Inputs | | Output |
|---|---|---|---|
| | A | B | F |
| 0 | 0 | 0 | |
| 1 | 0 | 1 | |
| 2 | 1 | 0 | |
| 3 | 1 | 1 | |

If the number of inputs more than 2 ,

The number of states in the truth table = 2 ^ number of inputs

For example If the number of inputs = 3 ,

The number of states in the truth table = 2 ^ 3 = 2*2*2= 8 and the truth table as shown below :

| No. of states in decimal | Inputs | | | Output |
|---|---|---|---|---|
| | A | B | C | X |
| 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 1 | |
| 2 | 0 | 1 | 0 | |
| 3 | 0 | 1 | 1 | |
| 4 | 1 | 0 | 0 | |
| 5 | 1 | 0 | 1 | |
| 6 | 1 | 1 | 0 | |
| 7 | 1 | 1 | 1 | |

For example If the number of inputs = 4 ,

The number of states in the truth table = 2 ^ 4 = 2*2*2*2= 16 and the truth table as shown below :

| No. of states in decimal | Inputs | | | | Output |
|---|---|---|---|---|---|
| | A | B | C | D | X |
| 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0 | 0 | 1 | |
| 2 | 0 | 0 | 1 | 0 | |
| 3 | 0 | 0 | 1 | 1 | |
| 4 | 0 | 1 | 0 | 0 | |
| 5 | 0 | 1 | 0 | 1 | |
| 6 | 0 | 1 | 1 | 0 | |
| 7 | 0 | 1 | 1 | 1 | |
| 8 | 1 | 0 | 0 | 0 | |
| 9 | 1 | 0 | 0 | 1 | |
| 10 | 1 | 0 | 1 | 0 | |
| 11 | 1 | 0 | 1 | 1 | |
| 12 | 1 | 1 | 0 | 0 | |
| 13 | 1 | 1 | 0 | 1 | |
| 14 | 1 | 1 | 1 | 0 | |
| 15 | 1 | 1 | 1 | 1 | |

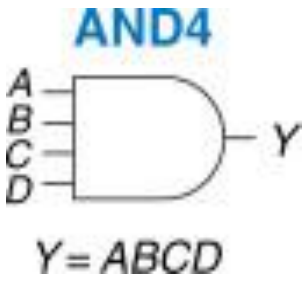
**Example**

***Example :*** Four-Input And Gate The figure below shows the symbol and Boolean equation for a four-input AND gate. Create a truth table.

Solution

Figure bellow shows the truth table. The output is TRUE only if all of the inputs are TRUE.

| A | B | C | D | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |



***Example***

Three-Input NOR Gate


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

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | |

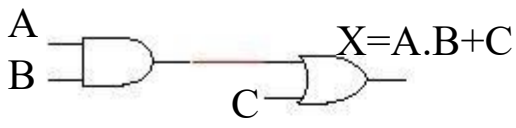
| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

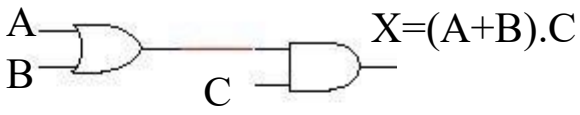
## Describing logic circuits algebraically

Any logic circuit, no matter how complex, may be completely described using the Boolean operations previously defined.

***Example:-***

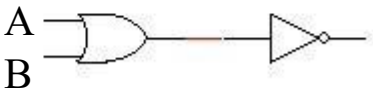
Determine the output expression for the logic indicated below: -





$X = \bar{A} + B$




$X = \overline{A + B}$

## Implementing circuits from Boolean expressions :-

If the operation of a circuit is defined by a Boolean expression, a logic circuit can be implemented directly from that expression.

*Example :*

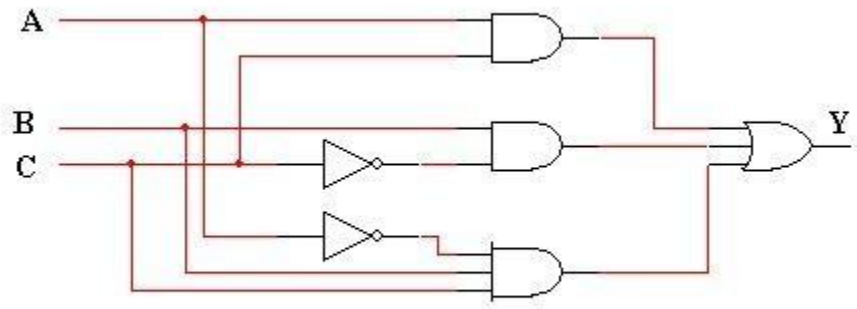
Implement the logic circuits defined by the following Boolean expressions :

a) $Y = AC + B\bar{C} + \bar{A}BC$

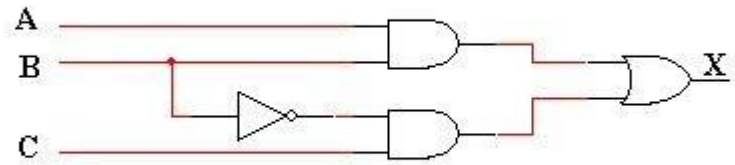
b) $X = AB + \bar{B}C$

Solution :-

a)




b)



# Canonical and Standard Form

Canonical Form – In Boolean algebra, the Boolean function can be expressed as Canonical Disjunctive Normal Form known as minterm and some are expressed as Canonical Conjunctive Normal Form known as maxterm.

- In Minterm, we look for the functions where the output results in "1"
- In Maxterm we look for functions where the output results in "0".
- We perform the Sum of minterm also known as the Sum of products(SOP).
- We perform Product of Maxterm also known as Product of sum(POS).

Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form.

A Boolean function can be expressed algebraically from a given truth table by forming a :

- Minterm for each combination of the variables that produces a 1 in the function and then takes the OR of all those terms.
- Maxterm for each combination of the variables that produces a 0 in the function and then takes the AND of all those terms.

## 1.POS Truth Table

**The Sum-of-Products (SOP) Form (Minterm) This form is sometimes called "minterm". A product term that contains each of the n-variables factors in either complemented or uncomplemented** form for output digits "1" only, is called SOP.

Consider a function X, whose truth table is as follows:

| No. of states in decimal | Inputs | | | Output | (Minterm) |
|---|---|---|---|---|---|
| | A | B | C | X | |
| 0 | 0 | 0 | 0 | 1 | $\bar{A}\bar{B}\bar{C}$ |
| 1 | 0 | 0 | 1 | 0 | |
| 2 | 0 | 1 | 0 | 1 | $\bar{A}B\bar{C}$ |
| 3 | 0 | 1 | 1 | 1 | $\bar{A}BC$ |
| 4 | 1 | 0 | 0 | 0 | |
| 5 | 1 | 0 | 1 | 0 | |
| 6 | 1 | 1 | 0 | 1 | $A\bar{B}\bar{C}$ |
| 7 | 1 | 1 | 1 | 1 | ABC |

The Logical SOP expression for the output digit "1" is written as"

$$F = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC$$

This function com be put in another form such as:

$$F = \sum 0, 2, 3, 6, 7$$

Since F= 1 in rows 0, 2,3,6,7 only.

The second form is called the Canonical Sum of Products (Canonical SOP).

## POS Truth Table

A Logical equation can also be expressed as a product of sum (POS) form (sometimes this method is called "Maxterm". This is done by considering the combination for F=0 (output = 0).

Consider a function X, whose truth table is as follows:

| No. of states in decimal | Inputs | | | Output | (Maxterm) |
|---|---|---|---|---|---|
| | A | B | C | X | |
| 0 | 0 | 0 | 0 | 1 | |
| 1 | 0 | 0 | 1 | 0 | $(A + B + \bar{C})$ |
| 2 | 0 | 1 | 0 | 1 | |
| 3 | 0 | 1 | 1 | 1 | |
| 4 | 1 | 0 | 0 | 0 | $(\bar{A} + B + C)$ |
| 5 | 1 | 0 | 1 | 0 | $(\bar{A} + B + \bar{C})$ |
| 6 | 1 | 1 | 0 | 1 | |
| 7 | 1 | 1 | 1 | 1 | |

```
If variable A is Low(0) - A
           A is High(1) - A'
```

The function X can be written in POS form by multiplying all the max-terms when X is LOW(0).

While writing POS, the following convention is to be followed:

So for the above example from the truth table F=0 is in rows 1, 4, 5 hence:

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

This is POS form. POS form can be expressed as:

$$F = \prod(1, 4, 5)$$

This form is called the Canonical Product of Sum (Canonical POS).

## What is the symbol for SOP and POS?

*SOP and POS are two forms of Boolean expression where SOP is denoted with the sign summation ∑ and POS is denoted by pi notation* ***Π****.*

**Example of SOP form = AB + BC + CA**
**Example of POS form = (A + B)(B + C)(C + A)**

# Converting an SOP Expression into a Truth Table

Consider the following *sum of product* expression:

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

## Sum of Product Truth Table Form

| Inputs | | | Output | Product |
|---|---|---|---|---|
| C | B | A | Q | |
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 0 | |
| 0 | 1 | 1 | 1 | $A.B.\overline{C}$ |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 1 | $A.\overline{B}.C$ |
| 1 | 1 | 0 | 1 | $\overline{A}.B.C$ |
| 1 | 1 | 1 | 0 | |

***Example :***The following Boolean Algebra expression is given as:

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

**1.** Use a truth table to show all the possible combinations of input conditions that will produces an output.

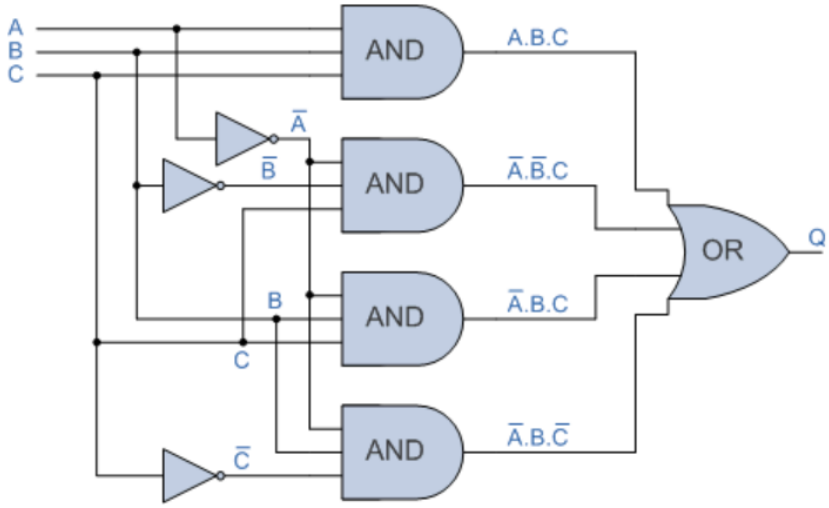
2. Draw a logic gate diagram for the expression.

Solution :

1.Sum of Product Truth Table Form

| Inputs | | | Output | Product |
|---|---|---|---|---|
| C | B | A | Q | |
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | **1** | $\overline{A}.B.\overline{C}$ |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | **1** | $\overline{A}.\overline{B}.C$ |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | **1** | $\overline{A}.B.C$ |
| 1 | 1 | 1 | **1** | $A.B.C$ |

2. Logic Gate SOP Diagram



# Converting an POS Expression into a Truth Table

Consider the following *product of sum* expression:

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

We can now draw up the truth table for the above expression to show a list of all the possible input combinations for A, B and C which will result in an output "0".

**Product of Sum Truth Table Form**

| Inputs | | | Output | Product |
|---|---|---|---|---|
| C | B | A | Q | |
| 0 | 0 | 0 | 0 | $A+B+C$ |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 0 | $A+\overline{B}+C$ |
| 0 | 1 | 1 | 1 | |
| 1 | 0 | 0 | 1 | |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 0 | $A+\overline{B}+\overline{C}$ |
| 1 | 1 | 1 | 1 | |

***Example:*** The following Boolean Algebra expression is given as:

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

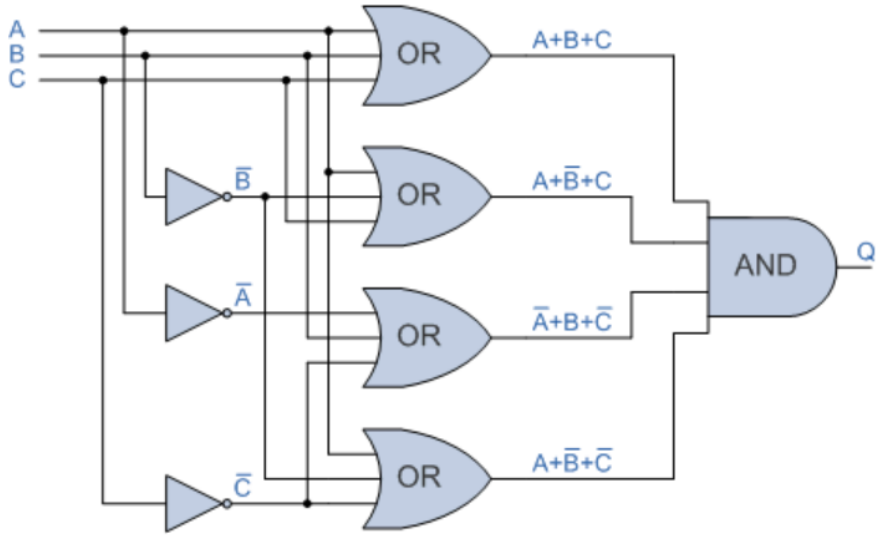
1. Use a truth table to show all the possible combinations of input conditions that will produces a “0” output.
2. Draw a logic gate diagram for the POS expression.

***Solution :***

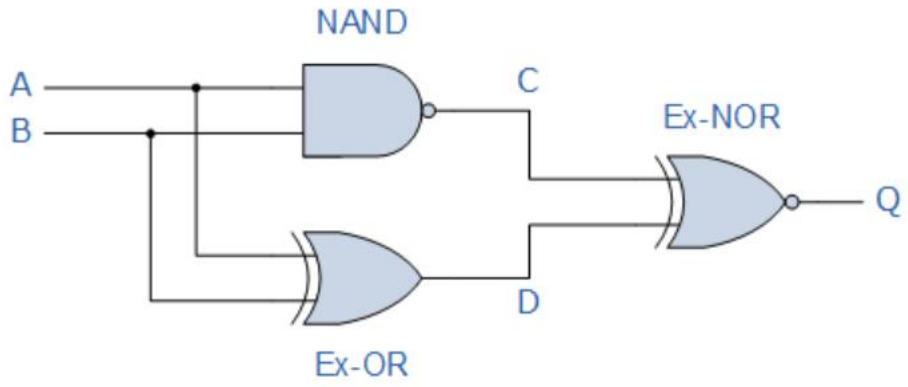
## 1.Product of Sum Truth Table Form

| Inputs | | | Output | Product |
|---|---|---|---|---|
| C | B | A | Q | |
| 0 | 0 | 0 | **0** | $A+B+C$ |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | **0** | $A+\overline{B}+C$ |
| 0 | 1 | 1 | 1 | |
| 1 | 0 | 0 | 1 | |
| 1 | 0 | 1 | **0** | $\overline{A}+B+\overline{C}$ |
| 1 | 1 | 0 | **0** | $A+\overline{B}+\overline{C}$ |
| 1 | 1 | 1 | 1 | |

2. Logic Gate Diagram



***Examples:*** Construct a Truth Table for the logical functions at points C, D and Q in the following circuit and identify a single logic gate that can be used to replace the whole circuit.

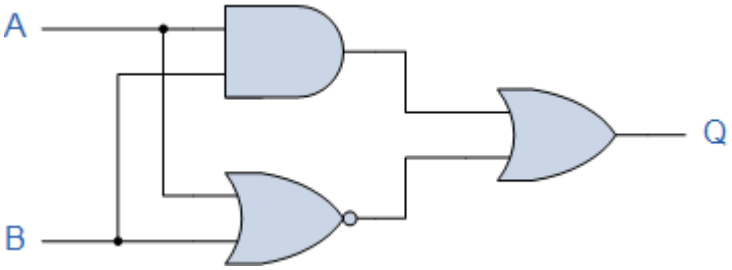



| Inputs | | Output at | | |
|---|---|---|---|---|
| A | B | C | D | Q |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |

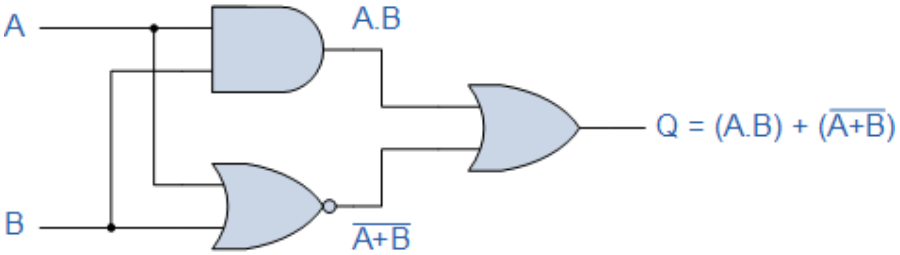
From the truth table above, column C represents the output function generated by the NAND gate, while column D represents the output function from the Ex-OR gate. Both of these two output expressions then become the input condition for the Ex-NOR gate at the output.

It can be seen from the truth table that an output at Q is present when any of the two inputs A or B are at logic 1. The only truth table that satisfies this condition is that of an OR Gate. Therefore, the whole of the above circuit can be replaced by just one single 2-input OR Gate.

***Examples :*** Find the Boolean algebra expression for the following system.



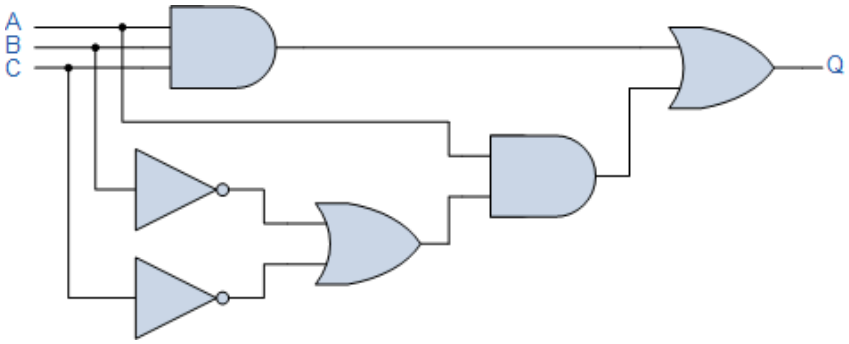

The system consists of an AND Gate, a NOR Gate and finally an OR Gate. The expression for the AND gate is A.B, and the expression for the NOR gate is A+B. Both these expressions are also separate inputs to the OR gate which is defined as A+B. Thus the final output expression is given as:



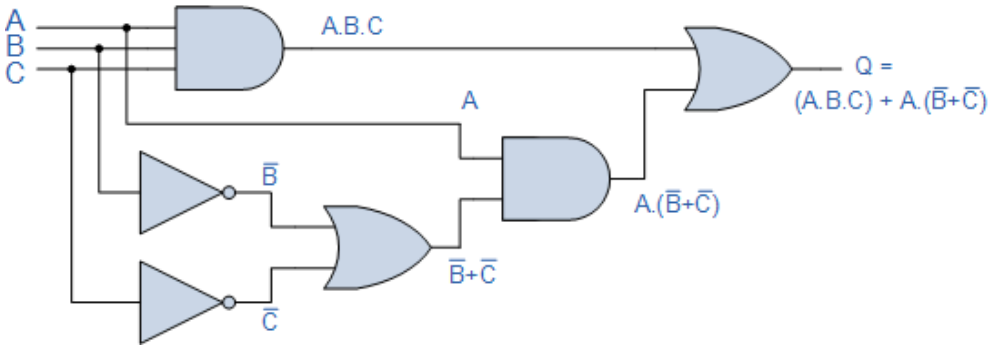

| Inputs | | Intermediates | | Output |
|---|---|---|---|---|
| B | A | A.B | $\overline{A+B}$ | Q |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

Then, the whole circuit above can be replaced by just one single Exclusive-NOR Gate and indeed an Exclusive-NOR Gate is made up of these individual gate functions.

*Example* **:** Find the Boolean algebra expression for the following system.




This system may look more complicated than the other two to analyses but again, the logic circuit just consists of simple AND, OR and NOT gates connected together.



| Inputs | | | Intermediates | | | | | Output |
|---|---|---|---|---|---|---|---|---|
| C | B | A | A.B.C | $\overline{B}$ | $\overline{C}$ | $\overline{B}+\overline{C}$ | $A.(\overline{B}+\overline{C})$ | Q |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

# 4/ Post test :-

1. Draw the circuit diagrams to show how a NOR gate can be made into a NOT gate.

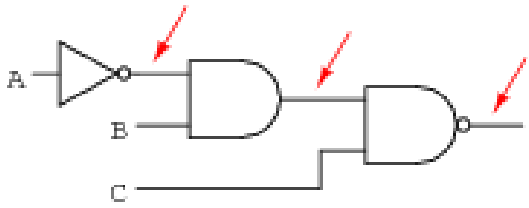
# key answer :-

**1- post test :-**

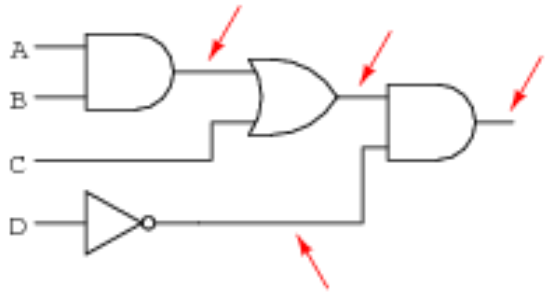


# 5/ HomeWorks: -

1.Convert the following logic gate circuit into a Boolean expression, writing Boolean sub-expressions next to each gate output in the diagram:



2.Convert the following logic gate circuit into a Boolean expression, writing Boolean sub-expressions next to each gate output in the diagram:

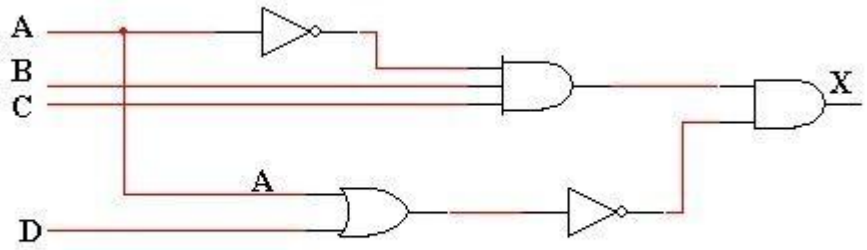



3.An engineer hands you a piece of paper with the following Boolean expression on it, and tells you to build a gate circuit to perform that function:

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

Draw a logic gate circuit for this function.

4.determine the output expression for the following circuit




determine the output logic level if A=1, B=1 and C=0, D=0

5. Design a logic circuit to verify the following functions
   - $F = \sum(0,1,2,4,6,9,11)$
   - $F = \boldsymbol{\Pi} \sum(0,1,2,5,6)$

A. Draw the circuit for the function shown below :

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

$$Q = (\overline{X + Y}) + (\overline{Z.W}$$

**Ministry of high Education and Scientific Research**
**Southern Technical University**
**Technological institute of Basra**
**Department of Computer Networks and Software Techniques**

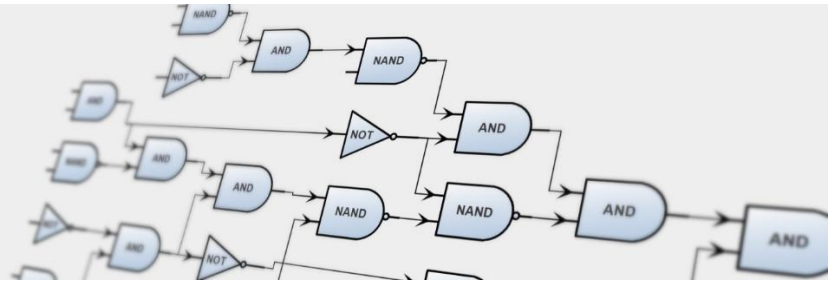
# Learning package

**In**

# Boolean Algebra

**For**

Students of First Year




**By**
**Ethar Abduljabbar Hadi**
**Assistant Lecturer**
**Dep. Of Computer Networks and Software Techniques**
**2025**

# 1/ Overview

## 1 / A –Target population :-

For First year students
Technological institute of Basra
Dep. Of Computer Networks and Software Technologies

## 1 / B –Rationale :-

Boolean algebra, in the context of logic, provides a powerful framework for analyzing and simplifying logical statements and arguments. It allows us to represent logical relationships using algebraic techniques, making it easier to determine the validity of inferences and the structure of logical systems. This is crucial in fields like digital circuit design, computer programming, and philosophical logic.

## 2 / C –Central Idea :-

1. Understand Boolean Algebra rules.
2. Simplify Logical Expressions.

## 1 / D – Performance Objectives

After studying this unit, the student will be able to:-

1. Know the Boolean Algebra rules.
2. know how to Simplify Logical Expressions.

## 2/ Pretest :

**What is the advantage of minimizing the Logic circuits?**

# 3/ Boolean Algebra

Boolean Algebra is a branch of algebra that deals with boolean values—true and false. It is fundamental to digital logic design and computer science, providing a mathematical framework for describing logical operations and expressions

## Boolean Algebra Operations

Various operations are used in Boolean algebra but the basic operations that form the base of Boolean Algebra are.

- ***Negation*** *or NOT Operation*
- ***Conjunction*** *or AND Operation*
- ***Disjunction*** *or OR Operation*

## Laws for Boolean Algebra

The basic laws of the Boolean Algebra are added in the table added below,

| NO. | Law | OR form | AND form |
|---|---|---|---|
| **1.** | **Identity Law** | A + 0 = A | A.1 = A |
| **2.** | **Idempotent Law** | A + A = A | A.A = A |
| **3.** | **Commutative Law** | A + B = B + A | A.B = B.A |
| **4.** | **Associative Law** | A+ (B + C) = (A + B) + C | A.(B.C) = (A.B).C |
| **5.** | **Distributive Law** | A + BC = (A + B).(A + C) | A.(B + C) = A.B + A.C |
| **6.** | **Inversion Law** | (A')' = A | (A')' = A |
| **7.** | **De Morgan's Law** | (A + B)' = (A)'.(B)' | (A.B)' = (A)' + (B)' |

# 1. Boolean Algebraic Identities

In mathematics, an identity is a statement true for all possible values of its variable or variables.

The algebraic identity of x + 0 = x tells us that anything (x) added to zero equals the original "anything," no matter what value that "anything" (x) may be.

Like ordinary algebra, Boolean algebra has its own unique identities based on the bivalent states of Boolean variables.

## 1.1 Additive Identities

### 1.1.1 Adding Zero

The first Boolean identity is that the sum of **anything** and **zero** is the same as the original "**anything**."
This identity is no different from its real-number algebraic equivalent:

$$A+0=A$$


No matter what the value of **A**, the output will always be the same: when **A=1**, the output will also be **1**; when **A=0**, the output will also be **0**.

### 1.1.2 Adding One

The next identity is most definitely different from any seen in normal algebra.
Here we discover that the sum of "**anything**" and **one** is **one**:

$$A+1=1$$


No matter what the value of A, the sum of A and 1 will always be 1. In a sense, the "1" signal overrides the effect of A on the logic circuit, leaving the output fixed at a logic level of 1.

### 1.1.3 Adding a Quantity to Itself

Next, we examine the effect of adding **A** and **A** together, which is the same as connecting both inputs of an **OR gate** to each other and activating them with the same signal:

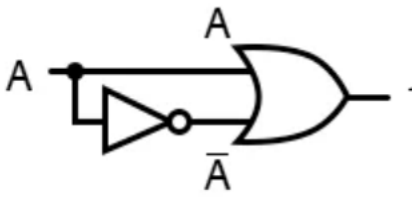
$$A+A=A$$


Thus, when we add a Boolean quantity to itself, the sum is equal to the original quantity: **0 + 0 = 0**, and **1 + 1 = 1**

### 1.1.4 Adding a Quantity to Its Complement

Introducing the uniquely Boolean concept of complementation into an additive identity, we find an interesting effect.
Since there must be one "1" value between any variable and its complement, and since the sum of any Boolean quantity and 1 is 1, the sum of a variable and its complement must be 1:

$$A+A'=1$$

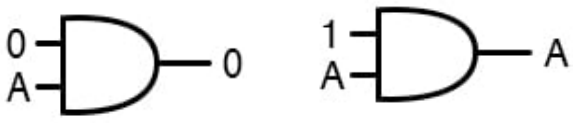



## 1.2 Multiplicative Identities

Just as there are four Boolean additive identities **(A+0, A+1, A+A, and A+A')**, so there are also four multiplicative identities: **Ax0, Ax1, AxA, and AxA'**. Of these, the first two are no different from their equivalent expressions in regular algebra:

### 1.2.1 Multiplying by 0 or 1

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



### 1.2.2 Multiplying a Quantity by Itself

The third multiplicative identity expresses the result of a Boolean quantity multiplied by itself.
since 0 x 0 = 0 and 1 x 1 = 1:

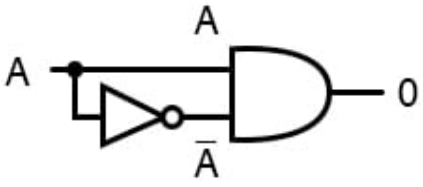
$$A.A=A$$


### 1.2.3 Multiplying a Quantity by Its Complement

The fourth multiplicative identity has no equivalent in regular algebra because it uses the complement of a variable, a concept unique to Boolean mathematics.

Since there must be one "0" value between any variable and its complement, and since the product of any Boolean quantity and 0 is 0, the product of a variable and its complement must be 0:

$$A.A'=A$$



To summarize, then, we have four basic Boolean identities for addition and four for multiplication:

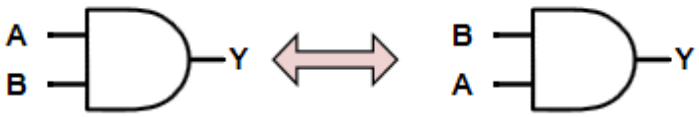
**Basic Boolean Algebraic Identities**

| Additive | Multiplicative |
|---|---|
| $A + 0 = A$ | $0A = 0$ |
| $A + 1 = 1$ | $1A = A$ |
| $A + A = A$ | $AA = A$ |
| $A + \overline{A} = 1$ | $A\overline{A} = 0$ |

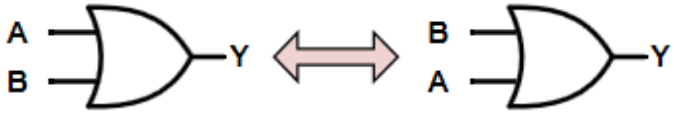
## 2. Commutative Law

Binary variables in Boolean Algebra follow the commutative law. This law states that operating boolean variables A and B is similar to operating boolean variables B and A. That is,

- $A.\ B = B.\ A$



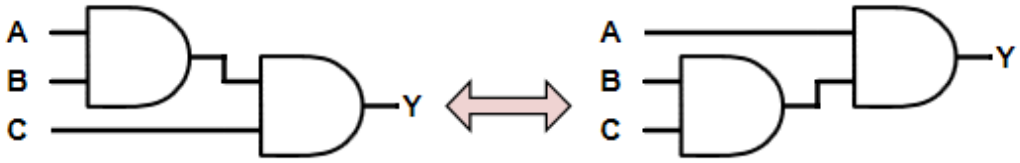

- $A + B = B + A$



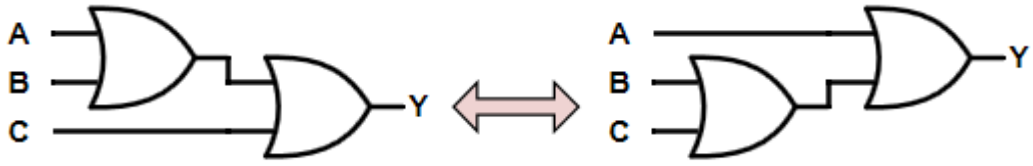
## 3.Associative Law

Associative law state that the order of performing Boolean operator is illogical as their result is always the same. This can be understood as,

- $(A \cdot B) \cdot C = A \cdot (B \cdot C)$



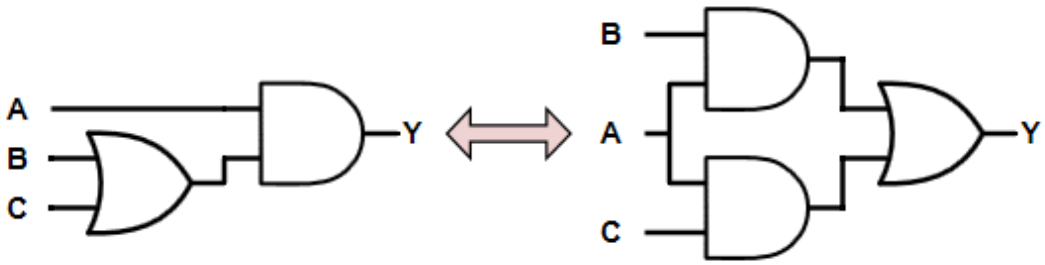

- $(A + B) + C = A + (B + C)$
-



## 4. Distributive Law

Boolean Variables also follow the distributive law and the expression for Distributive law is given as:

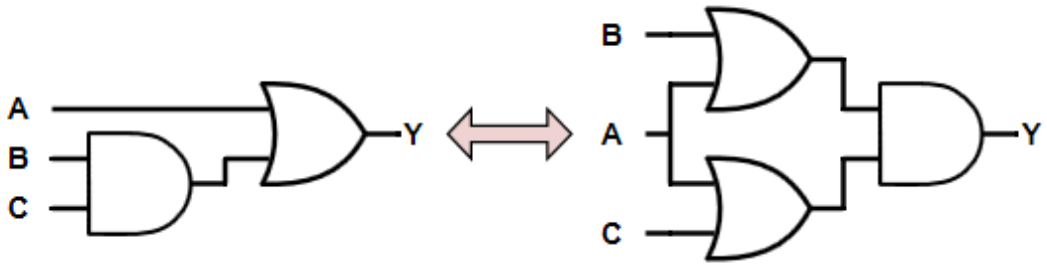
- $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$




PROOF: Distributivity of AND over OR

| NO. Of state | inputs | | | LHS | | RHS | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | (B+C) | A·(B+C) | (A.B) | (A.C) | (A·B)+(A·C) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

- $A + ( B . C) = (A + B) . (A + C)$




PROOF: Distributivity of OR over AND

| NO. Of state | inputs | | | LHS | | RHS | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | (B.C) | A+(B.C) | (A+B) | (A+C) | (A + B) . (A + C) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 5 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

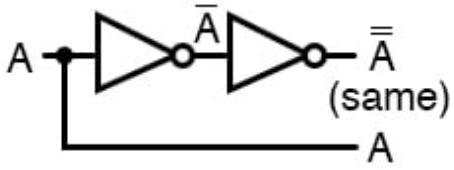
## 5.Double Complement

Another identity that has to do with complementation is that of the double complement: a variable inverted twice.

Complementing a variable twice (or any even number of times) results in the original Boolean value.

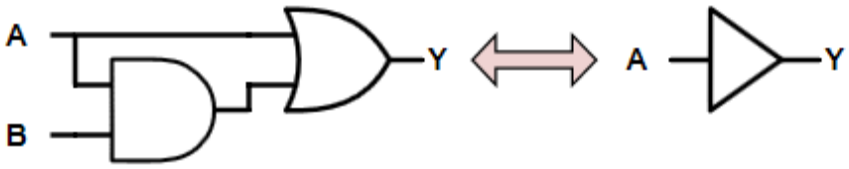
This is analogous to negating (multiplying by -1) in real-number algebra: an even number of negations cancel to leave the original value:

$$\overline{\overline{A}} = A$$




## *Absorption Law*

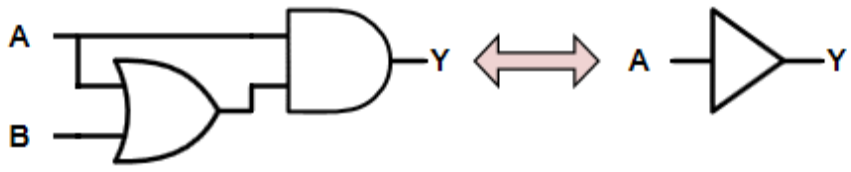
One of the more useful Boolean identities is absorption because it allows users to remove unneeded variables. However, in addition, it also allows us to introduce variables that then frequently allow us to make even greater simplifications



can just distribute the OR over the AND. Let's use the first approach as this is the one that is usually easier to see in practice.

$$A+(A\cdot B)=(A+A).(A+B)$$
$$= A(A+B)$$
$$=AA+AB$$
$$= A+AB$$
$$=A(1+B)$$
$$=A$$

| NO. Of State | inputs | | *LHS* | | *RHS* |
|---|---|---|---|---|---|
| | *A* | *B* | *(A.B)* | *A+(A.B)* | *A* |
| *0* | *0* | *0* | *0* | *0* | *0* |
| *1* | *0* | *1* | *0* | *0* | *0* |
| *2* | *1* | *0* | *0* | *1* | *1* |
| *3* | *1* | *1* | *1* | *1* | *1* |




$$A\cdot(A+B)=(AA)+(A.B)$$
$$=A+AB$$
$$=A(1+B)$$
$$=A$$

| NO. Of state | inputs | | *LHS* | | *RHS* |
|---|---|---|---|---|---|
| | *A* | *B* | *(A+B)* | *A.(A+B)* | *A* |
| *0* | *0* | *0* | *0* | *0* | *0* |
| *1* | *0* | *1* | *1* | *0* | *0* |
| *2* | *1* | *0* | *1* | *1* | *1* |
| *3* | *1* | *1* | *1* | *1* | *1* |

Two very important rules of simplification in Boolean algebra are as follows:

Rule 1: $A + AB = A$

Rule 2: $A + \overline{A}B = A + B$

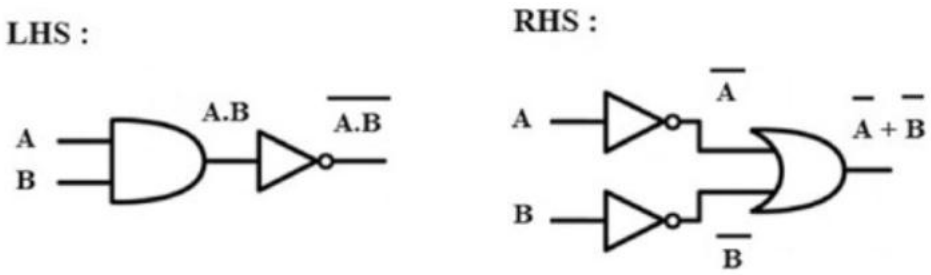
# Boolean Algebra Theorems

There are two basic theorems of great importance in Boolean Algebra, which are De Morgan's First Laws, and De Morgan's Second Laws. These are also called De Morgan's Theorems. Now let's learn about both in detail.

## 1.De Morgan's First laws

**De Morgan's Law** states that the complement of the product (AND) of two Boolean variables (or expressions) is equal to the sum (OR) of the complement of each Boolean variable (or expression).

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



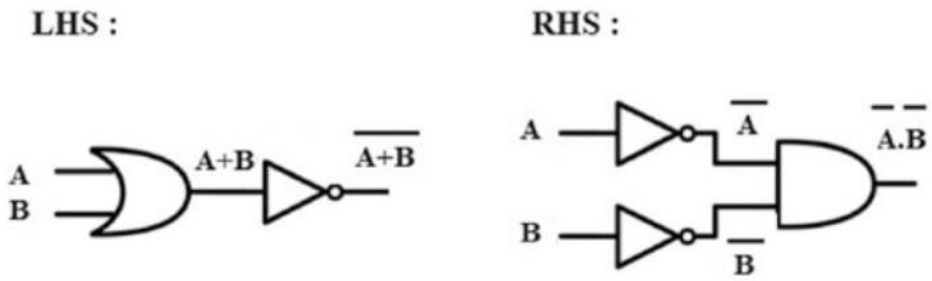

**The truth table for the same is given below:**

| A | B | (A)' | (B)' | (A.B)' | (A)' + (B)' |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 |

## 2. De Morgan's Second laws

**Statement:** The Complement of the sum (OR) of two Boolean variables (or expressions) is equal to the product(AND) of the complement of each Boolean variable (or expression).

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




**Proof:**
**The truth table for the same is given below:**

| A | B | (A)' | (B)' | (A+B)' | (A)'.(B)' |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |

We can clearly see that truth values for (A + B)' are equal to truth values for (A)'.(B)', corresponding to the same input. Thus, De Morgan's Second Law is true

**Advantages, Disadvantages of Boolean Algebra :**

**Advantages**

- Simplifies the design and analysis of digital circuits.
- Reduces the complexity of logical expressions and functions.
- Enhances efficiency in digital logic design and computer programming.

**Disadvantages**

- Limited to binary values, which may not always represent real-world complexities.
- Requires a strong understanding of logical operators and rules.

## 4/ Post test :-

Proof the first and second laws of De Morgan's .

## key answer :-

**1- post test :-**

# 5/ HomeWorks: -

1. Simplify the following expression using Boolean algebra, then design a logic circuit to verify it.

$$F = (\bar{X} + \bar{Y})(X + Y) + \overline{\overline{(\bar{X} + Y)(X + \bar{Y})}}$$

2. Simplify the following expression using Boolean algebra, then design a logic circuit to verify it.

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

3. design the logic circuit with following requirements:

F=1 , If the inputs less than 5
F=0 , Otherwise

A. Write the truth table .
B. Write the SOP function.
C. Simplify F .
D. Draw the circuit in step 3.

**Ministry of high Education and Scientific Research**
**Southern Technical University**
**Technological institute of Basra**
**Department of Computer Networks and Software Techniques**

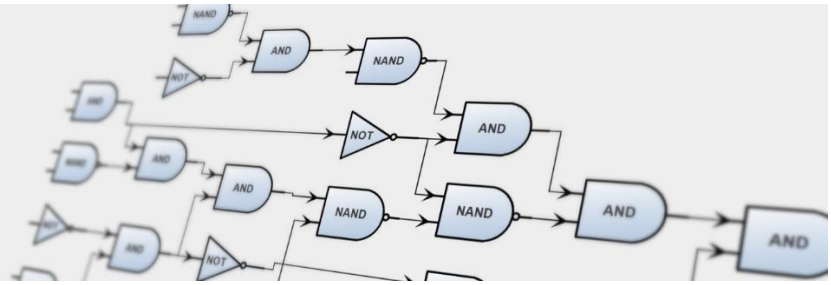
# Learning package

**In**

# K-Map (Karnaugh Map)

**For**

Students of First Year




**By**
**Ethar Abduljabbar Hadi**
**Assistant Lecturer**
**Dep. Of Computer Networks and Software Techniques**
**2025**

# 1/ Overview

## 1 / A –Target population :-

For First year students
Technological institute of Basra
Dep. Of Computer Networks and Software Technologies

## 1 / B –Rationale :-

Karnaugh Maps (K-maps), in the context of logic, provides a powerful framework for analyzing and simplifying logical statements and arguments. It allows us to represent logical relationships using algebraic techniques, making it easier to determine the validity of inferences and the structure of logical systems. This is crucial in fields like digital circuit design, computer programming, and philosophical logic.

We study Karnaugh Maps (K-maps) in logic design because they help us simplify Boolean expressions, which is essential in designing efficient digital circuits.

## 3 / C –Central Idea :-

1.Understand Karnaugh Maps (K-maps) rules.

2.Simplify Logical Expressions by using Karnaugh Maps (K-maps) .

3.Design of Efficient Digital Circuits

## 1 / D – Performance Objectives

After studying this unit, the student will be able to:-

1. Know the Karnaugh Maps (K-maps rules.

2. know how to Simplify Logical Expressions.

3. K-maps make it easier to minimize Boolean functions without needing long algebraic manipulations.

4. Simplified expressions use fewer logic gates, saving space and power.

## 2/ Pretest :

**1.Why we need another simplification method (k-map) ?**

**2. What is the difference between K-maps and Boolean Algebra to minimize Boolean functions ?**

## 3 / K-Map (Karnaugh Map)

In many digital circuits and practical problems, we need to find expressions with minimum variables. We can minimize Boolean expressions of 3, 4 variables very easily using K-map without using any Boolean algebra theorems.

**Steps to Solve Expression using K-map**

1. Select the K-map according to the number of variables.
2. Identify minterms or maxterms as given in the problem.
3. For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).
4. For POS put 0's in blocks of K-map respective to the max terms (1's elsewhere).
5. Make rectangular groups containing total terms in power of two like 2,4,8 ..(except 1) and try to cover as many elements as you can in one group.
6. From the groups made in step 5 find the product terms and sum them up for SOP form.

## 2 Variable Truth Table and K-Map

A logical specification is often created using a truth table. A truth table is a list of the inputs (A, B) on the left and the corresponding output (F) on the right. See Figure 1 showing a 2 variable truth table and corresponding K-Map.

2 Variable Truth Table

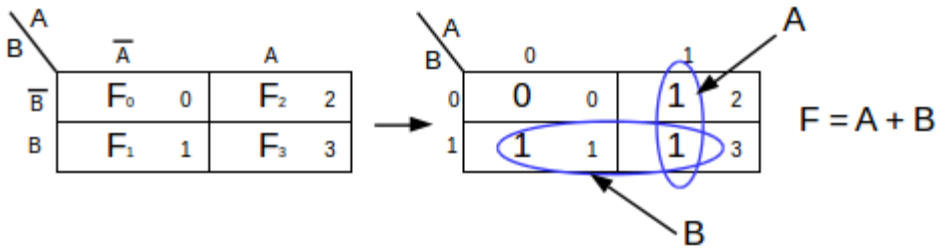
| | A | B | F |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 2 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 |

K-Map

| B \ A | 0 | 1 |
|---|---|---|
| 0 | 0 (0) | 1 (2) |
| 1 | 1 (1) | 1 (3) |

Each cell of the K-Map represents an input state (A, B). The value of each cell represents the output function (F). In order to find

the minimum logic function, it is necessary to identify matching adjacent cells. Once these matches are found, an expression can be written.




## 3 Variable Truth Table and K-Map

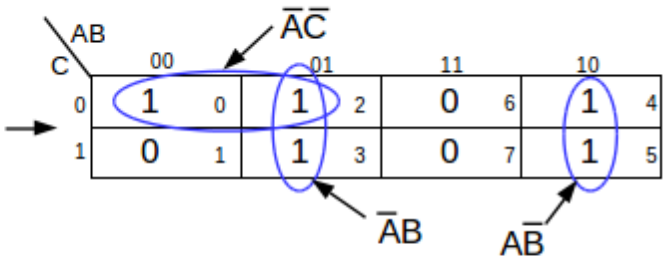
Below is an example of a 3 variable K-Map. Notice that the cells are ordered in the K-Map to ensure only one bit changes on any adjacent cell. From left to right instead of 0, 1, 2, 3, 4, 5 ,6, 7, the cell ordering is 0, 1, 2, 3, 6, 7, 4, 5.

| | | | |
|---|---|---|---|
| 000 | 010 | 110 | 100 |
| 001 | 011 | 111 | 101 |

3 Variable Truth Table

| | A | B | C | F |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 0 |

K-Map




Note : $\bar{A}B + A\bar{B} = A \oplus B$ (Exclusive OR)

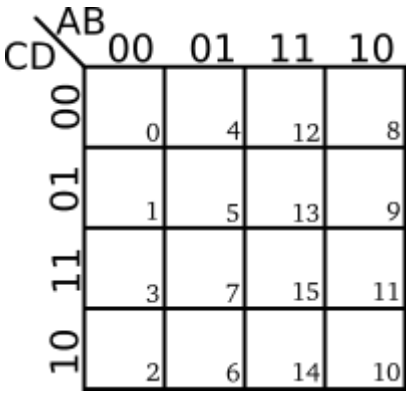
$F = \bar{A}\bar{C} + \bar{A}B + A\bar{B}$

$F = \bar{A}\bar{C} + A \oplus B$

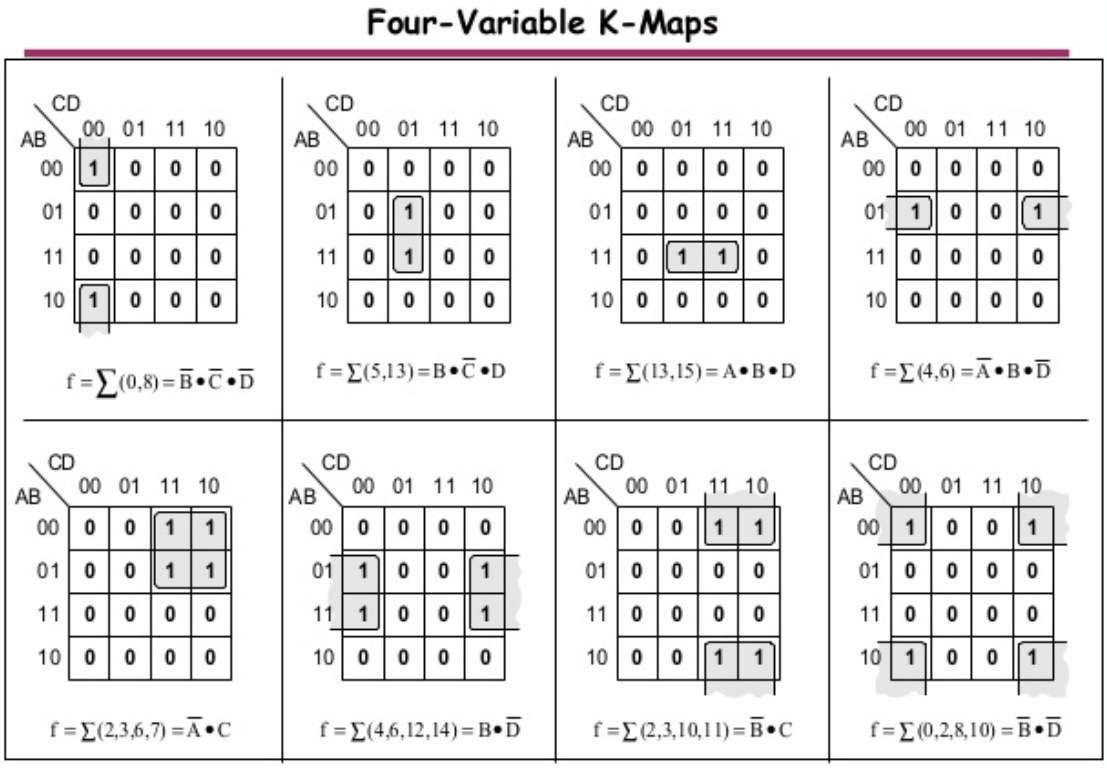
# 4 Variable Truth Table and K-Map

Below is an example of a 4 variable K-Map. Notice that the cells are ordered in the K-Map to ensure only one bit changes on any adjacent cell.

**4-Variables K-map**



Examples :




# 4/ Post test :-

For the following function : -

$$Q = \prod M(5,7,13,15)$$

1. Write the truth table for the function above.
2. By using K-map , Simplify Q

Design the circuit to implement the function in step 2.

# 5/ HomeWorks: -

1. Four chairs A,B,C,D are placed in a circle. Each chair may be occupied (“1”) or empty (“0”) .A Boolean function F is “1” if and only if there are two or more adjacent chairs that are empty .
    a. Give the truth table defining the Boolean function F.
    b. Simplify the function F and draw the logic circuit .

2. Design a digital system whose output is defined as logically low if the 4-bit input binary number is a multiple of 3; otherwise, the output will be logically high. The output is defined if and only if the input binary number is greater than 2.

In the given example:

- The number of input variables = 4, which we will call A, B, C, and D.
- The number of output variables = 1, which we will call Y.

Where:

- Y = "Don't Care," if the input number is less than 3 (orange entries in the truth table)
- Y = 0, if the input number is an integral multiple of 3 (green entries in the truth table)
- Y = 1, if the input number is not an integral multiple of 3 (blue entries in the truth table)

**Ministry of high Education and Scientific Research**
**Southern Technical University**
**Technological institute of Basra**
**Department of Computer Networks and Software Techniques**

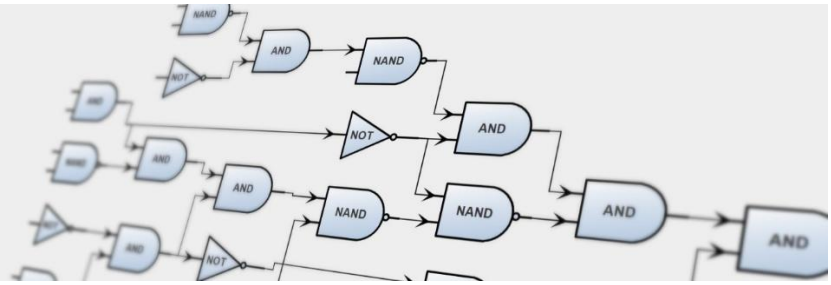
# Learning package

**In**

# Combinational Logic Circuits

**For**

Students of First Year




**By**
**Ethar Abduljabbar Hadi**
**Assistant Lecturer**
**Dep. Of Computer Networks and Software Techniques**
**2025**

# 1/ Overview

## 1 / A –Target population :-

For First year students
Technological institute of Basra
Dep. Of Computer Networks and Software Technologies

## 1 / B –Rationale :-

The idea of studying combinational logic circuits is rooted in the goal of understanding and designing circuits that produce outputs based solely on current inputs — with no memory or feedback involved. These circuits form the foundation of digital systems, such as computers, calculators, and embedded systems.. This is crucial in fields like digital circuit design, computer programming, and philosophical logic.

## 1 / C –Central Idea :-

1.Understand the Foundation of Digital Electronics.

2. learning how to design combinational logic Circuits

3.Design Digital Circuits like Adder, subtracter.

## 1 / D – Performance Objectives

After studying this unit, the student will be able to:-

Understand Foundation of Digital Electronics where , Combinational logic is the building block for all digital systems.

It includes essential components like adders, multiplexers, decoders, encoders, and comparators

## 2/ Pretest :

**What do you know about Combinational Logic Circuits?**

# 3 / Combinational Logic Circuits

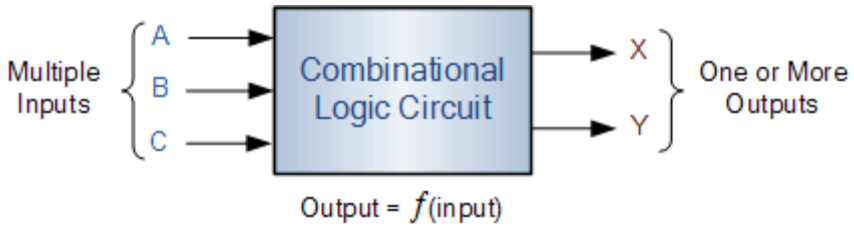
Combinational Logic Circuits are memoryless digital logic circuits whose output at any instant in time depends only on the combination of its inputs.

Unlike Sequential Logic Circuits whose outputs are dependant on both their present inputs and their previous output state giving them some form of Memory. The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic “0” or logic “1”, at any given instant in time.

The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a Combinational Logic Circuit, the output is dependant at all times on the combination of its inputs. Thus a combinational circuit is memoryless.

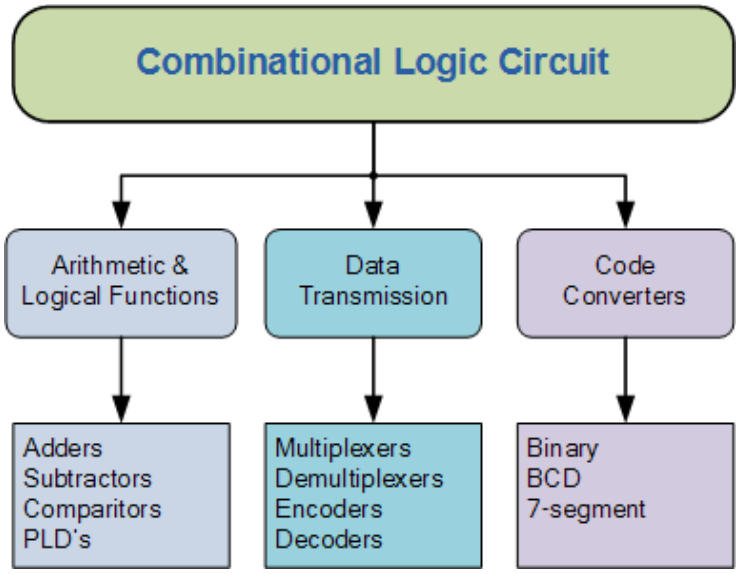
So if one of its inputs condition changes state, from 0-1 or 1-0, so too will the resulting output as by default combinational logic circuits have “no memory”, “timing” or “feedback loops” within their design.

## Combinational Logic




Common combinational circuits made up from individual logic gates that carry out a desired application include *Multiplexers*, *De-multiplexers*, *Encoders*, *Decoders*, *Full* and *Half Adders* etc.

## Classification of Combinational Logic



# Binary Adders

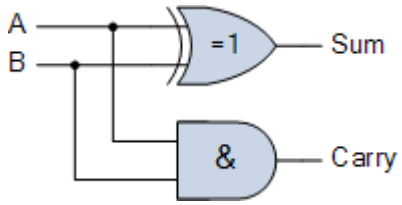
A common and very useful combinational logic circuit which can be constructed using just a few basic logic gates allowing it to add together two or more binary numbers is the **Binary Adder.**

| | | |
|---|---|---|
| 123 | A | (Augend) |
| + 789 | B | (Addend) |
| 912 | SUM | |

Binary Adders are arithmetic circuits in the form of half-adders and full-adders used to add together two binary digits.

## *1.Half Adder Circuit*

A half adder is a logical circuit that performs an addition operation on two binary digits. The half adder produces a sum and a carry value which are both binary digits.

| Logic circuit | Truth Table | | | |
|---|---|---|---|---|
|  | A | B | SUM | CARRY |
| | 0 | 0 | 0 | 0 |
| | 0 | 1 | 1 | 0 |
| | 1 | 0 | 1 | 0 |
| | 1 | 1 | 0 | 1 |



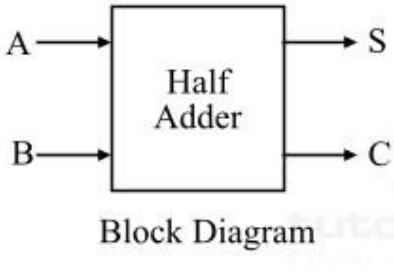
For the **SUM** bit:

$$SUM = A \text{ XOR } B = A \oplus B$$

For the **CARRY** bit:

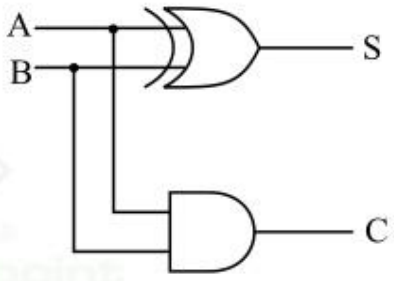
$$CARRY = A \text{ AND } B = A.B$$

From the truth table of the half adder we can see that the SUM (S) output is the result of the Exclusive-OR gate and the Carry-out (Cout) is the result of the AND gate. Then the Boolean expression for a half adder is as above.





Block Diagram



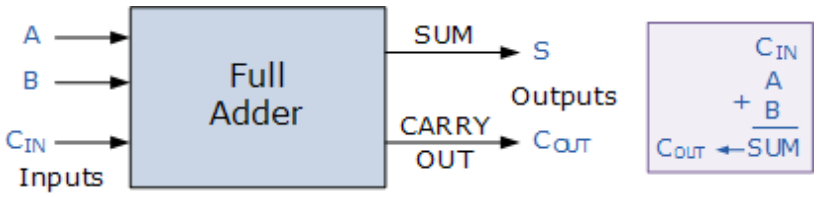


Circuit Diagram

## *2.Full Adder Circuit*

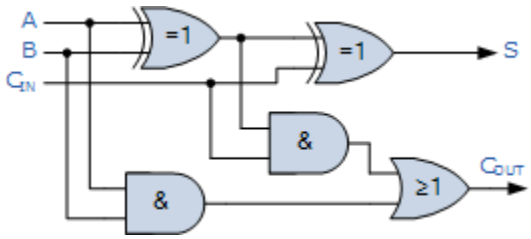
The main difference between **the Full Adder** and the previous **Half Adder** is that a full adder has three inputs. The same two single bit data inputs A and B as before plus an additional *Carry-in (C-in*) input to receive the carry from a previous stage as shown below.

**Full Adder Block Diagram**




Then the **full adder** is a logical circuit that performs an addition operation on three binary digits and just like the half adder, it also generates a carry out to the next addition column. Then a Carry-in is a possible carry from a less significant digit, while a Carry-out represents a carry to a more significant digit.

**Full Adder Truth Table with Carry**

| Logic circuit | Truth Table | | | | |
|---|---|---|---|---|---|
|  | A | B | C-in | Sum | C-out |
| | 0 | 0 | 0 | 0 | 0 |
| | 0 | 0 | 1 | 1 | 0 |
| | 0 | 1 | 0 | 1 | 0 |
| | 0 | 1 | 1 | 0 | 1 |
| | 1 | 0 | 0 | 1 | 0 |
| | 1 | 0 | 1 | 0 | 1 |
| | 1 | 1 | 0 | 0 | 1 |
| | 1 | 1 | 1 | 1 | 1 |



Then the Boolean expression for a full adder is as follows.

For the **SUM** (S) bit:

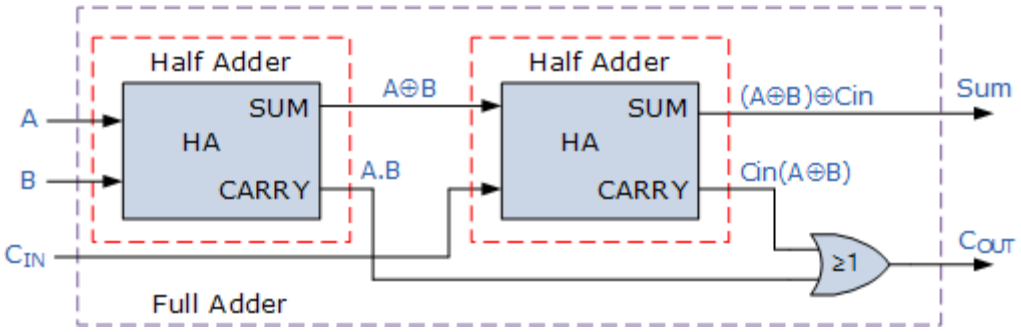
SUM = (A XOR B) XOR Cin = (A ⊕ B) ⊕ Cin

For the **CARRY-OUT** (Cout) bit:

CARRY

OUT = A AND B OR Cin(A XOR B) = A.B + Cin(A ⊕ B)

**(Implementation of Full Adder using Half Adder)**




**Full Binary Adder Logic Diagram**

As the full adder circuit above is basically two half adders connected together, the truth table for the full adder includes an additional column to take into account the Carry-in, CIN input as well as the summed output, S and the Carry-out, COUT bit.

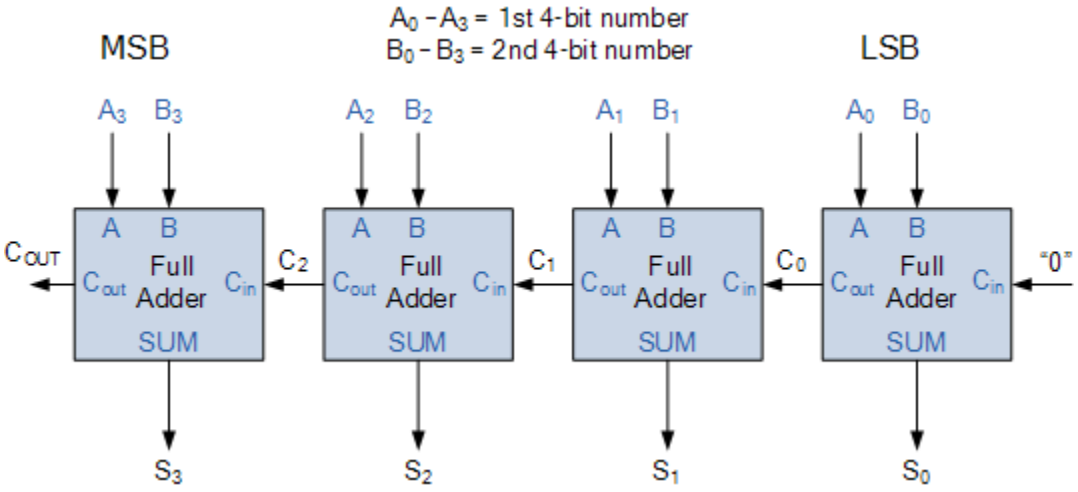
## *An n-bit Binary Adder*

We have seen above that single 1-bit binary adders can be constructed from basic logic gates. But what if we wanted to add together two n-bit numbers, then n number of 1-bit full adders need to be connected or "cascaded" together to produce what is known as **a Ripple Carry Adder.**

A “ripple carry adder” is simply “n“, 1-bit full adders cascaded together with each full adder representing a single weighted column in a long binary addition. It is called a ripple carry adder because the carry signals produce a “ripple” effect through the binary adder from right to left, (LSB to MSB).

For example, suppose we want to “add” together two 4-bit numbers, the two outputs of the first full adder will provide the first place digit sum (S) of the addition plus a carry-out bit that acts as the carry-in digit of the next binary adder.

## A 4-bit Ripple Carry Binary Adder



# Binary Subtractor

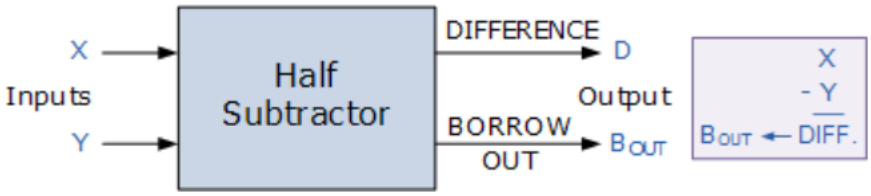
The Binary Subtractor is another type of combinational arithmetic circuit that produces an output which is the subtraction of two binary numbers.

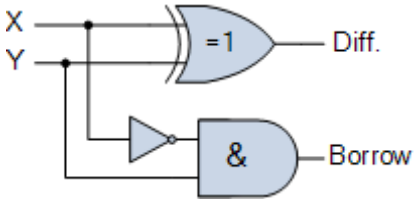
As their name implies, a Binary Subtractor is a decision making circuit that subtracts two binary numbers from each other, for example, X – Y to find the resulting difference between the two numbers. the binary subtractor produces a DIFFERENCE, D by using a BORROW bit, B from the previous column.

We learnt from our maths lessons at school that the minus sign, "–" is used for a subtraction calculation, and when one number is subtracted from another, a borrow is required if the subtrahend is greater than the minuend. Consider the simple subtraction of the two denary (base 10) numbers below.

| | | |
|---|---|---|
| 123 | X | |
| – 78 | Y | (Subtrahend) |
| 45 | DIFFERENCE | |

# 1. Half Subtractor




| Symbol | Truth Table | | | |
|---|---|---|---|---|
|  | X | Y | DIFFERENCE | BORROW |
| | 0 | 0 | 0 | 0 |
| | 0 | 1 | 1 | 1 |
| | 1 | 0 | 1 | 0 |
| | 1 | 1 | 0 | 0 |



From the truth table of the half subtractor we can see that the **DIFFERENCE (D)** output is the result of the Exclusive-OR gate and the **Borrow-out (Bout)** is the result of the NOT-AND combination. Then the Boolean expression for a half subtractor is as follows.

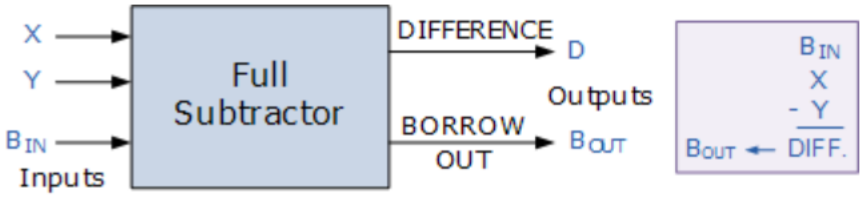
For the DIFFERENCE bit:

$$D = X \text{ XOR } Y = X \oplus Y$$

For the **BORROW** bit

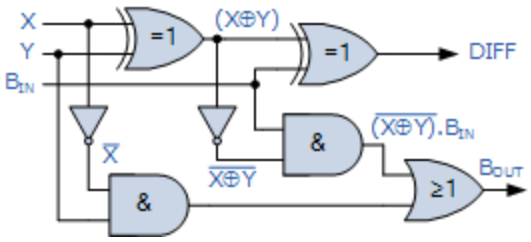
$$B = \text{not-}X \text{ AND } Y = X'.Y$$

# 2.Full Binary Subtractor Circuit




**Full Binary Subtractor Block Diagram**

**Full Subtractor Truth Table**

| Symbol | Truth Table | | | | |
|---|---|---|---|---|---|
|  | **X** | **Y** | **B-in** | **Diff.** | **B-out** |
| | **0** | **0** | **0** | **0** | **0** |
| | **0** | **0** | **1** | **1** | **1** |
| | **0** | **1** | **0** | **1** | **1** |
| | **0** | **1** | **1** | **0** | **1** |
| | **1** | **0** | **0** | **1** | **0** |
| | **1** | **0** | **1** | **0** | **0** |
| | **1** | **1** | **0** | **0** | **0** |
| | **1** | **1** | **1** | **1** | **1** |



Then the Boolean expression for a full subtractor is as follows. For the DIFFERENCE (D) bit:

$$D = (\bar{X}.\bar{Y}.B_{IN}) + (\bar{X}.Y.\bar{B_{IN}}) + (X.\bar{Y}.\bar{B_{IN}}) + (X.Y.B_{IN})$$

which can be simplified too:

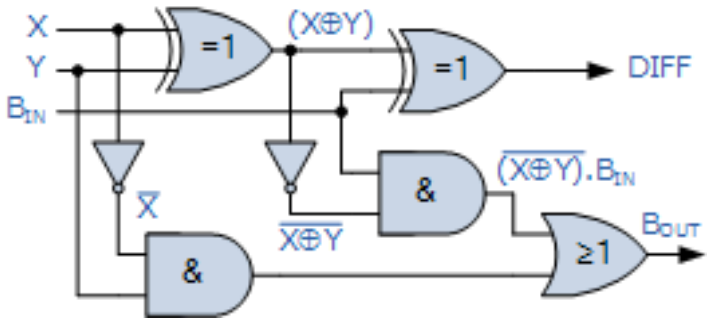
$$D = (X \text{ XOR } Y) \text{ XOR } B_{IN} = (X \oplus Y) \oplus B_{IN}$$

For the BORROW OUT ($B_{OUT}$) bit:

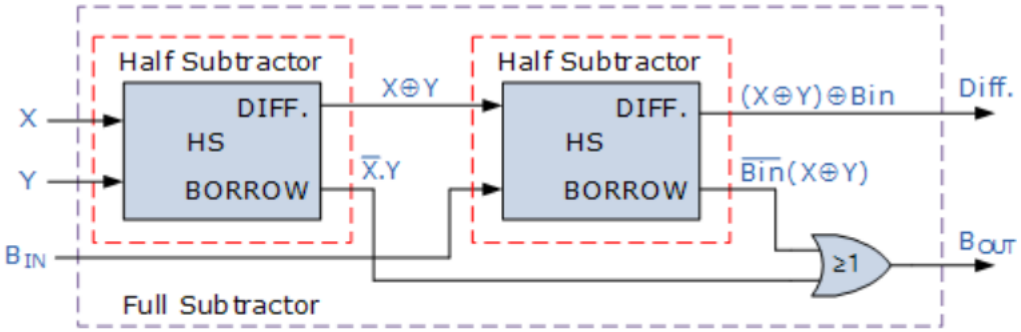
$$B_{OUT} = (\overline{X}.\overline{Y}.B_{IN}) + (\overline{X}.Y.\overline{B_{IN}}) + (\overline{X}.Y.B_{IN}) + (X.Y.B_{IN})$$

which will also simplify too:

$$B_{OUT} = \overline{X}\ \text{AND}\ Y\ \text{OR}\ \overline{(X\ \text{XOR}\ Y)}B_{IN} = \overline{X}.Y + \overline{(X \oplus Y)}B_{IN}$$




Then the combinational circuit of a "full subtractor" performs the operation of subtraction on three binary bits producing outputs for the difference D and borrow B-out. Just like the binary adder circuit, the full subtractor can also be thought of as two half subtractors connected together, with the first half subtractor passing its borrow to the second half subtractor as follows.




**Full Binary Subtractor Logic Diagram**

## ***Binary Adder-Subtractor***

A binary adder-subtractor is a digital circuit that is used to perform two basic arithmetic operations namely, binary addition and binary subtraction. It is an important component in various digital systems like computers, calculators, etc.

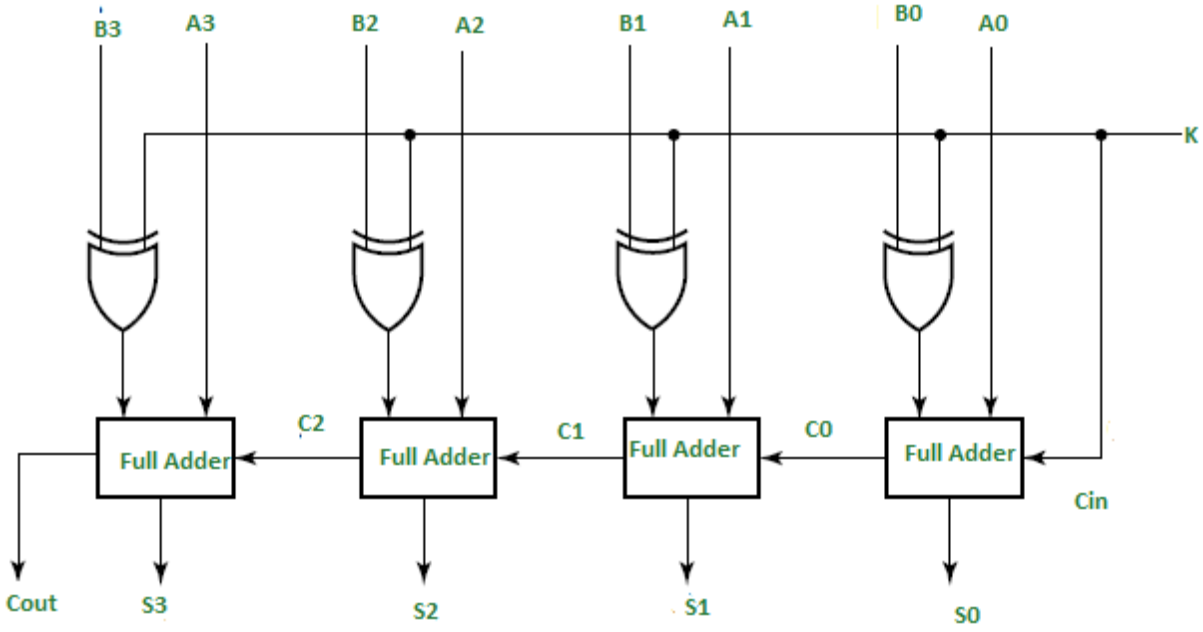
The most significant advantage of using a binary adder-subtractor is that it combines the addition and subtraction operations in a single circuit which results in compact size and lower cost.

This Circuit Requires prerequisite knowledge of Xor Gate, Binary Addition and Subtraction, and Full Adder.

Let's consider two 4-bit binary numbers A and B as inputs to the Digital Circuit for the operation with digits

A0 A1 A2 A3 for A
B0 B1 B2 B3 for B

The circuit consists of 4 full adders since we are performing operations on 4-bit numbers. There is a control line K that holds a binary value of either 0 or 1 which determines that the operation is carried out is addition or subtraction.




As shown in the figure, the first full adder has a control line directly as its input (input carry Cin), The input A0 (The least significant bit of A) is directly input in the full adder. The third input is the EXOR of B0 and K. The two outputs produced are Sum/Difference (S0) and Carry (C0).

In this circuit, the input K is called the mode input. It controls the operation of the circuit as described below:

- When K = 0, the circuit operates as a binary adder. Under this mode, we get B⊕0=B. Thus, each full adder receives the inputs A and B and performs their addition, i.e., A + B+0.
- When K = 1, the circuit operates as a binary subtractor. In this case, w get B⊕1=1s complement(B) and the input carry $C_{in}$ = 1. Under this mode, the full adders receive B inputs in their complemented form and a 1 is added through the input carry $C_{in}$. Hence, the final output of the circuit is A+ 2s complement of $B_x$ which is the subtraction of A and B.

## Advantage of Binary Adder and Subtractor

- **Low Design Complexity**: Both binary adder and subtractor circuits are easy to design using logic gates like XOR, AND, and OR.
- **High-Speed Operations**: Binary adders such as parallel adders-and subtractors can do their operations at a high speed.
- **Versatility**: It allows the same hardware : to add ,to subtract. Saving on redundancy of separate components and designs for different functions.

## Disadvantages of Binary Adder and Subtractor

- **Carry Propagation Delay**: In a simple ripple carry adder, the carry has to ripple through all stages of the adder; this increases the computation time as more bits are added. This will be more for a larger number of bits in binary.
- **Hardware Complexity**: The more the number of bits, the more complicated the circuit gets due to more gates. To eliminate the problem of delays, higher versions of adders such as carry lookahead adders are included which increases complexity.

## 4/ Post test :-

In Adder-Subtractor circuit , can you replace the X0R logic gate by another gate, Why?

## 5/ HomeWorks: -

By using Adder-Subtractor circuit, implement the operation (9-5)

**Ministry of high Education and Scientific Research**
**Southern Technical University**
**Technological institute of Basra**
**Department of Computer Networks and Software Techniques**

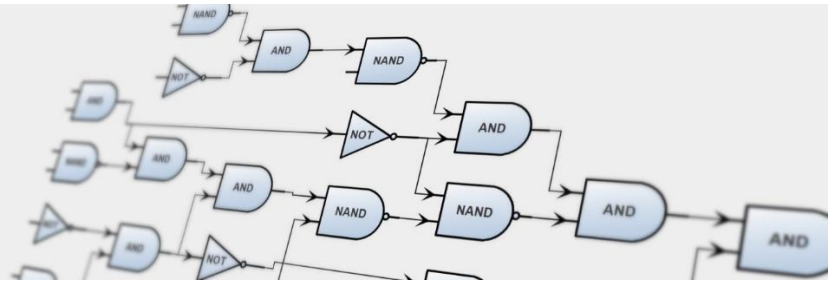
# Learning package

**In**

# Sequential Logic Circuits

**For**

Students of First Year




**By**
**Ethar Abduljabbar Hadi**
**Assistant Lecturer**
**Dep. Of Computer Networks and Software Techniques**
**2025**

# 1/ Overview

## 1 / A –Target population :-

For First year students
Technological institute of Basra
Dep. Of Computer Networks and Software Technologies

## 1 / B –Rationale :-

The idea of studying sequential logic circuits is to understand how digital systems can store, remember, and respond to sequences of inputs over time — in other words, how they gain memory and timing behavior. This is crucial in fields like digital circuit design, computer programming, and philosophical logic.

## 2 / C –Central Idea :-

1.Understand the Foundation of Digital Electronics.

2. learning how to design Sequential Logic Circuits.

## 1 / D – Performance Objectives

After studying this unit, the student will be able to:-

1. Understand Foundation of Digital Electronics where , Sequential logic is the building block for all digital systems.

Introduce the Concept of Memory

2. Unlike combinational circuits, sequential circuits remember past inputs using storage elements (like flip-flops).

3. Output depends on both current input and past states.

Sequential logic circuits allow digital systems to store data and operate over time, making them essential for building intelligent, interactive, and real-time systems like CPUs, timers, and state machines.

## 2/ Pretest :

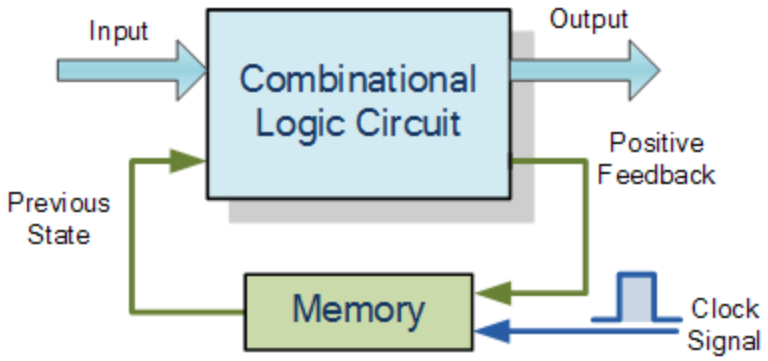
**What do you know about Sequential Logic Circuits?**

# 3 / Sequential Logic Circuits

Unlike Combinational Logic circuits that change state depending upon the actual signals being applied to their inputs at that time, Sequential Logic circuits have some form of inherent “Memory” built in.

This means that sequential logic circuits are able to take into account their previous input state as well as those actually present, a sort of “before” and “after” effect is involved with sequential circuits.

In other words, the output state of a “sequential logic circuit” is a function of the following three states, the “present input”, the “past input” and/or the “past output”. Sequential Logic circuits remember these conditions and stay fixed in their current state until the next clock signal changes one of the states, giving sequential logic circuits “Memory”.




The word “Sequential” means that things happen in a “sequence”, one after another and in Sequential Logic circuits, the actual clock signal determines when things will happen next. Simple sequential logic circuits can be constructed from standard Bistable circuits such as: Flip-flops, Latches and Counters

### Flip Flop in Digital Electronics

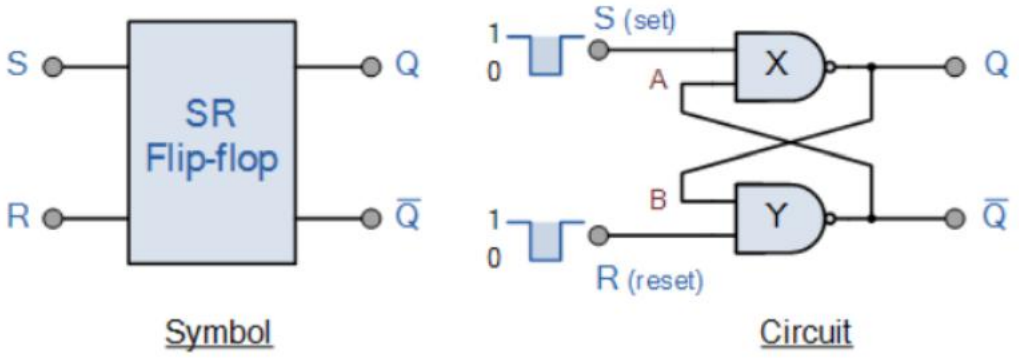
A flip-flop in digital electronics is a circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Both are used as data storage elements.

The term “Flip-flop” relates to the actual operation of the device, as it can be “flipped” into one logic Set state or “flopped” back into the opposing logic Reset state.

- Types of Flip-Flop in Electronics and Their Working
  - 1. S-R Flip Flop
  - 2. JK Flip-Flop
  - 3. D Flip-Flop
  - 4. T Flip-Flop

## 1.The Set-Reset (S-R) Flip-Flop.

An S-R flip-flop has two inputs named Set (S) and Reset (R), and two outputs Q and Q'. The outputs are complement of each other, i.e., if one of the outputs is 0 then the other should be 1. This can be implemented using NAND or NOR gates. The NAND gate S-R flip-flop is shown in Figure below:-

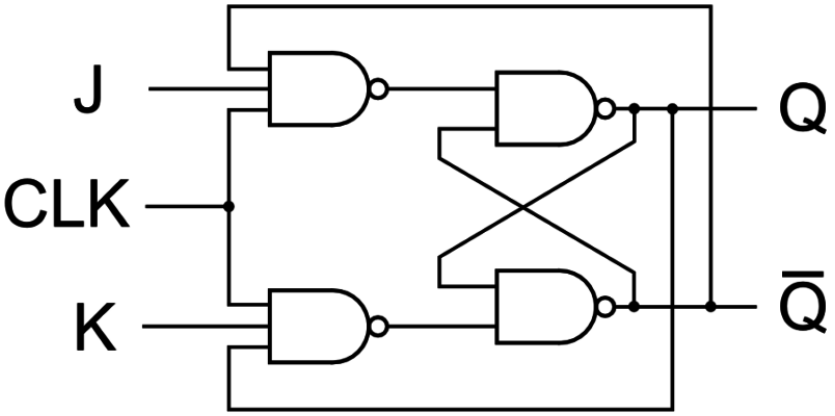



**Truth Table for this Set-Reset Function**

| State | S | R | Q | | Description |
|---|---|---|---|---|---|
| Set | 1 | 0 | 0 | 1 | Set $\overline{Q}$ » 1 |
| | 1 | 1 | 0 | 1 | no change |
| Reset | 0 | 1 | 1 | 0 | Reset $\overline{Q}$ » 0 |
| | 1 | 1 | 1 | 0 | no change |
| Invalid | 0 | 0 | 1 | 1 | Invalid Condition |

## 2. JK Flip-Flop

The JK flip-flop is an improvement on the SR flip-flop where S=R=1 is not a problem.




The input condition of J=K=1 gives an output inverting the output state. However, the outputs are the same when one tests the circuit practically.

In simple words, If J and K data input are different (i.e. high and low), then the output Q takes the value of J at the next clock edge. If J and K are both low, then no change occurs.

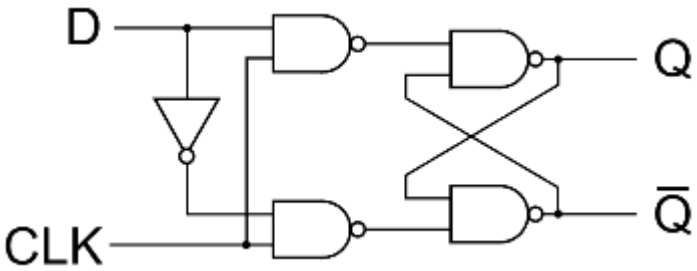
If J and K are both high at the clock edge, then the output will toggle from one state to the other. JK Flip-Flops can function as Set or Reset Flip-flops.

**JK Flip-flop Truth Table:**

| J | K | Q | Q′ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 |

## 3. D Flip-Flop

Delay or D flip-flop is a better alternative that is very popular with digital electronics. They are commonly used for counters, shift registers, and input synchronization.




D Flip-Flop Circuit

In the D flip-flops, the output can only be changed at the clock edge, and if the input changes at other times, the output will be unaffected.

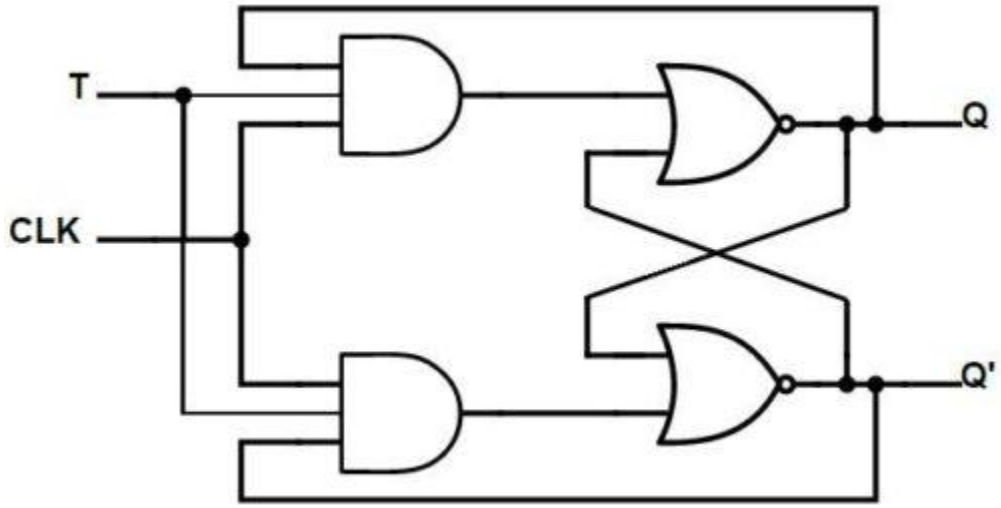
**Truth Table:**

| Clock | D | Q | Q' |
|---|---|---|---|
| ↓ » 0 | 0 | 0 | 1 |
| ↑ » 1 | 0 | 0 | 1 |
| ↓ » 0 | 1 | 0 | 1 |
| ↑ » 1 | 1 | 1 | 0 |

The change of state of the output is dependent on the rising edge of the clock. The output (Q) is the same as the input and can only change at the rising edge of the clock.

## 4. T Flip-Flop

A T flip-flop is like a JK flip-flop. These are single-input versions of JK flip-flops. This modified form of the JK is obtained by connecting inputs J and K together. It has only one input along with the clock input.




T Flip Flop Circuit

These flip-flops are called T flip-flops because of their ability to complement their state i.e. Toggle, hence they are named **Toggle flip-flops**.

**Truth Table:**

| T | Q | Q (t+1) |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

## Applications:

These are the various types of flip-flops being used in digital electronic circuits and the applications of Flip-flops are as specified below.

- Counters
- Frequency Dividers
- Shift Registers
- Storage Registers

# 1/ References


1. Holdsworth, Brian, and Clive Woods. Digital logic design. Elsevier, 2002.
2. Alam, Mansaf, and Bashir Alam. Digital Logic Design. PHI Learning Pvt. Ltd., 2015.
3. Dally, William James, and R. Curtis Harting. Digital design: a systems approach. Cambridge University Press, 2012.
4. https://www.geeksforgeeks.org/digital-logic/digital-electronics-logic-design-tutorials/
5. https://www.electronics-tutorials.ws/_1.html
6. https://www.geeksforgeeks.org/digital-logic/flip-flop-types-their-conversion-and-applications/#sr-flip-flop