REFERENCES

1. Mehta, V. K., & Mehta, R. (2017). Basic electrical and electronics engineering. Chand Publishing.
2. Theraja, B. L., & Theraja, A. K. (2017). Basic electronics: Solid state. S. Ch Publishing.
3. Aggarwal, V., & Singh, S. (2018). Fundamentals of electrical and electro engineering. Cengage Learning India.
4. Bakshi, U. A., & Bakshi, A. V. (2008). Basic electrical and electronics enginee Technical Publications.
5. Sharma, V. K., & Kumar, S. (2017). Fundamentals of electrical and electro engineering. PHI Learning Pvt. Ltd.
6. https://www.sanfoundry.com/network-theory-mcqs-ohms-law/
7. https://www.indiabix.com/electronics/bipolar-junction-transistors/
8. https://www.electricaltechnology.org/2013/04/three-phase-ac-circuits-mcqs.html
9. https://electricalvoice.com/thevenins-theorem-mcq/
10. https://www.mechanicaltutorial.com/ac-circuit-objective-questions-and-answers-02-04

DIGITAL LOGIC & MICROPROCESSOR (AExE02)

2.1 DIGITAL LOGIC

Number System: A number system is a writing system for expressing numbers. There are several different number systems, including the following:

Number system is a basis for counting various items. On hearing word 'number', all of us immediately think of the familiar decimal number system with its 10 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

Number system refers to the digits, its arrangements, positional value and base of number system.

Introduction to positional and non-positional number system

In positional number system, each digit of a number has its unique positional weight or place value. Examples of positional number system are decimal, binary, octal, hexadecimal etc. In non-positional number system, each digit/symbol of a number has no positional weight or place value. Example of non-positional number system is Roman Number System.

There are basically two number systems

1. Non-positional Number System
2. Positional Number system

1. Non-Positional Number System:

Non positional number systems were used in early days of human civilization. People used to count any things on fingers. When ten fingers were not adequate, stones, pebbles, or sticks were used to indicate the values. In this system the symbols such as

I for 1, II for 2
III for 3 IIII for 4
IIIII for 5 etc....

2. Positional Number System:

The positional number system is one in which the position of a number represents different values depending upon the position they occupy in the number. There are only few symbols are used. And the value of each digit is determined by three considerations.

1. The digit itself
2. The position of the digit in the number
3. The base of the number system.

A number with radix r is represented by a string of digits as below: the general form of all number can be written as a general form:

$$MSB - A_{r-1} - A_{r-2} - A_{r-3} \ldots\ldots\ldots + A_2 + A_1 + A_0 . A_{-1} + A_{-2} \ldots\ldots\ldots + A_{-m+1} - (A_{-m}) - LSB$$

$$(\text{Number}) = \left(\sum_{i=0}^{i=n-1} A_i\right) + \left(\sum_{j=-1}^{j=-m} A_j\right)$$

(Integer Portion) + (Fraction Portion)

Where ($0 < A_i < r$) each symbol for a particular base system.

For r = 10 (decimal number system) A_i will be one of 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The subscript i gives the position of the coefficient and hence the weight riby which the coefficient must be multiplied.

- A_{n-1} is the MSB (Most Significant Digit)
- A_{-m} is the LSB (Least Significant Digit)

A digit/bit having maximum positional weight is called Most Significant Digit (MSD)/Most Significant Bit (MSB) and a digit having minimum positional weight is called Least Significant Digit (LSD)/Least Significant Bit (LSB). In other words, the left most digit or bit is called MSD or MSB and the right most digit or bit is called LSD or LSB.

The characteristics of positional number systems are

- The value of the base determines the total number of different symbols or digits available in the number system and the first digit is always zero (0).
- The maximum value of a single digit is always equal to one less than the value of the base.

Types of Positional Number System

1. Decimal Number System
2. Binary Number System
3. Octal Number System
4. Hexadecimal Number System

1. Decimal Number System

A number system having base 10 is called decimal number system

Following are the characteristics of a decimal number system.

- Uses ten digits 0,1,2,3,4,5,6,7, 8 and 9.
- Called base 10 number system.
- Radix (r) = 10
- Symbols = 0 through r-1 = 0 through 10-1 = {0, 1, 2... 8, 9}
- Each position in a decimal number represents a 0 power of the base (10). Example: 10^0
- Last position in a decimal number represents an x power of the base (10).

2. Binary Number System

Binary number system is one whose base is 2. That means there is only two symbols or digits (0 and 1) are used.

A number system having base 2 is called binary number system

Following are the characteristics of a binary number system.

- Uses two digits, 0 and 1.
- Called base 2 number system
- Radix (r) = 2
- Symbols = 0 through r-1 = 0 through 2-1 = {0, 1}
- Each position in a binary number represents a 0 power of the base (2). Example: 2^0
- Last position in a binary number represents an x power of the base (2). Example: 2^x where x represents the last position - 1.
- .Digits in a binary number are called bits (Binary digits).

3. Octal Number System

Octal number system is one whose base is 8. Eight symbols or digits (0, 1, 2, 3, 4, 5, 6 and 7) are used.

A number system having base 8 is called octal number system

Following are the characteristics of an octal number system.

- Uses eight digits 0,1,2,3,4,5,6 and 7.
- Called base 8 number system.
- Radix (r) = 8
- Symbols = 0 through r-1 = 0 through 8-1 = {0, 1, 2...6, 7}
- Each position in an octal number represents a 0 power of the base (8). Example: 8^0
- Last position in an octal number represents an x power of the base (8). Example: 8^x where x represents the last position - 1.

The decimal equivalent of an octal number 360 (written 360_8) is

4. Hexadecimal Number System

Hexadecimal number system is one with base is 16. That means there is 16 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F) are used. The first ten digits are decimal number and the remaining six digits are denoted by the symbol A, B, C, D, E and F representing the decimal values 10, 11, 12, 13, 14, and 15.

A number system having base 16 is called Hexa-Decimal number system

Following are the characteristics of a hexadecimal number system.

- Uses 10 digits and 6 letters, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.
- Letters represents numbers starting from 10. A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.
- Called base 16 number system.
- Radix (r) = 16

- Symbols = 0 through r-1 = 0 through 10-1 {A, B, C, D, E, F}
- Each position in a hexadecimal number represents a 0 power of the base (16). Example – 16^0
- Last position in a hexadecimal number represents an x power of the base (16) where x represents the last position - 1. Example – 16^x

NUMBER BASE CONVERSION SYSTEM

Table: Four-bit Binary numbers with their Octal values, Decimal values and Hexadecimal values.

Decimal Number (base - 10)	Binary Number (base - 2)	Octal Number (base - 8)	Hexadecimal Number (base - 16)
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

CASE 1: Conversion from base-r system to decimal number system

The base r can be binary (r = 2) or octal (r = 8) or hexadecimal (r = 16) or any other. The general expression is:

$$(\text{Number})_r = A_{n-1}r^{n-1} + A_{n-2}r^{n-2} + A_{n-3}r^{n-3} + \ldots\ldots + A_1 r^1 + A_0 r^0 + A_{-1} r^{-1} + \ldots\ldots + A_{-m} r^{-m}$$

$$(\text{Number})_r = \left(\sum_{i=0}^{i=n-1} A_i r^i\right) + \left(\sum_{j=-m}^{j=-1} A_j r^j\right)$$

(Integer Portion) + (Fraction Portion)

CASE 2: Conversion from decimal to base-r system

The base r can be binary (r = 2) or octal (r = 8) or hexadecimal (r = 16) or any other.

1. Separate the number into integer part and fractional part
2. Divide "decimal integer part" by base r repeatedly until the quotient becomes zero and storing remainders at each step
3. Multiply "decimal fractional part" by r and accumulate the integer digits so obtained
4. Combined the accumulated results and parenthesis, the whole result with subscript r.

CASE 3: Binary to Octal & Hexadecimal and vice versa

A conversion from a binary number system to octal number system and hexadecimal number system plays an important in digital computers.

a. Binary to octal and vice versa
b. Binary to hexadecimal and vice versa

a. Binary to Octal and vice versa

An octal number is represented by at least three bits of binary numbers. So, each digits of an octal number is represented by at least three bits of binary numbers. ($8 = 2^3$)

While converting a binary number to an octal number system a binary number are grouped into the multiple of three digits to represent its equivalent octal number. If there are integer part and a fractional part in the given binary number then the integer parts of a binary number is grouped into three bits from right side to left side and if the integer part of a binary number is not a multiple of three bits then '0' is append in-front of the number. The fractional part of a binary number is grouped from left side to right side and if it is not a multiple of three bits then '0' is append at the end of a fractional part. ($2^3 = 8$)

b. Binary to Hexadecimal and vice versa

A hexadecimal number is represented by at least four bits of binary numbers. So, each digits of a hexadecimal number is represented by at least four bits of binary numbers. ($16 = 2^4$)

While converting a binary number to a hexadecimal number system a binary number are grouped into the multiple of four digits to represent its equivalent hexadecimal number. If there are integer part and a fractional part in the given binary number then the integer parts of a binary number is grouped into four bits from right side to left side and if the integer part of a binary number is not a multiple of four bits then '0' is append in-front of the number. The fractional part of a binary number is grouped from left side to right side and if it is not a multiple of four bits then '0' is append at the end of a fractional part. ($2^4 = 16$)

COMPLEMENT

Complements are used in the digital computers in order to simplify the subtraction operation and for the logical manipulations. For each radix-r system (radix r represents base of number system) there are two types of complements. There are two types of complements for each base-r system.

1. The r's complement
2. The (r-1)'s complement

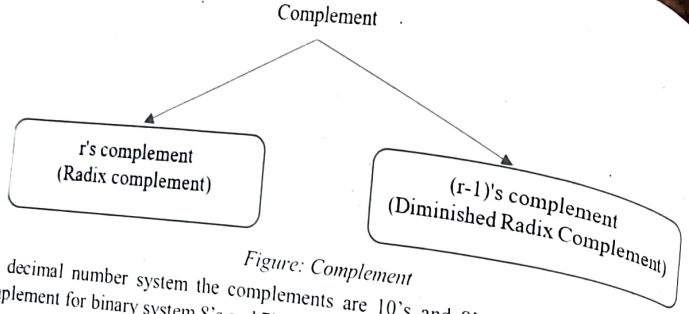


Figure: Complement

For decimal number system the complements are 10's and 9's complement. 2's and 1's complement for binary system.8's and 7's complement for octal number system and so on.

***Table:** r's and (r-1)'s Complement*

S.N.	Complement	Description
1	Radix Complement	The radix complement is referred to as the r's complement
2	Diminished Radix Complement	The diminished radix complement is referred to as the (r-1)'s complement

1. **The r's complement**
   - Given the positive number N in base r with integer part of n digits, the r's complement of N is defined as $r^n - N$ for $N \neq 0$ and 0 for $N = 0$;
     - r's Complement (radix complement)
     - r's complement of a number N is defined as $r^n - N$

   Where N is the given number
     - r is the base of number system
     - n is the number of digits in the given number
   - To get the r's complement fast, add 1 to the low-order digit of its (r-1)'s complement.

   Example:
   - 10's complement of 835_{10} is $164_{10} + 1 = 165_{10}$
   - 2's complement of 1010_2 is $0101_2 + 1 = 0110_2$
2. **The (r-1)'s complement**
   - Given a positive number N in base r with an integer part of n digits and a fraction part of m digits, the (r-1)'s complement of N is defined as $(r^n - r^{-m} - N)$
   - (r-1)'s Complement (diminished radix complement)
   - (r-1)'s complement of a number N is defined as $(r^n - 1) - N$
     - Where N is the given number
     - r is the base of number system
     - n is the number of digits in the given number
   - To get the (r-1)'s complement fast, subtract each digit of a number from (r-1).

Example:

- 9's complement of (835)10 is (164)10 (Rule: $(10^n - 1) - N$)
- 1's complement of $(1010)_2$ is $(0101)_2$ (bit by bit complement operation)

Subtraction with complements

When subtraction is implemented with digital hardware, this method is found to be less efficient than the method that uses complements. Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation. Using complements, all the arithmetic operations can be performed in the form of addition. There are two types of compliments: (r – 1)'s complement and r's complement where r is base of a number system. So, for binary number system, r = 2 hence, it has two complements- 1's complement and 2's complement. Likewise, for decimal number system, r = 10 hence it also has two complements-9's complement and 10's complement.

The subtraction of two n-digit unsigned numbers M - N in base-r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. This performs
   $= M + (r^n - N)$
   $= M - N + r^n$.
2. If M >= N, the sum will produce an end carry, r^n, which is discarded; what is left is the result M - N.
3. If M <N, the sum does not produce an end carry and is equal to r^n- (N - M), which is the r's complement of (N - M). To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front it.

Things to Remember:

- A system which process a continuous time variant signal is called analog system and a system which process a discrete time variant signal is called digital system.
- The digital system contains devices such as logic gates, flip-flop, shift register and counters etc.
- A digital computer is an electronics device that read the data from the environment through an input device, store them in a memory, process them according to the command given by the user (programmer) and display the result or output to the environment through an output unit. A digital computer contains
  1. Input Unit
  2. Output Unit
  3. Storage Unit
  4. Central Processing Unit
- Number system is a basis for counting various items or data. The most popular and daily used number system is decimal number system. Number system is mainly classified in two type
  1. Non-Positional Number System

2. Positional Number System

- Non positional number system were used in early days of civilization. In positional number system the position of a number doesn't matter. For example, counting items on fingers, pebbles, marking a line on the wall etc.
- Positional number system is the most popular and daily used number system. In positional number system the position of a number represents different values depending upon the position they occupy in the number. Positional numbers are classified as follow:
  1. Binary Number System
  2. Octal Number System
  3. Decimal Number System
  4. Hexadecimal Number System
- Binary number system is one whose base is 2. i.e only two symbols are used 0 and 1.
- Octal number system is one whose base is 8. i.e only eight symbols are used 0, 1, 2, 3, 4, 5, 6 and 7.
- Decimal number system is one whose base is 10. i.e only ten symbols are used 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.
- Hexadecimal number system is one whose base is 16. i.e only sixteen symbols are used 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F..
- Note that the positional number system the symbols always begins with 0 and end with one less than the base system.
- A binary number is a weighted number in which the weight of each whole number digits is a positive power of two and the weight of each fractional digit is a negative power of two.
- Various system uses different number system, human being uses decimal number system in their day to day life. So, it is necessary to convert a number from one base system to another. While converting a number system following situations are occur"
  1. Conversion from base-r system to decimal system
  2. Conversion from decimal system to base-r system
  3. Conversion from binary system to Octal system and Hexadecimal system and vice versa
- A binary number can be converted into decimal number by summing decimal values of the weights of all the 1s in the binary number.
- A decimal whole number can be converted to binary by using the sum- of –weights or the repeated division by -2 method.
- A decimal fraction can be converted to binary by using the sum –of- weights or the repeated multiplication-by-2-method.
- One hexadecimal digit represents a 4-bit binary number, and its primary usefulness is in simplifying bit patterns and making them easier to read.

- Complement in digital computer are used for subtraction operation. Binary subtraction can be accomplished with addition by using the 1's or 2's complement method.
- There are two types of complements for each base-r system
- The r's complement
- The (r-1)'s complement
- The 1's complement of a binary number is derived by changing 1s to 0s and 0s to 1s.
- The 2's complement of a binary number can be derived by adding 1 to the 1's complement.
- The basic rules for binary addition are as follows:
  - $0 + 0 = 0$
  - $0 + 1 = 1$
  - $1 + 0 = 1$
  - $1 + 1 = 10$
- The basic rules for binary subtraction are as follows:
  - $0 - 0 = 0$
  - $1 - 1 = 0$
  - $1 - 0 = 1$
  - $10 - 1 = 0 - 1$ with a borrow of 1.
- Positive binary number is represented by a 0-sign bit and A negative binary number is represented by a 1 sign bit.
- When numbers, alphabets or words are represented by a specific group of symbols, we can say that they are encoded. The group of symbols used to encode them is called codes. The codes are classified as follows
  1. Weighted Codes
  2. Non-weighted Codes
  3. Reflective Codes
  4. Sequential Codes
  5. Alphanumeric Codes
  6. Error detecting and correcting Codes
- In weighted codes, each digit position of the number represents a specific weight. For example, in decimal code,
- In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code.
- Excess-3 code is a modified form of a BCD number. The excess-3 code can be derived from the natural BCD code by adding 3 to each coded number.
- Gray code is a non-weighted code and is a special case of unit-distance code.
- When the digital information in the binary form is transmitted from one circuit or system to another circuit or system an error may occur

- Error detection methods are parity checking, checksum error detection and cyclic redundancy check.
- A parity bit is used to detect an error in a code
  1. **Even Parity:** In even parity the added parity bit will make the total number of 1s an even amount.
  2. **Odd Parity:** In odd parity the added parity bit will make the total number of 1s an odd amount.
- The hamming code is the single bit error correction method using redundant bits.
- The CRC (Cyclic Redundancy Check) is based on polynomial division using modulo-2.
- Alphanumeric codes are called character codes and binary codes are used to represent alphanumeric data.
- ASCII (American Standard Code for Information Interchange) code is very popular code used in all personal computers and workstation. It is a standard binary code for alphanumeric character.
- Unicode typically uses 16 bits to store a character. This allows for the unique representation of characters in all languages and platforms.
- EBCDIC (Extended Binary Coded Decimal Interchange Code) is an eight bit character encoding used mainly on IBM mainframe and IBM mid-range computer operating systems.
- Integrated Circuit (IC) is a silicon chip contains the electrical components such as transistor, diodes, resistors and capacitor. On the basis of functions ICs are classified as
  1. Analog or Linear ICs
  2. Digital or Logic ICs
  3. Hybrid ICs
- In order to protect ICs from external environment and to provide mechanical protection, various forms of encapsulation are used for integrated circuit. On the basis of the packing of integrated circuits it is classified into two.
  1. Flat Package
  2. Dual in Line

  In 1965 Gorden Moore, who was cofound Intel Corporation in 1968 said that the components fabricated into an Integrated Circuit (IC) is double in every 18 months.

Logic Levels

- In digital logic, a logic level refers to the specific voltage or current used to represent binary values of 0 and 1. In other words, it is the electrical representation of a binary digit (bit).
- The most commonly used logic levels in digital electronics are TTL (Transistor-Transistor Logic) and CMOS (Complementary Metal-Oxide-Semiconductor).
- In TTL logic, a high logic level is typically represented by a voltage of around 2-5 volts, while a low logic level is represented by a voltage of around 0 volts.

- In CMOS logic, a high logic level is typically represented by a voltage close to the power supply voltage (e.g., 5 volts), while a low logic level is represented by a voltage close to 0 volts.

Logic Gates

A logic gate is an elementary building block of a digital circuit. Most of the logic gates has two inputs and one output. Each input and outputs are represented by binary values (0 for "false" and 1 for "true").

The electronic gate is a circuit that is able to operate on a number of binary inputs in order to perform a particular logical function

In most logic gates, the low state is approximately zero volts (0 V), while the high state is approximately five volts positive (+5 V).

Logic gates are the basic element that makes up a digital system. The types of gates available are the NOT, AND, OR, NAND, NOR, exclusive-OR, and the exclusive -NOR. Expect for the exclusive- NOR gate, are they available in monolithic integrated circuit form.

1. **Basic Logic Gates**
   a. OR Gate
   b. AND Gate
   c. NOT Gate
2. **Universal Logic Gates**
   a. NAND Gate
   b. NOR Gate
3. **Derived Logic Gates/ Arithmetic Gate**
   a. XOR Gate
   b. EX-Nor Gate

Basic Logic Gates

1. **OR Gate**
   - The OR gate is a basic logic gate whose output is **"true" or "high" if any one of the inputs are "true" or "high" otherwise the output is "false" or "low". I.e. the output is "false" or "low" if all the inputs are "false" or "low".**

Things to Remember

For a 2-input OR gate, output Y is HIGH when either input A or input B is HIGH, or when both A and B are HIGH; Y is low only when both A and B are low.

- The OR gate performs logical addition, more commonly known as the OR function. An OR gate has two or more inputs and one output, as indicated by the standard logic symbol in Figure 2.20 where OR gates with two and four inputs are illustrated.

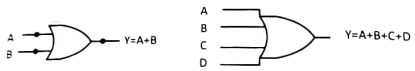


Figure (a) Two inputs OR gate *Figure (b) Four inputs OR gate*

Table: *Truth Table of OR gate*

Inputs		Output
A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

2. AND Gate

- The AND gate is one of the basics gates that can be combined to form any logic function. An AND gate can have two or more inputs and performs what is known as logical multiplication.
- An AND gate procures a HIGH output only when all of the inputs are HIGH. When anyone of the input is LOW, the output is LOW.

For 2 input AND Gate, output Y is high only when inputs A and B are HIGH: Y is Low when either A or B is LOW, or when both A and B are LOW.

- The AND gate performs logical multiplication, more commonly known as the AND function. The AND gate may have two or more inputs and a single output, as indicated by the standard logic symbols shown in the Figure.

Figure: Two Input AND Gate

The Figure illustrates a two-input AND gate with all four possibilities of input combinations, and the resulting output for each.

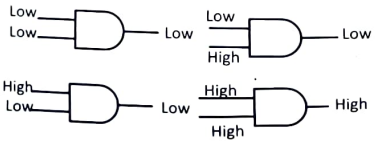


Figure: Four possible inputs for two input AND gate and resulting outputs

The total number of possible combinations of binary input to a gate is determined by the following formula:

$N = 2^n$

Where N is the number of possible input combination and n is the total number of input variables. To illustrate,

For 2 input variables, $N = 2^2 = 4$ Combinations

For 3 input variables, $N = 2^3 = 8$ Combinations

For 4 input variables, $N = 2^4 = 16$ Combinations

Table: *Truth table for 2 input AND gate*

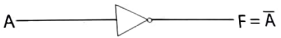
Inputs		Output
A	B	Y = A · B
0	0	0
0	1	0
1	0	0
1	1	1

3. NOT Gate/ Inverter

- The NOT gate or the inverter is a basic logic gate whose output is just inverse of the input. NOT gate have a single input and a single output. The output is "true" if the input is "false" and the output is "false" if the input is "true".

When the input is LOW, the output is HIGH; when the input is HIGH, the output is LOW, thereby producing an inverted output pulse.

- The Figure 2.28 shows the symbol for the inverter.



The bubble [o] appearing on the output is the negation (inversion) indicator.

Figure: NOT Gate

Table: *Truth Table of NOT gate*

Inputs	Outputs
A	$F = \overline{A}$
0	1
1	0

An Application:

Following Figure shows a circuit for producing the 1's complement of 4-bit binary number. The bits of the binary number are applied to the inverter inputs and the 1's complement of the number appears on the outputs.

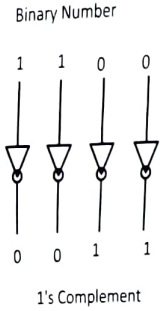


Figure: Example of a 1's complement circuit using inverters

Universal Gate

- **A universal gate is a gate which can implement any Boolean function without using any other types of gates. NAND and NOR gates are known as universal gates.**
- Hence any Boolean function can be implemented using only NAND and NOR gate.
- NAND gate and NOR gates are the universal gates.
- The NAND gate and NOR gates are easier to fabricate and economical and can be used in all logic circuit.
- By connecting them together in various combinations the three basic gate types of AND, OR and NOT function can be formed using these gates.

NAND and NOR gates are called Universal Gate because all the other basic gates (NOT, AND and OR gate) can be constructed by using NOR gate only and by using NAND gate only.

1. NAND Gate

The NAND gate is a universal gate whose output is "true" if any one or all of the inputs to the gate is "false". The output of NAND gate is "false" if all the inputs to the gate is "true".

The Boolean function can be implemented using only NAND gates. By using NAND gate we can derive basic OR gate, AND gate and NOT gate.

Things to Remember

For a 2-input NAND gate, output F is LOW only when inputs A and B are HIGH: F is *HIGH* when either A or B is LOW, or when both A and B are LOW.

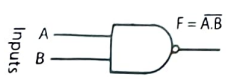


Figure 2-input NAND Gate

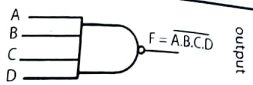


Figure: (b) 4-input NAND Gate

The NAND gate is the combination of two gates (One AND gate and an inverter). The inputs are first ANDed and then it is inverted as shown in below.

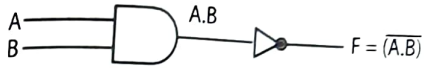


***Table:** Truth Table of NAND gate*

Inputs		Outputs
A	B	Z = $\overline{(A \cdot B)}$
0	0	1
0	1	1
1	0	1
1	1	0

2. NOR Gate

For a 2-input NOR gate, output F is LOW when either input A or input B is HIGH, or when both A and B are HIGH: F is HIGH only when both A and B are LOW.

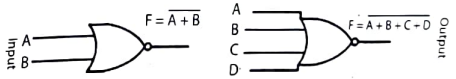


Figure: (a) 2-input NOR Gate *Figure: (b) 4-input NOR Gate*

The NOR gate is the combination of two gates (One OR gate and an inverter). The inputs are first ORed and then it is inverted as shown in below.

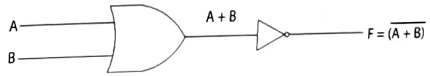


Figure: AND gate with Inverter

***Table:** Truth Table*

Inputs		Output
A	B	F = ($\overline{X + Y}$)
0	0	1
0	1	0
1	0	0
1	1	0

Derive Gate/ Arithmetic Gate/Exclusive Gate

1. XOR Gate
2. XNOR Gate

1. XOR Gate

The XOR gate is a derived gate in which the output is "true" if either, but not both, of the inputs are "true". The output is "false" if both inputs are "false" or if both inputs are "true".

For an exclusive -OR gate, output F is HIGH when input A is LOW and input B is HIGH, or when input a is HIGH and input B is LOW; F is LOW when A and B are both HIGH or both LOW.

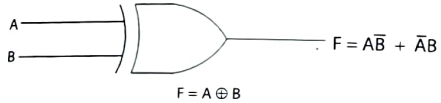


Figure: Logic symbol of 2 input Ex-OR Gate

***Table:** Truth Table of 2 input XOR gate*

Inputs		Output
A	B	$F = A\overline{B} + \overline{A}B$
0	0	0
0	1	1
1	0	1
1	1	0

The Ex-OR gate can be constructed by using basic gates as shown in logic circuit below.

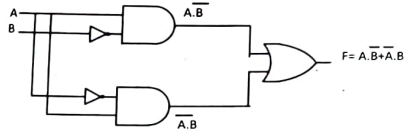


Figure: Circuit diagram of 2 input XOR

2. XNOR Gate

The XNOR gate is a combination of XOR gate followed by an inverter. Its output is "true" if the inputs are the same and "false" if the inputs are different.

For an exclusive-NOR gate, output F is LOW when input a is LOW and input B is HIGH, or when A is HIGH and B is LOW; F is HIGH when A and B are both HIGH or both LOW.

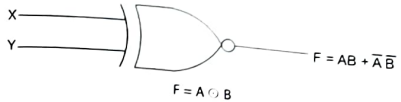


Figure: Logic Symbol of 2 input EXNOR gate

***Table:** Truth Table of XNOR gate*

Inputs		Output
A	B	$F = AB + \overline{A}\ \overline{B}$
0	0	1
0	1	0
1	0	0
1	1	1

The Ex-NOR gate can be constructed by using the basic gates only as shown in logic circuit below.

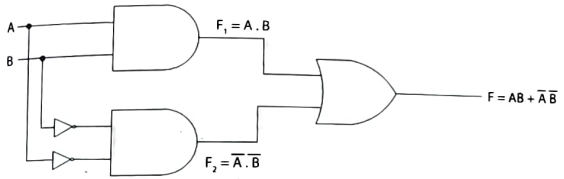


Figure: Circuit diagram of 2 input EXNOR gate

Boolean algebra:

Boolean Algebra is a mathematical system that uses binary variables (i.e., variables that can have only two values: 0 or 1) and logical operations (such as AND, OR, NOT, etc.) to describe and analyze logic circuits. It was invented by the mathematician George Boole in the 19th century and is used in many areas of computer science, including computer hardware design, digital electronics, and computer programming.

In Boolean Algebra, binary variables are used to represent logical values (e.g., true or false, on or off, etc.). The logical operations in Boolean Algebra are used to manipulate these binary variables and to build more complex expressions. The basic operations in Boolean Algebra are AND, OR, and NOT, which can be used to build more complex expressions using De Morgan's laws and other rules.

A Boolean function is an algebraic expression formed with Boolean variables (having values true or 1 and false or 0) and the logical operators (i.e. OR, AND, and NOT).

K-map is in fact a visual diagram of representing all possible ways a Boolean function may be expressed.

Boolean Functions: Terminology

$F(a,b,c) = \bar{a}bc + ab\bar{c} + a\,b + c$

Variable

- Represents a value (0 or 1). Three variables are a, b and c.

Prime Implicant:

- A prime implicant is product or sum term obtain by combining the maximum possible number of adjacent squares (minterm/maxterm) in the map.

Essential Prime Implicant:

- If a minterm max term in a sequence is covered by only 1 prime implicant is said to be essential prime implicant.

Literal

- Appearance of variable either in complemented form or in an un complemented form
- Nine literals are; $\bar{a}$,b,c, a b,$\bar{c}$,a, b ,c

Product Term:

- It is a term where literals are ANDed.
- **Example:** $\bar{a}\,\bar{b}$, a c, abc,

Minterms:

- A Minterm is a special product (ANDing terms) of literals, in which each input variable appears exactly once. A function with n input variables has 2^n Minterms.

  For example a three variable function like F(X, Y, Z) has 8 Minterms ($2^3 = 8$). They are

  $\bar{A}\bar{B}\bar{C}$, $A\bar{B}\bar{C}$, $\bar{A}B\bar{C}$, $\bar{A}\bar{B}C$, $\bar{A}BC$, $A\bar{B}C$, $AB\bar{C}$, ABC

  And each minterm is true for exactly one combination of inputs

Sum Term: A term where literals are ORed.

- Example: $\bar{x} + \bar{y}$, $x + z$, $x + y + z$

Maxterms:

- A Maxterm is a special sum (ORing terms) of literals, in which each input variable **appears exactly once. A function with n input variables has maximum of 2^n Maxterms.**

  For example, a three-variable function like F (X, Y, Z) has 8 Maxterms ($2^3 = 8$). They are

  $\bar{A}+\bar{B}+\bar{C}$, $A+\bar{B}+\bar{C}$, $\bar{A}+B+\bar{C}$, $\bar{A}+\bar{B}+C$, $\bar{A}+B+C$, $A+\bar{B}+C$, $A+B+\bar{C}$, $A+B+C$

  And each maxterm is false for exactly one combination of inputs

Table: Minterms and Maxterms for 3 Binary variables with their symbolic shorthand

Inputs			Minterms		Maxterms	
A	B	C	Terms	Designation	Terms	Designation
0	0	0	$\bar{A}\,\bar{B}\,\bar{C}$	m_0	$A+B+C$	M_0
0	0	1	$\bar{A}\,\bar{B}\,C$	m_1	$A+B+\bar{C}$	M_1
0	1	0	$\bar{A}\,B\,\bar{C}$	m_2	$A+\bar{B}+C$	M_2
0	1	1	$\bar{A}\,BC$	m_3	$A+\bar{B}+\bar{C}$	M_3
1	0	0	$A\,\bar{B}\,\bar{C}$	m_4	$\bar{A}+B+C$	M_4
1	0	1	$A\bar{B}C$	m_5	$\bar{A}+B+\bar{C}$	M_5
1	1	0	$\bar{A}\,\bar{B}\,\bar{C}$	m_6	$\bar{A}+\bar{B}+C$	M_6
1	1	1	ABC	m_7	$\bar{A}+\bar{B}+\bar{C}$	M_7

Sum of Product/Sum of Minterms Method

Boolean functions can be expressed as a sum of Minterms or product of Maxterms and are said to be in canonical form. In canonical form every variable appears in every term. An arbitrary logic function can be expressed in following form.

1. Sum of Products (SOP)
2. Product of the Sums (POS)

Sum of Product (SOP)/Sum of Minterms

For n binary input variables there will be 2^n distinct Minterms. The Minterms whose sum defines the Boolean function are those that give the 1's of the function in a truth table. It is sometimes convenient to express the Boolean function in its Sum of Minterms form. If the terms are not in SOP form it can be made so by first expanding the expression into a sum of AND terms. Each term is then inspected to see if it contains all the variables. If any one of the variables is missed then it is ANDed with an expression such as $A + \bar{A}$, where A is one of the missing variables.

Truth Table to Karnaugh Map

A Karnaugh map (K-map) is a graphical tool used to simplify Boolean algebra expressions. It is used to minimize the number of logic gates required to implement a digital circuit, and to simplify the expression of a logical function. The K-map consists of a grid of cells, each representing a possible combination of inputs to a Boolean function. A truth table can be converted into a Karnaugh map (K-map) to simplify the Boolean expression for a logical function. The process of converting a truth table into a K-map involves the following steps:

- Draw the K-map grid
- Fill in the K-map cells

- Group adjacent cells with the same value
- Simplify the expression

Maurice Karnaugh developed Karnaugh maps which are used for minimizing Boolean expression for few variables. These maps are also known as Veitch diagram.

The Boolean expression can be minimized by two different ways:

1. Boolean algebra method
2. Map method

1. Boolean algebra Method

In Boolean algebra method we need better understanding of Boolean laws, rules and theorems. During the process of simplification, we have to predict each successive step. For these reasons, we can never be absolutely certain that an expression simplified by Boolean algebra alone is the simplest possible expression.

In Boolean algebra method we can use grouping, Multiplication by redundant variables and Application of De-Morgan's theorem as given by:

a. Grouping

$= A + AB + BC$

$= A(1 + B) + BC$

$= A + BC \quad [\because 1 + B = 1]$

b. Multiplication by redundant variables

Multiply by terms of the form $(A + \overline{A})$ doesn't alter the logic

Such multiplications by a variable missing from a term may enable minimization

$AB + A\overline{C} + BC$

$= AB(C + \overline{C}) + A\overline{C} + BC$

$= ABC + AB\overline{C} + A\overline{C} + BC$

$= BC(A + 1) + A\overline{C}\ (B + 1)$

$= BC + A\overline{C} \qquad [\because A + 1 = 1]$

c. Application of De-Morgan's theorem

Expressions containing several inversions stacked one upon the other often are simplified by using De-Morgan's law which un-wraps multiple inversions:

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

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

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

$= \overline{AB}\,CD$

2. Map method

The map method gives us the systematic approach for simplifying a Boolean expression. The map method, first proposed by Veitch and modified by Karnaugh, hence it is known as the Veitch diagram of the Karnaugh map. The map method is regarded as a pictorial form of a truth table.

Karnaugh Map is a graphical representation of the truth table of the given expression.

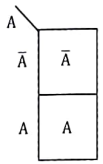
Karnaugh Map

a. For one variable and two variables
b. Three variables
c. Four variables

a. For one variable and two variables

The basic method is a graphical chart known as Karnaugh map (K-map). It contains boxes called cells. Each cell represents one of the 2^n possible products that can be formed from n variables. Thus a 1 variable map contains 2^1=2, a 2 variables map contains $2^2 = 4$ cells, a 3 variables map contains $2^3 = 8$ cells, and a 4 variables map contains $2^4 = 16$ cells and so forth.

For One variable

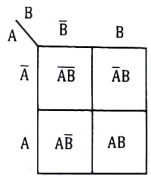


1-variable map is an array of two cells. The value of a binary variable is along the left side as shown in figure above.

(2 cells)

Figure (a): K-map for one variable

For Two Variables



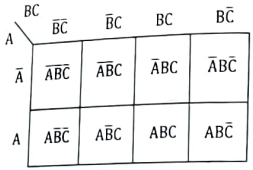
2-variable map is an array of four cells. The value of binary variable A is along the left side and the value of binary variable B is at the top as shown in above figure.

(4 cells)

Figure (b): K-map for two variable

b. For Three Variables

The 3-variable Karnaugh map is an array of eight cells as shown below. Let A, B and C are three binary variables. The binary variable A is along left side and the binary variables B and C are along the top of the map. The cell in the upper left corner has a binary value of 000 and the cell in the lower right corner has a binary value of 110. The product terms that is represented by each cell in the Karnaugh map.

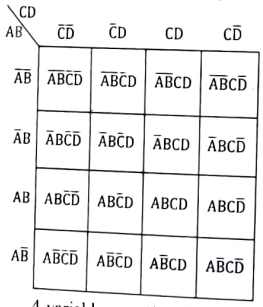


3-variable map (8 cells)

Figure: K-map for three variable

c. For Four Variables

The 4-variable Karnaugh map is an array of sixteen cells as shown in figure below. Let A, B, C and D are four binary variables. The binary values of A and B are along left side and the values of C and D are across the top of the map. The value of a given cell is the binary values of A and B at the left in the same row combined with the binary values of C and D at the top in the same column. For example, the cell in the upper left corner has a binary value of 0000, the cell in the upper right corner has a binary value of 0010, and the cell in the lower right corner has a binary value of 1010 and so on.



4-variable map (16 cells)

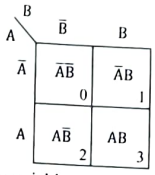
Figure: K-map for four variable

For One variable

1-variable map
(2 cells)

Figure (a): K-map for one variable

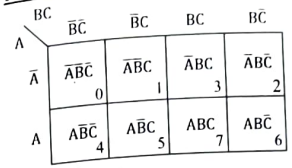
For Two Variables



2-variable map
(4 cells)

Figure (b): K-map for two variables

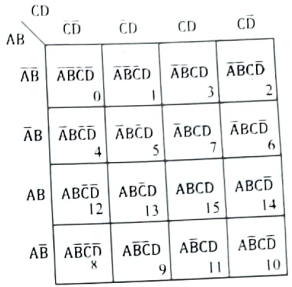
For Three Variables



3-variable map
(8 cells)

Figure. (c): K-map for three variable

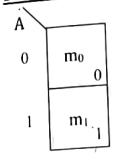
For Four Variables



4-variable map
(16 cells)

Figure (d): K-map for four variable

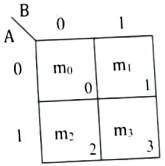
For One variable



1-variable map
(2 cells)

Figure (a): K-map for one variable

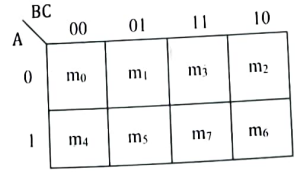
For Two Variables



2-variable map
(4 cells)

Figure (b): K-map for two variable

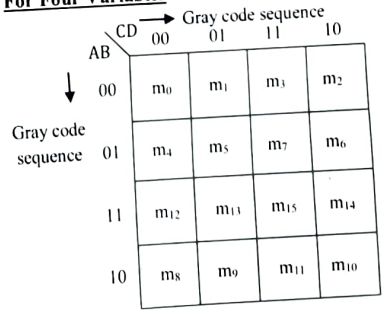
For Three Variables



3-variable map
(8 cells)

Figure (c): K-map for three variable

For Four Variables



4-variable map
(16 cells)

Figure (d): K-map for four variable

Things to Remember

- Boolean algebra was developed by the English mathematician George Boole. His work was based on the analysis of logic and is contained is his book named "An investigation of the laws of thought on which are founded the Mathematical Theories of Logic and Probabilities", published in 1854.
- Commutative laws: $A + B = B + A$

  $AB = BA$
- Associative laws: $A + (B + C) = (A + B) + C$

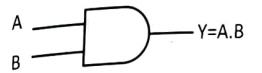
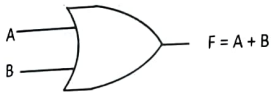
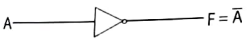
  $A(BC) = (AB)C$
- Distributive law: $A(B + C) = AB + AC$
- Boolean rules:

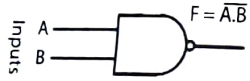
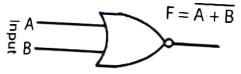
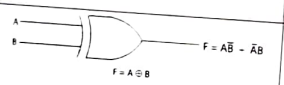
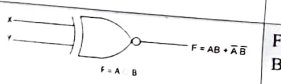
  1. $A + 0 = A$
  2. $A + 1 = 1$
  3. $A.0 = 0$
  4. $A.1 = A$
  5. $A + A = A$
  6. $A + \overline{A} = 1$
  7. $A.A = A$
  8. $A.\overline{A} = 0$
  9. $\overline{\overline{A}} = A$
  10. $A + AB = A$
  11. $A + \overline{A}B = A + B$
  12. $(A + B)(A + C) = A + BC$
- DeMorgan's theorems:
  1. The complement of a product is equal to the sum of the complements of the terms in the product.

     $\overline{XY} = \overline{X} + \overline{Y}$
  2. The complement of a sum is equal to the product of the complements of the terms in the sum.

     $\overline{X + Y} = \overline{XY}$
- According to Principle of Duality, dual of a Boolean expression can be obtained by replacing AND (.) with OR (+) and vice versa, 1 with 0 and vice versa keeping variables and complements and variables are unchanged.
- Logic gates are one of the fundamental building blocks of digital systems.
- The inverter output is the complement of the input.
- When the input is LOW, the output is high; when the input is HIGH, the output is LOW, thereby producing an inverted output pulse.
- The AND gate output is HIGH only if all the inputs are HIGH.
- For a 2-input AND gate, output X is HIGH only when inputs A and B are HIGHT; X is LOW when either A or B is LOW, or when both A and B are LOW.
- The OR gate output is HIGH if any of the input is HIGH.

- The NAND gate output is LOW only if all the inputs are HIGH.
- The NAND gate can be viewed as a negative-OR whose output is HIGH when any input is LOW.
- The NOR gate output is LOW is any of the inputs is HIGH.
- The NOR can be viewed as a negative-AND whose output is HIGH only if all the inputs are LOW.
- The exclusive-OR gate output is HIGH when the inputs are not the same.
- The exclusive-NOR gate is LOW when the inputs are not the same.
- The name, Graphic symbol, Algebraic function and Truth table of digital logic gates are given below:

Name	Graphic symbol	Algebraic function	Truth table
AND		$Y = A.B$	Inputs: A, B; Output: $Y = A \cdot B$ 0 0 0 0 1 0 1 0 0 1 1 1
OR		$Y = A + B$	Inputs: A, B; Output: $Y = A + B$ 0 0 0 0 1 1 1 0 1 1 1 1
Inverter		$F = \overline{A}$	Inputs: A; Outputs: $F = \overline{A}$ 0 1 1 0

Buffer		F = A	Inputs: A; Outputs: F = A 0 → 0 1 → 1
NAND		F = $\overline{A.B}$	Inputs: A, B; Outputs: F=$\overline{A.B}$ 0 0 → 1 0 1 → 1 1 0 → 1 1 1 → 0
NOR		F = $\overline{A + B}$	Inputs: A, B; Output: F = $(\overline{X + Y})$ 0 0 → 1 0 1 → 0 1 0 → 0 1 1 → 0
Exclusive-OR (XOR)		F = A ⊕ B	Inputs: A, B; Output: F = A $\overline{B}$ + $\overline{A}$ B 0 0 → 0 0 1 → 1 1 0 → 1 1 1 → 0
Exclusive-NOR Or equivalence		F = A ⊙ B	Inputs: A, B; Output: F = AB + $\overline{A}$ $\overline{B}$ 0 0 → 1 0 1 → 0 1 0 → 0 1 1 → 1

2.2 COMBINATIONAL AND ARITHMETIC CIRCUITS

A combinational logic is a circuit consisting of logic gates whose output at any time is determined by the present combinations of inputs. That means in combinational circuit the current output is not determined by the previous input and previous output, so there is no feedback and memory in a combinational circuit.

Combinational logic circuit consist of logic gate whose output at any instant of time are determined by the present combination of input. The basic building block of combinational logic circuit is gate.

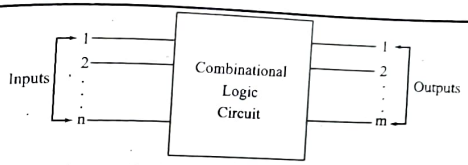


Figure: Block diagram of Combinational Circuit

Design of a Combinational Circuit

The design of a Combinational Circuit starts from the verbal communications and verbal outlines and ends with a logic circuit diagram. The procedure involves the following steps:

1. *State the given problem completely and exactly*
2. *Interpret the problem and determine the available input variables and required output variables.*
3. *Assign a letter symbol to each input and output variables*
4. *Design the truth table, which defines the required relations between inputs and outputs.*
5. *Obtain the simplified Boolean expressions for each output*
6. *Draw the logic circuit diagram to implement the Boolean expression*

The practical design method would have to consider such constrains as:

1. *Minimum **number of gates***
2. *Minimum **number of inputs** to a gate*
3. *Minimum **propagation time** of the signal through the circuit*
4. *Minimum **number of interconnections***
5. *Limitations of the **driving capabilities** of each gate*

Adder

The most common operation performed operation in a digital computer is arithmetic operation. The common arithmetic operation is addition of two binary digits. The simple addition of two digit 0 and 1 consists of:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

The binary adder can be classified into two types.

1. Half Adder
2. Full Adder

1. Half Adder

Half adder is a combinational circuit which is used to add two 1-bit numbers X and Y the input variables designate the augends and addend bits and the output variables produce the sum and carry. The output variables are **S** for **Sum** and **C** for **Carry**.

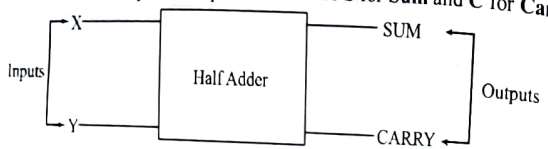


Figure: Block diagram of Half Adder

The half adder accepts two binary digits on its inputs and produces two binary digits on its outputs. – A **sum** bit and a **carry** bit.

The truth table for half adder is shown below.

***Table:** The truth table for Half Adder*

Inputs		Outputs	
Input X	Input Y	Carry C	Sum S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Sum (S) $= \overline{X}\,Y + X\,\overline{Y}$

$= X \oplus Y$

Carry (C) $= XY$

2. Full Adder

Full adder is a combinational circuit which is used to add three 1-bit numbers X, Y and Z. The full adder has three input lines and two output lines. The input variables are denoted by X, Y and Z similarly the output variables are denoted by S for SUM and C for CARRY.

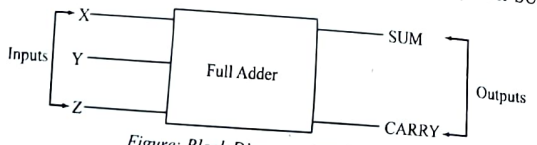


Figure: Block Diagram of Full Adder

The full adder accepts three binary digits on its inputs and produces two binary digits on its outputs. – A **sum** bit and a **carry** bit.

***Table:** The truth table for Full Adder*

Inputs			Outputs	
Input X	Input Y	Input Z	Carry C	Sum S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The Boolean function for full –adder from the about truth table is shown below.

Sum (S) $= X\,\overline{Y}Z + \overline{X}Y\,\overline{Z} + X\,\overline{Y}\,\overline{Z} + XYZ$

$= X\,\overline{Y}Z + XYZ + \overline{X}Y\,\overline{Z} + X\,\overline{Y}\,\overline{Z}$

$= Z(X\,\overline{Y} + XY) + \overline{Z}(\overline{X}Y + X\,\overline{Y})$

$= Z(X \odot Y) + \overline{Z}(X \oplus Y)$

$= Z(X \overline{\oplus} Y) + \overline{Z}(X \oplus Y)$

Let, P $= X \oplus Y$

$= Z\,\overline{P} + \overline{Z}P$

$= Z \oplus P$

Replacing the value of P we get

SUM (S) $= Z \oplus X \oplus Y$

$= X \oplus Y \oplus Z$

$\therefore$ SUM (S) $= X \oplus Y \oplus Z$

Carry (C) $= XY + XZ + YZ$

Sum (S) $= \overline{X}\,\overline{Y}Z + \overline{X}Y\,\overline{Z} + X\,\overline{Y}\,\overline{Z} + XYZ$

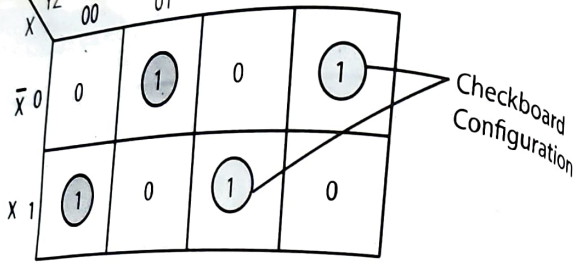


Figure (a): K map for Sum

Carry (C) = XY + XZ + YZ

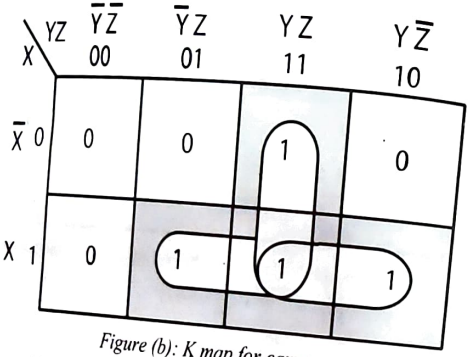


Figure (b): K map for carry

Figure: K map for Sum and Carry of Full Adder

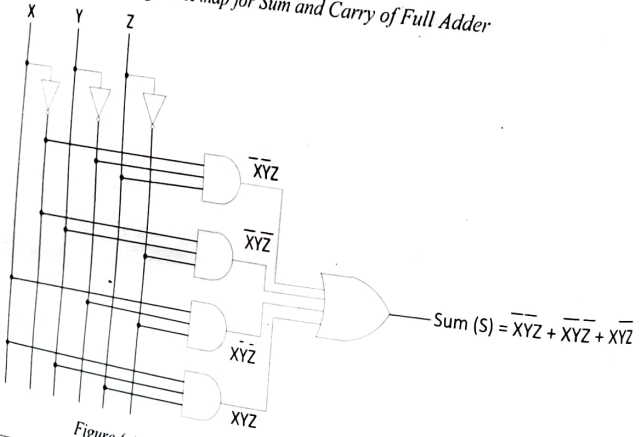


Figure (a): Logic Circuit diagram for Sum in Full Adder

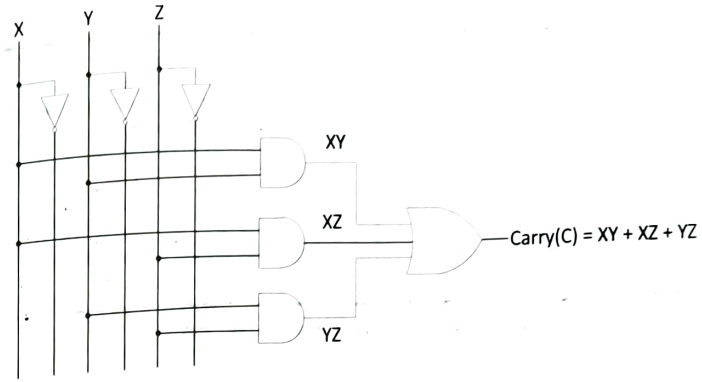


Figure (b): Logic Circuit diagram for Carry in Full Adder

Subtractor

The subtraction of two binary numbers is accomplished by taking the complement of the subtrahend and adding it to the minuend. If the minuend bit is smaller than the subtrahend bit, a bit is borrowed from the next higher significant position.

1. **Half Subtractor**

Half subtractor is a combinational logic circuit that subtracts two number of one bit and produces their difference. There are two input variable let say X and Y of a single bit and there will be two output variables B for borrow from higher significant bit and D for difference.

Let Y is subtracted from X. i.e. X – Y then

0 – 0 = 0
0 – 1 = 1
1 – 0 = 1
1 – 1 = 0

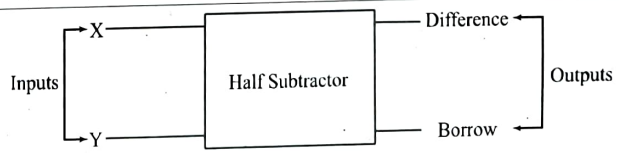


Figure: Block diagram of Half Subtractor

The half subtractor accepts two binary digits on its inputs and produces two binary digits on its outputs. – A difference bit and a borrow bit.

Table: The truth table for Half Subtractor

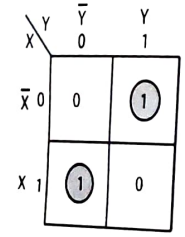
Inputs		Outputs	
Input X	Input Y	Borrow B	Difference D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

The Boolean function from above truth table is written as follows.

Difference (D) $= \overline{X}Y + X\overline{Y}$

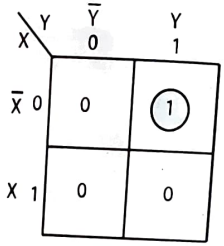
$= X \oplus Y$

Borrow (B) $= \overline{X}Y$



Difference (D) = $\overline{X}Y + X$

Figure (a): K map for Difference



Borrow (B) = $\overline{X}Y$

Figure (b): K map for borrow

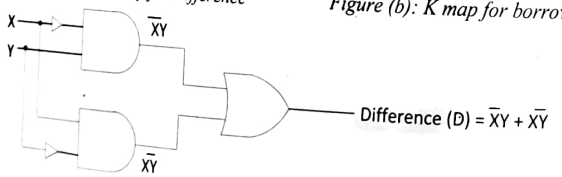


Figure (a): Logic circuit diagram for Difference of Half Subtractor

Figure (b): Logic circuit for Borrow of Half Subtractor

2. Full Subtractor

A full subtractor is a combinational logic circuit that performs a subtraction operation between three numbers of a single bit. Let say the input variables are X, Y and Z and the outputs are Difference (D) and Borrow (B).

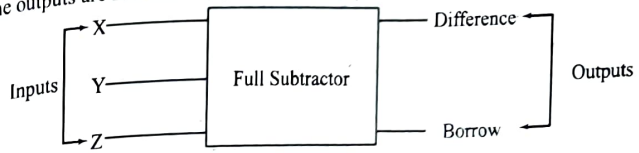


Figure: Block Diagram of Full Adder

The full subtractor accepts three binary digits on its inputs and produces two binary digits on its outputs. – A difference bit and a borrow bit.

***Table:** The truth table for Full Subtractor*

Inputs			Outputs	
Input X	Input Y	Input Z	Borrow B	Difference D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

The Boolean function from the above truth table is calculated as follows.

Difference (D) $= \overline{X}\ \overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\ \overline{Z} + XYZ$

$\therefore F_D = X\ \bar{Y}\ \bar{Z} + XYZ + \bar{X}\ \bar{Y}Z + \bar{X}Y\ \bar{Z}$

$F_D = X(\bar{Y}\ \bar{Z} + YZ) + \bar{X}(\bar{Z} + Y\ \bar{Z})$

$= X(Y \odot Z) + \bar{X}(Y \oplus Z)$

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

Let $Y \oplus Z = P$

$= X\bar{P} + \bar{X}P$

$= X \oplus P$

Difference (D) $= X \oplus Y \oplus Z$

$\therefore F_D = X \oplus Y \oplus Z$

Borrow (B) = $\overline{X}Y + \overline{X}Z + YZ$

Difference (D) = $\overline{X}\,\overline{Y}Z + \overline{X}Y\overline{Z} + X\,\overline{Y}\,\overline{Z} + XYZ$

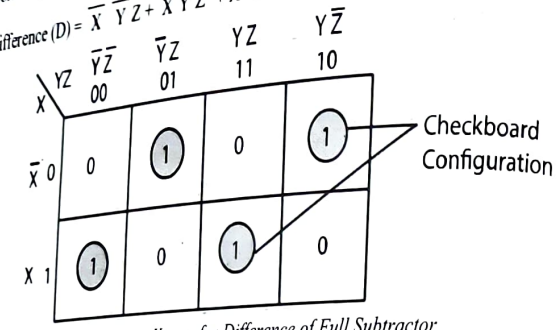


Figure: K map for Difference of Full Subtractor

Borrow (B) = $\overline{X}Y + \overline{X}Z + YZ$

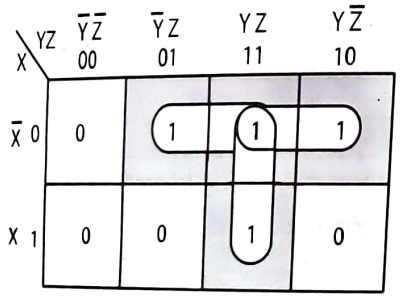


Figure: K map for Borrow of Full Subtractor

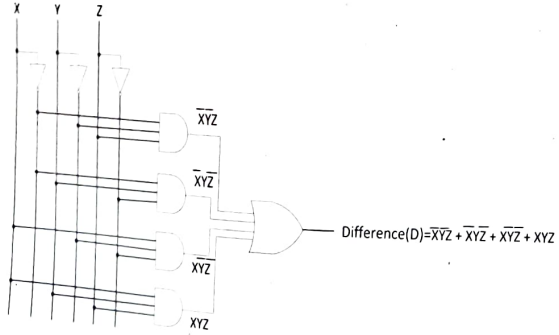


Figure: Logic Circuit diagram for Difference in Full Subtractor

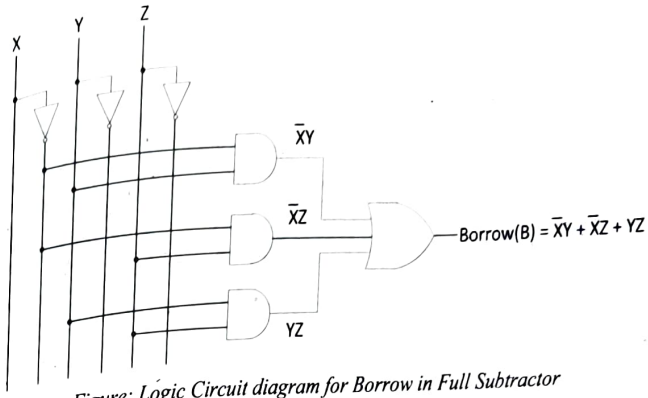


Figure: Logic Circuit diagram for Borrow in Full Subtractor

Decoder

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n-bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.

- It is a combinational circuit that converts binary coded information other codes (e.g., octal, decimal, hexadecimal, etc.)
- A decoder (with enable input) is exactly the same as DEMUX.
- Binary code of n bits can represent <=2^n distinct elements of information.
- A decoder has n inputs and <=2^n output.

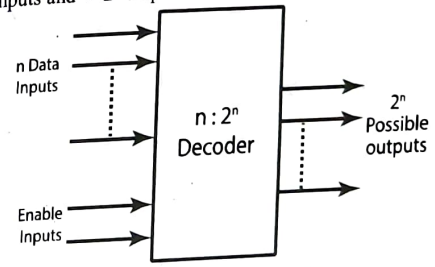


Figure: Block diagram of a decoder

A decoder is a combinational logic circuit that receives n number of inputs and produces 2^n number of outputs.

There are following types of decoder:

1) 2-to-4-line decoder
2) 3-to-8-line decoder
3) BCD-to-Decimal Decoder
4) 4 × 16-line decoder by using two 3 × 8 decoders

Encoder

- An encoder is a combinational logic circuit that performs the reverse operation of decoder.
- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or fewer) input lines and **n** output lines.
- The output lines generate the binary code corresponding to the input value.

It has maximum 2^n input lines and n output lines, hence it encodes the information from 2^n inputs into an n-bit code. Therefore, the encoder encodes 2^n input lines with 'n' bits.

There are following types of encoder:

1) 4-to-2 encoder
2) 8-to-3 encoder
3) Priority Encoder
4) Decimal to BCD
5) Hexadecimal to binary

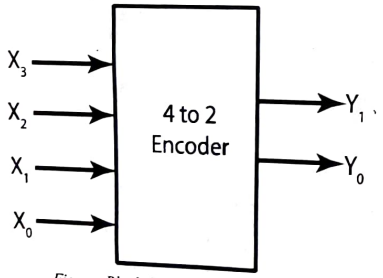


Figure: Block diagram of 4-to-2 encoder

Multiplexers and Demultiplexer

Multiplexer is a circuit that accepts many inputs but one output. A demultiplexer is a circuit that performs function exactly in the reverse of a multiplexer. Generally, multiplexer and demultiplexer are used together, because of the communication systems are bi-directional.

1. **Multiplexer / Data Selector**

A multiplexer is a circuit used to select and route any one of the several input signals to a single output.

It is a combinational logic circuit that select binary information from one of many lines and direct it to a single output line

For m input lines and n selection lines

$2^n = m$

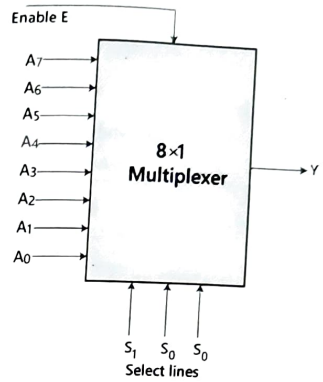


Figure: Block diagram of multiplexer (Source: https://www.javatpoint.com/multiplexer-digital-electronics)

Types of Multiplexer

Following are types of Multiplexer:

a. 2 to 1 Multiplexer
b. 4 to 1 Multiplexer
c. 8 to 1 Multiplexer
d. 16 to 1 Multiplexer

8-to-1 multiplexer

Step 1: Construction of block diagram of 8:1 Mux

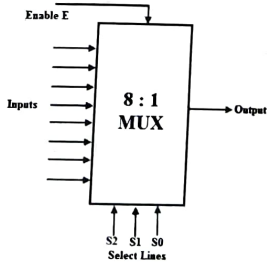


Figure: Block diagram of 8-to-1 multiplexer

Step 2: Calculating the number of select lines:

It has 8 data input line and 1 output lines

Number of select lines (m) = $\log_2 n$

Where n is number of input lines

(m) $= \log_2(8)$

$= \log_2(2^3)$

$= 3 \log_2(2)$

(m) $= 3 \times 1 = 3$ [$\because \log_a a = 1$]

Step 3: Construction of Truth table for 8 to 1 Multiplexer

Table: *Function Table for 8-to-1 multiplexer*

Inputs								Selectors			Outputs
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	S_2	S_1	S_0	Y
-	-	-	-	-	-	-	D_0	0	0	0	D_0
-	-	-	-	-	-	D_1	-	0	0	1	D_1
-	-	-	-	-	D_2	-	-	0	1	0	D_2
-	-	-	-	D_3	-	-	-	0	1	1	D_3
-	-	-	D_4	-	-	-	-	1	0	0	D_4
-	-	D_5	-	-	-	-	-	1	0	1	D_5
-	D_6	-	-	-	-	-	-	1	1	0	D_6
D_7	-	-	-	-	-	-	-	1	1	1	D_7

$Y = S_2 \overline{S_1}\ \overline{S_0}\ \overline{D_0} + S_2 \overline{S_1}\ \overline{S_0} D_0 + S_2 \overline{S_1} S_0 D_0 + S_2 S_1 \overline{S_0}\ \overline{D_0} + S_2 S_1 D_0 + S_2 S_1 S_0 \overline{D_0} + S_2 S_1 S_0 D_0$

Step 4: Construction of Circuit diagram for 8 to 1 multiplexer

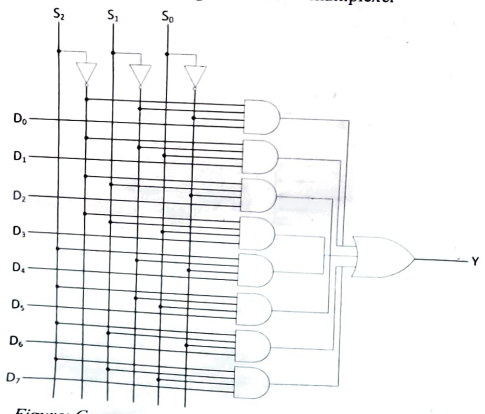


Figure: Construction of Circuit diagram for 8 to 1 multiplexer

Application of Multiplexer:

- Communication system
- Telephone network
- Computer memory
- Transmission from the computer system of a satellite.
- Multiplexer are used in various fields where multiple data need to be transmitted using a 4g. single line.
- An example of 4 to - 1 multiplexer is 1C 74153 in which output is same as the input.

Advantages of Multiplexer

- It reduces the number of wires.
- Reduces circuit complexity and cost.
- Implementation of various circuit in MUX.

2. Demultiplexer/Data Distributor

A demultiplexer is a circuit that receives information on a single input line and transmits information on one of 2^n possible output lines. The selection of a specific output line is controlled by the bit values of n selection lines. The demultiplexer works just inverse of the multiplexer.

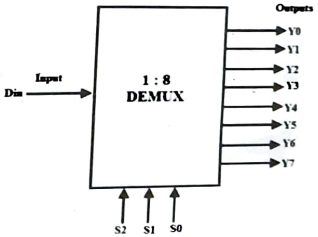


Figure: Demultiplexer

A decoder with enable input can be function as demultiplexer. It is also known as data distributor

Types of Demultiplexer

a. 1-to-2 Demultiplexer
b. 1-to-4 Demultiplexer
c. 1-to-8 Demultiplexer
d. 1-to-2 Demultiplexer

Things to Remember

- Combinational logic circuit is a circuit consisting of logic gates whose output at any time is determined by the combination of present inputs only. The output of combinational logic circuit is not affected by previous inputs as well as previous output.
- The design of a combinational circuit is very crucial task. The design process starts with a verbal communication and ends with logic circuit diagram. While designing a combinational logic circuit following things must be considered.
  1. Number of logic gates
  2. Number of inputs
  3. Propagation time
  4. Number of interconnections
  5. Driving capabilities.
- Binary adder is a combinational logic circuit that is used to add binary numbers. Binary adders are
  1. Half Adder
  2. Full Adder
- Half adder is a combinational logic circuit that is used for adding two single bit binary numbers. The output of half adder is sum and carry and it is given by following Boolean expression.
  Sum (S) = $X \oplus Y$
  Carry (C) = XY
  Where X and Y are inputs
- Full adder is a combinational logic circuit that is used for adding three single bit binary numbers. The output of half adder is sum and carry and it is given by following Boolean expression...
  Sum (S) = $X \oplus Y \oplus Z$
  Carry (C) = XY + XZ + YZ
  Where X, Y and Z are inputs
- Binary subtractor is a combinational logic circuit that is used for subtracting binary numbers. Binary subtractors are
  1. Half Subtractor
  2. Full Subtractor
- Half subtractor is a combinational logic circuit that is used to subtract two single bit binary numbers. The output of half subtractor is Difference and Borrow and it is given by following Boolean expression...
  Difference (D) = $X \oplus Y$
  Borrow (B) = $\overline{X}\,Y$
  Where X and Y are inputs

- Full Subtractor is a combinational logic circuit that is used for subtract three single bit binary numbers. The output of full subtractors is difference and borrow and it is given by following Boolean expression...
  Difference (D) = $X \oplus Y \oplus Z$
  Borrow (B) = $\overline{X}\,Y + \overline{X}\,Z + YZ$
  Where X, Y and Z are inputs
- In digital system it is sometime necessary to convert a code from one form to another. A code converter is a combinational logic circuit for converting a code from one form to another.
- BCD to excess-3: there are ten numbers in BCD (from 0 to 9). So, to represent all numbers four binary bits are required. The number except 0 to 9 are represented by don't care condition. Excess-3 is three number excess to BCD number.
  $W = A + BC + BD = A + B(C + D)$
  $X = \overline{B}\,C + \overline{B}\,D + B\,\overline{C}\,D$
  $Y = CD + \overline{C}\,\overline{D}$
  $Z = \overline{D}$
  Where A, B, C and D are BCD variables and W, X, Y and Z are excess-3 variables

Analysis process of a combinational logic circuit is somewhat reverse process of design.

Binary Arithmetic:

Everything that is stored in or manipulated by the computer is a number. The computer only understands the numbers 1 and 0. Therefore, every number has to be converted to binary (0 s and 1 s) digits. The basic arithmetic operations of the binary number system are addition and subtraction.

a. Binary Addition

Binary addition is a simple mathematical operation used to add two binary numbers together. The logical rules of this operation are implemented in every digital computer – through digital circuits known as adders. There are four rules of binary addition.

For addition, we have four simple rules to remember:

0 + 0 = 0,
0 + 1 = 1,
1 + 0 = 1, and
1 + 1 = 0 (10, 0 is the sum and the carry bit are added to the next column bit)

Table: Addition of Binary Number.

	Input		Output	
s.no	A	B	Sum(S)	Carry(C)
1	0	0	0	0
2	0	1	1	0
3	1	0	1	0
4	1	1	0	1

b. Binary Subtraction

Binary subtraction includes subtracting two binary numbers (comprising only two digits, 0 and 1). It is one of the four binary operations. The binary subtraction has two new terms involved – the difference and the borrow. We have four main rules to remember for the binary Subtraction:

- 0 – 0 = 0,
- 0 – 1 = 1, borrow/take 1 from the adjacent bit to the left
- 1 – 0 = 1, and
- 1 – 1 = 0

Table: Subtraction of binary numbers

	Input		Output	
S.No	A	B	Difference	Borrow
1	0	0	0	0
2	0	1	1	0
3	1	0	1	0
4	1	1	0	1

Operation on Unsigned and Signed Binary Number:

The operations performed on unsigned binary numbers are different from the operations performed on signed binary numbers. Unsigned binary numbers represent positive integers and are used to perform arithmetic operations such as addition, subtraction, multiplication, and division. On the other hand, signed binary numbers represent positive and negative integers and are used to perform arithmetic operations such as addition, subtraction, multiplication, and division, as well as comparisons.

Unsigned Binary Number:

An unsigned binary number is a binary number that represents positive integers only. It is used to perform arithmetic operations such as addition, subtraction, multiplication, and division.

In unsigned binary representation, each bit represents a power of 2, starting with 2^0 for the least significant bit (LSB) and 2^(n-1) for the most significant bit (MSB), where n is the number of bits in the representation.

Signed Binary Number:

A signed binary number is a binary number that represents positive and negative integers. It is used to perform arithmetic operations such as addition, subtraction, multiplication, and division, as well as comparisons.

There are two methods to represent signed binary numbers: sign-magnitude representation and two's complement representation.

- **Sign-magnitude representation**: In sign-magnitude representation, the most significant bit (MSB) is used to represent the sign of the number. A 0 in the MSB indicates a positive number, and a 1 in the MSB indicates a negative number. The remaining bits represent the magnitude of the number.
- **Two's complement representation:** In two's complement representation, a positive number is represented in binary form, and a negative number is represented by taking the binary representation of the positive number and inverting all the bits (changing 0s to 1s and 1s to 0s) and then adding 1.

2.3 Sequential Logic Circuit

- Sequential circuit is basic memory element of sequential logic system.
- The logic circuits whose output at any instant of time depend not only on the present inputs but also on the past outputs are called sequential circuits.
- A sequential circuit consists of a combinational circuit to which storage elements are connected to form a feedback path.
- Sequential circuit = Combinational logic+ memory element

Sequential circuit also called regenerative circuit fall into three types:

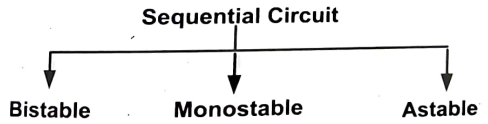


Figure: Types of sequential Circuit.

- Bistable: Two Stable operating point
- Monostable: Only one stable operating point
- Astable: It has no stable operating point

Basic building block of sequential circuit is flip flop. Flip flop is a storage device which store one bit of information. There is a memory element to form a feedback path.

A sequential circuit is specified by a time sequence of inputs, outputs and internal states. The block diagram of sequential circuit is given below:

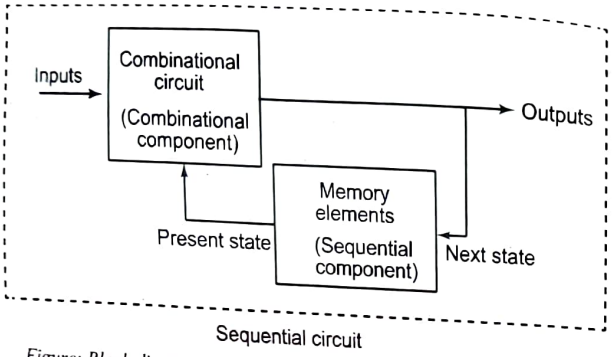


Figure: Block diagram of sequential circuit/ Finite state machine (FSM)

Sequential circuit is slower in operation than combinational circuit. It may or may not contain clock input.

There are two types of sequential circuits:

a. **Synchronous Sequential Circuit:**

It is a system whose behavior can be defined from the knowledge of its signals at discrete instant of time (i.e., by the clock signal parallelly driven by one clock signal).

b. **Asynchronous Sequential circuit:**

It is a system whose behavior depends in the order in which its input signals change and can be affected at any instant of time (one's output is given to clock of another).

Comparison between Combinational and Sequential circuits

***Table:** Comparison between Combinational and Sequential circuits*

S.No	Combinational circuits	Sequential circuits
1	In combinational circuits, the output variables are all times dependent on the combination of input variables.	In sequential circuits, the output variables depend not only on the present input variables but they also depend upon the past history of these input variables.
2	Memory unit is not required in combinational circuits	Memory unit is required to store the past history of input variables in the sequential circuit
3	Combinational circuits are faster in speed because the delay between input and output is due to propagation delay of gates.	Sequential circuits are slower than the combinational circuits
4	Combinational circuits are easy to design.	Sequential circuits are comparatively harder to design.
5	Parallel adder is a combinational circuit.	Serial adder is a sequential circuit.

Flip Flop:

- It is a memory device which can assume only two stable states, which has a pair of complementary outputs and one or more inputs that can cause output state to change. It is also called binary latch.
- A flip flop is a sequential circuit which is popularly known as a basic digital memory circuit. A flip flop stores 1 bit; therefore, it is called as 1-bit memory cell. It has two stable states: logic 1 and logic 0.
- The output of circuit (Q and $\overline{Q}$) will always be complementary. This means that if Q=0, then $\overline{Q}$=1 and vice versa. They will never be equal; Q=$\overline{Q}$=0 or 1 is an invalid state. If Q=1, $\overline{Q}$=0, it is called 1 state or SET state. If Q=0, $\overline{Q}$=1, it is called 0 state or RESET state.

Features of flip flop:

Binary Storage: Flip-flops can store one bit of binary data, either 0 or 1, and retain it until the next clock pulse.

Edge Triggered: Flip-flops change state only when there is a transition (i.e., a rising or falling edge) in the clock signal.

Synchronous Operation: Flip-flops operate in synchrony with the clock signal, which ensures stable and predictable operation.

Clocked Latches: Flip-flops are clocked latches, which means that the state of the flip-flop is controlled by the clock signal and can only change on the rising or falling edge of the clock.

Positive Feedback: Flip-flops use positive feedback, which means that the output of the flip-flop is connected back to its input, creating a feedback loop that is responsible for maintaining the state of the circuit.

Applications of Flip-Flop:

- As a memory element.
- In various types of register
- In counter/timer.
- As a delay time

Types of Flip-flop:

- SR Flip-Flop:
- D Flip flop (Delay or Data Flip Flop)

- JK Flip Flop
- T Flip Flop (Toggle Flip flop)
- Master Slave Flip-Flop

Introduction to S-R Flip-Flop

The SR flip-flop is a type of bistable latch circuit in digital electronics. It has two inputs: "set" (S) and "reset" (R), and two outputs: Q and its inverse, Q'.

SR stands for "Set-Reset". The reset input resets the flip-flop back to its original state with an output Q that will be either at a logic level "1" or logic "0" depending upon this set/reset condition.

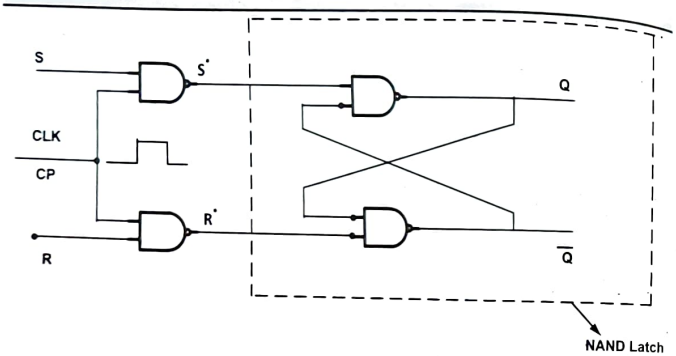


Figure: Logic Circuit diagram of SR Flip Flop

Table: *Truth table for S-R NAND latch*

S*	R*	Q	$\overline{Q}$
0	0	Not used	
0	1	1	0
1	0	0	1
1	1	Previous State (Memory)	

$\therefore \quad S^* = \overline{S.CLK}$

$= \overline{S} + C\overline{LK}.$

$\therefore \quad R^* = \overline{R.CLK}$

$\therefore \quad R^* = \overline{R} + C\overline{LK}$

$[\overline{A+B} = \overline{A}.\ \overline{B}$ and $\overline{A.B} = \overline{A} + \overline{B}]$

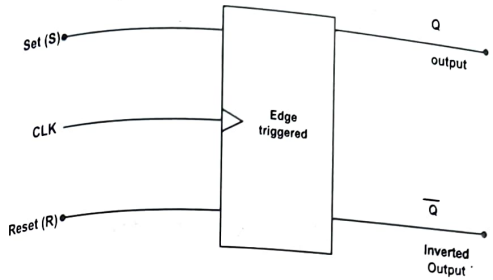


Figure: Graphics Symbol of SR Flip Flop

Table: *Truth table for S-R flip flop*

CLK	S	R	Q	$\overline{Q}$
0	X	X	Memory Q_n(Previous State)	
1	0	0	Memory Q_n (Previous State)	
1	0	1	0	1
1	1	0	1	0
1	1	1	Not used [Invalid]	

Case 1: When CLK=1

When CLK = 0, $S^* = \overline{S} + C\overline{LK}$

$\therefore \quad S^* = \overline{S} + \overline{0} = \overline{S} + 1 = 1$..... Equation (3)

and $R^* = \overline{R} + C\overline{LK} = \overline{R} + \overline{0} = \overline{R} + 1 = 1$..... Equation (4)

Whatever the value of $\overline{S}$ and $\overline{R}$, it's not going to change the value of R* and S*.

Case 2:

When CLK = 1

$S^* = \overline{S} + \overline{CLK} = \overline{S} + \overline{1} = \overline{S} + 0 = \overline{S}$.....Equation (5)

$R^* = \overline{R} + \overline{CLK} = \overline{R} + \overline{1} = \overline{R} + 0 = \overline{R}$.....Equation (6)

$\therefore \quad S^* = \overline{s}$

$\therefore \quad R^* = \overline{R}$

Case 2.1

When CLK = 1, S=0, R=0

$\therefore \quad S^* = \overline{S} = \overline{0} = 1$

$\therefore \quad R^* = \overline{R} = \overline{0} = 1$

Case 2.2 When CLK=1, S=0, R=1

$S^* = \bar{S} + \overline{CLK} = \bar{0} + \bar{1} = 1 + 0 = 1$

$S^*=1$

$R^* = \bar{R} + \overline{CLK}$

$= \bar{R} + \bar{1}$

$= \bar{R} + 0 = \bar{1} + 0 = 0$

$\therefore$ **$R^* = 0$**

Case 3: When CLK=1, S=1, R=0

$\therefore \quad R^* = \bar{R} + \overline{CLK} = \bar{0} + \bar{1} = 1 + 0 = 1$

$S^* = \bar{S} + \overline{CLK}$

$= \bar{S} + \bar{1} = \bar{1} + \bar{1} = 0 + 0 = 0$

Case 4: When CLK=1, S=1, R=1

$S^* = \bar{S} + \overline{CLK} = \bar{1} + \bar{1} = 0 + 0 = 0$

$R^* = \bar{R} + \overline{CLK} = \bar{1} + \bar{1} = 0 + 0 = 0$

***Table:** Truth table for S-R flip flop*

CLK	S	R	Q_{n+1} (Next State)
0	X	X	Q_n = Present State (Memory)
1	0	0	Q_n = Present State (Memory)
1	0	1	0
1	1	0	1
1	1	1	Invalid [Do not use this configuration]

***Table:** Characteristics table of S-R flip flop*

Q_n	S_n	R_n	Q_{n+1}
0	0	0	Q_n (0) = 0
0	0	1	0 = 0
0	1	0	1 = 1
0	1	1	X = X (Indeterminate)
1	0	0	Q_n (1) = 1
1	0	1	0 = 0
1	1	0	1 = 1
1	1	1	X = X (Indeterminate)

***Table:** Excitation table for S-R flip flop*

Q_n		Q_{n+1}	S	R
0		0	0	X
0		1	1	0
1		0	0	1
1		1	X	0

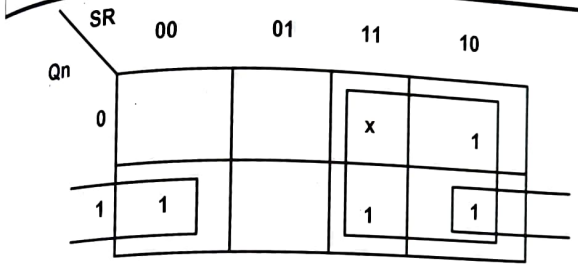


***Table:** K-map for Excitation table of S-R flip flop*

The characteristic equation can be generating from the excitation table of SR flip flop. The Characteristics equation is given by:

$\therefore \quad Q_{n+1} = Q_n \bar{R}_n + S_n$

Next State = function of present state and input.

Gated Flip-Flops:

The addition of two AND gates at the R and S inputs will result in a flip flop that can be enabled or disabled. When the ENABLE input is HIGH, information at the R and S will be transmitted directly to the outputs. The latch is said to be enabled. When the ENABLE input is LOW, the AND gate outputs must be low and changes in neither R nor S will have any effect on the flip flop output Q. The latch is said to be disabled.

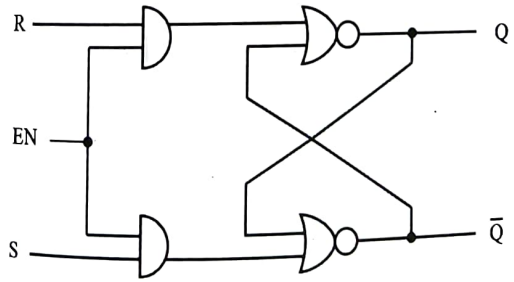


Figure: Gated R-S flip-flop (NOR)

Enable (EN)	S	R	Q_{n+1}
0	X	X	Qn
1	0	0	Qn
1	0	1	0
1	1	0	1
1	1	1	Invalid

Figure: Truth Table

When two inputs signal R and S are applied to the circuit, there is time delay between these signals, which may lead to a wrong output result. In order to overcome this problem of unequal propagation delay time of input signals, the gated flip flop is used.

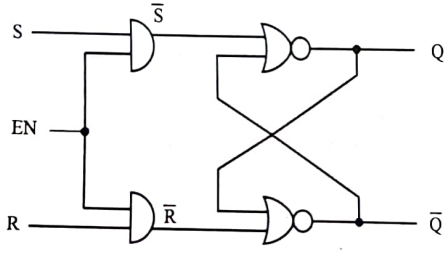


Figure: Gated RS flipflop (NAND)

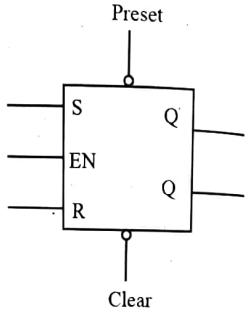
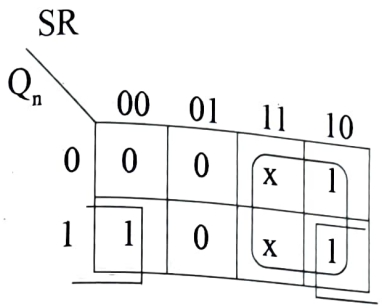


Figure: Logic symbol

The characteristics table of gated RS Flip flop is given by:

Q_n	S_n	R_n	Q_{n+1}	Remark
0	0	0	Q_n (0) = 0	No change
0	0	1	0 = 0	Reset
0	1	0	1 = 1	set
0	1	1	X = X (Indeterminate)	Invalid
1	0	0	Q_n (1) = 1	set
1	0	1	0 = 0	Reset
1	1	0	1 = 1	set
1	1	1	X = X (Indeterminate)	Invalid



$Q_n + 1 = S + Q_n R$

This is the characteristic equation

Gated D flip flop (Delay or Data Flip-flop)

The data flip flop has only one input called data (D) input and two output Q and $\overline{Q}$. It can be constructed from S-R flip flop by inserting an inverter between S and R and assigning the symbol D to S input. The output Q is same as D input.

The D flip flop is designed to overcome the problem of forbidden state and race conditions. It is a circuit that has single S input that is inverted using a NAND gate and given to the R input. Hence, two separate inputs are not required. The single S input is called the D-input. D-flip flop is the modification of the clock S-R flip-flop.

Table: *Truth tables for S-R flip flop*

CLK	S	R	Q_n+1 (N.S)
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	Invalid X

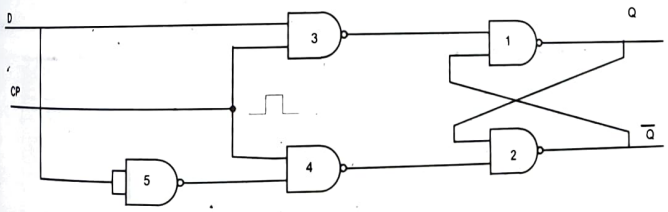


Figure: Logic Circuit Diagram of gated D Flip Flop using NAND gate

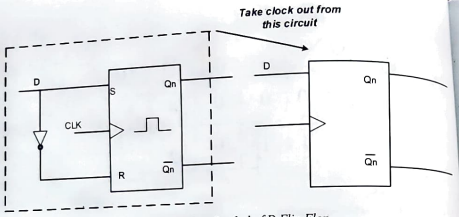


Figure: Graphics Symbol of D Flip Flop

***Table:** Truth table for D flip flop*

CLK	D	Q_{n+1}	
0	X	Q_n(Previous i/p)	Whatever the value of D the o/p will not be affected.
1	0	0	
1	1	1	

The Characteristics table is derived from Truth Table

Table: Characteristics table for D flip flop

Q_n	D	Q_{n+1}	Remark	
0	0	0	Same as D	The next state will depend on the input and pervious state.
0	1	1	Same as D	
1	0	0	Same as D	
1	1	1	Same as D	

The next state is same as Data input.

$\boxed{D = Q_n + 1}$

Excitation table is derived from characteristics table:

***Table:** Excitation table for D flip flop*

Q_n	$Q_n + 1$	D
0	0	0
0	1	1
1	0	0
1	1	1

$\therefore D = Q_n + 1$

K -Map for characteristics equation

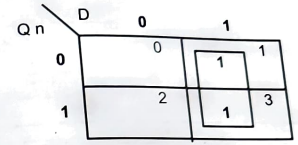


Table: K-map for Excitation table of D flip flop

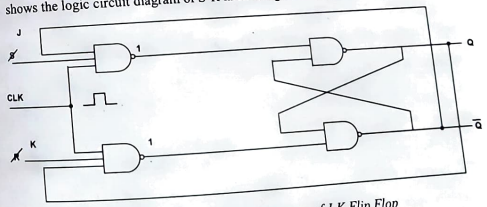
$Q_{n+1} = D$

Advantages of D flip flop over RS Flip flops:

In some events, both inputs become high in Rs flip-flop which is undesirable condition. The drawback of RS flipflop is overcome in D flip flop. There is only one input that is D which drive the flip flop

Edge Triggered Flip-Flop:

- A J-K flip flop is basically a gated S-R flip flop with addition of clocked input circuit
- A Jk flip-flop is a refinement of the S-R flip-flop in that the indeterminate state of SR type is defined in the JK types. Input J and K behave like inputs S and R to set and clear the flip flop (not that in a JK flip-flop, the letter J is for set and the letter K is for clear). When inputs are applied to both J and K simultaneously, the flip-flop switches to its complement state, that is, if Q=1, it switches to Q=0, and vice versa.
- Following Figure (a) shows the logic circuit diagram of J-K flip flop and figure (b) shows the logic circuit diagram of S-R latch using NAND gate



Figure(a): Logic Circuit Diagram of J-K Flip Flop

Table: Truth table for J-K flip flop

CLK	S (J)	R (K)	Q_n+1
0	X	X	Q_n (memory)
1	0	0	Q_n (memory)
1	0	1	0
1	1	0	1
1	1	1	Not used

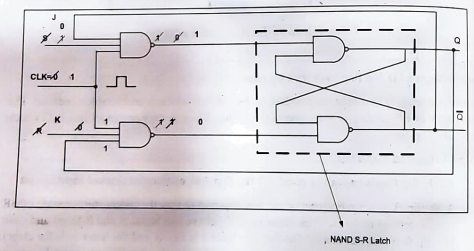


Figure (b): Logic Circuit Diagram of SR latch using NAND gate

Following Four condition are observed in JK flip-

1. When Clock is low, CLK=0 and When the input to the NAND S-R Latch is 1,1 That is CLK= 0, Memory (Q_n)
2. When CLK= 1, J=1 , K=0 The value of Q= 1 and $\overline{Q}$=0
3. When CLK= 1, J=0, K=1, $\overline{Q}$=1, and Q=0
4. For CLK= 1, J=1, K=1

Initially, assume Q=0 and $\overline{Q}$=1

Q = 0, 1, 0, 1 .....

And $\overline{Q}$ = 1, 0, 1, 0 .....

This condition is called Racing.

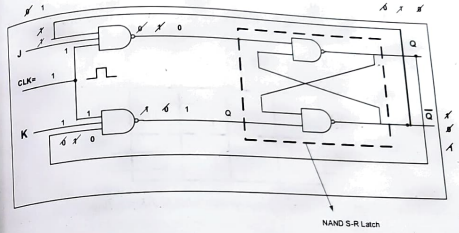


Figure: Logic Circuit Diagram of JK latch using NAND gate

Table: *Truth table for J-K flip flop*

CLK	S (J)	R (K)	Q_n+1
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	Q_n(Toggle)

Toggle means that output quickly changes from 0 to 1, 1 to 0, and 0 to 1 and so on.

Characteristics Table for J-K Flip Flop

Since, next state (Qn +1) is dependent on the present input J and K and previous state.

Table: *Characteristic table for J-K flip flop*

Q_n	J	K	Q_n+1
0	0	0	$Q_n = 0$
0	0	1	0
0	1	0	1
0	1	1	$\overline{Qn} = \overline{0} = 1$
1	0	0	$Q_n = 1$
1	0	1	0
1	1	0	1
1	1	1	$\overline{Qn} = \overline{1} = 0$

Characteristics Equation of JK Flip Flop

The characteristics equation of JK flip flop can be generated from the characteristics table. The table is simplified using k map.

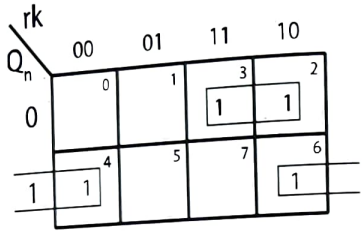


Figure: K-map for Characteristic table of J-K flip flop

Characteristics equation of JK FF

Therefore, $Q_{n+1} = Q_n \overline{K} + \overline{Qn} J$ (This is the characteristics equation of JK FF)

Excitation Table:

Excitation table can be derived from the characteristics table.

Table: *Excitation table for J-K flip flop*

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

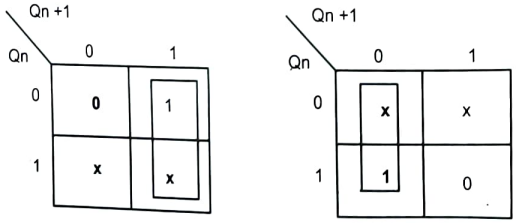


Figure: K-map for Excitation table of J-K flip flop

Therefore, $J = Q_{n+1}$ $\quad K = \overline{Qn+1}$

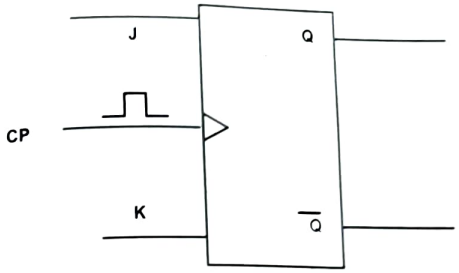


Figure: Graphics symbol of JK flip flop

Race around condition in JK Flip Flop

For J-K flip-flop, if J=K=1, and if CLK=1 for a long period of time, then Q output will toggle as long as CLK is high, which makes the output of the flip-flop unstable or uncertain. This problem is called race around condition in J-K flip-flop. This problem (Race around Condition) can be avoided by ensuring that the clock input is at logic "1" only for a very short time. This introduced the concept of **Master Slave JK** flip flop.

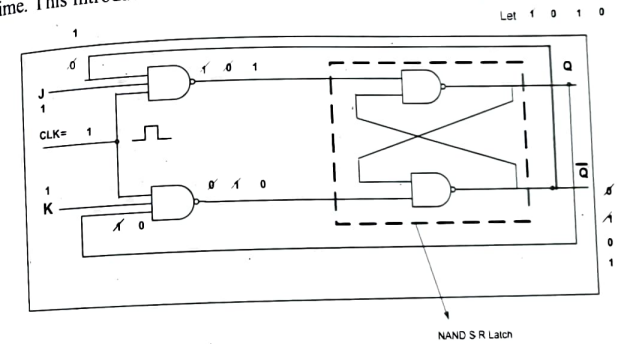


Figure: Race around condition in J-K Flip Flop

Mater- Slave:

- The given Figure describes the concept of a Master-Slave Flip-Flop. This type of circuit is composed of two JK flip-flops that are connected in a series configuration, with one acting as the "master" and the other as the "slave." The output from the master flip-flop is connected to the inputs of the slave flip-flop, and the output from the slave is fed back to the inputs of the master. The circuit also includes an inverter that is connected to the clock pulse. The inverter inverts the clock pulse, so that the clock pulse for the slave flip-flop is the inverse of the clock pulse for the master. In other words, if the CP is 0 for the master flip-flop, it is 1 for the slave, and if the CP is 1 for the master, it is 0 for the slave.

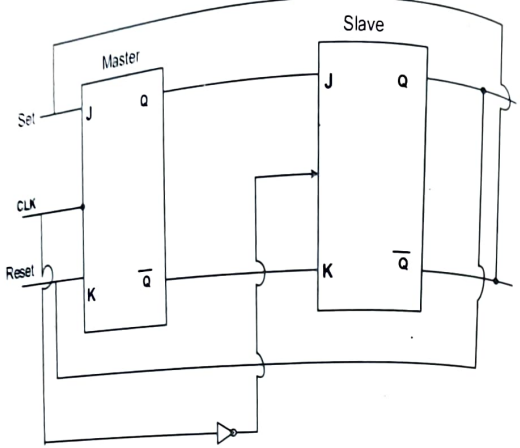


Figure: Logic diagram of Master Slave J-K flip flop

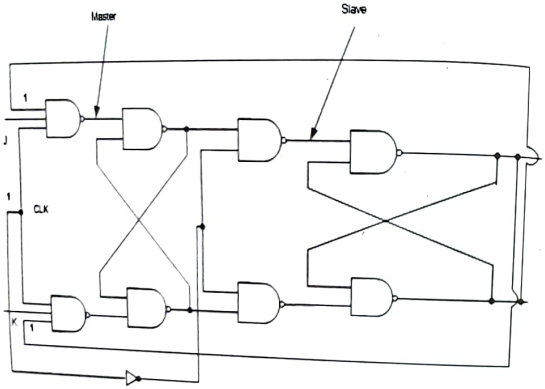


Figure: Working principle of Master Slave J-K Flip Flop

***Table:** Truth table for J-K flip flop*

CLK	S (J)	R (K)	Q_{n+1}
0	X	X	Q_n (memory)
1	0	0	Q_n (memory)
1	0	1	0
1	1	0	1
1	1	1	Q_n(Toggle)

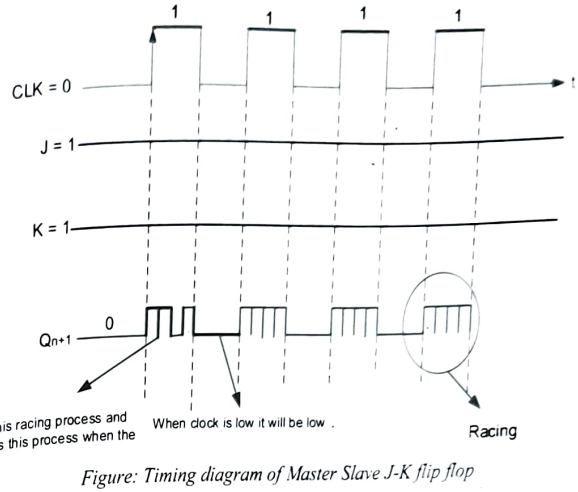


Figure: Timing diagram of Master Slave J-K flip flop

Conditions to overcome racing:

- $T/2$ < propagation delay of Flip Flop
- Edge triggering
- Master slave (Negative edge triggered flip-flop)

Excitation Table of Flip-Flop

A flip-flop excitation table is a table used in the design of sequential circuits to determine the necessary input conditions that result in a desired change in state. This table provides a convenient way to determine the input conditions that will cause the required transition and is an essential tool in the design of flip-flop based sequential circuits.

SR Flip-flop

Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

D Flip-flop

Q(t)	Q(t+1)	DR
0	0	0
0	1	1
1	0	0
1	1	1

JK flip-flop

Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

T flip-flop

Q(t)	Q(t+1)	DR
0	0	0
0	1	1
1	0	1
1	1	0

Figure: Excitation table of different flip-flop

Things to Remember

- The output of the flip-flop changes more than once.
- It occurs when J=K=1 and propagation delay of flip-flop T_{PD} is less than clock pules width T_{PW}.
- The overall circuit acts as a single flip-flop and stores one bit of information.
- The problem of race around condition is not present in M-S J-K flip-flop.
- The master is triggered when the clock goes to 1.
- The slave is triggered when the clock goes to 0
- When clock is inactive, then the next state of the flip-flop is the previous state or no change in the state.
- D flip-flop behaves as a combinational circuit and the buffer behaves as a D flip-flop
- If the output of the flip flop changes more than once in same clock period, then it is called race around condition.
- Race around condition can be eliminated by master slave flip flop or edge triggered flip-flop.

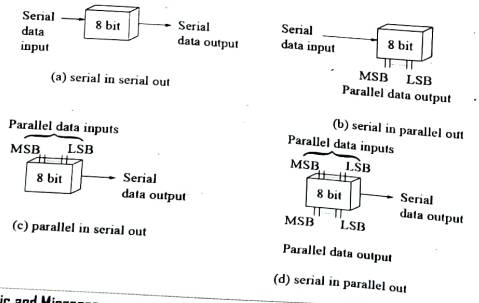
Introduction to Register

A register is the group of binary cells suitable for holding information.

- It is a group of flip flop sensitive to pulse transition.
- A register is a digital circuit with two basic functions: Data Storage and data movement.
- A register consists of one or more flip flop used to store and shift data.
- An 'n' bit register consists of a group of 'n' flip flop capable of storing n bits binary information. A register capable of shifting its binary information in one or more direction is called shift register.
- Register are essential synchronous system

Types of Register:

There are four types of Register:



a) **Serial In Serial Out (SISO):** SISO shift registers accepts data serially i.e., one bit at a time and output taken from it is also in serial form.

b) **Serial In Parallel Out (SIPO):** Data are entered serially but output is taken parallelly. Once the data are stored, each bits appears on its respective output line, all at a time.

c) **Parallel In Serial Out (PISO):** A parallel-in serial-out (PISO) shift register is a type of digital circuit that takes a parallel input and converts it into a serial output. The input data is stored in parallel, meaning each data bit is stored in a separate flip-flop or storage element. The data is then shifted out in a serial fashion, with each bit being shifted out one after the other in sequence. PISO shift registers are useful for converting parallel data into a serial format for transmission or storage, as serial data is easier to transmit or store than parallel data. The circuit can be used for a variety of applications, including data storage, data transmission, and data manipulation.

d) **Parallel In Parallel Out (PIPO):** Data are entered simultaneously into their respective stage on parallel lines rather than on a bit by bit basis. Output is available simultaneously.

Applications of shift Register:

Shift registers have a variety of applications in digital circuits, some of the most common include:

- **Data storage:** Shift registers can be used as a form of memory to store digital data. They are especially useful in situations where a small amount of data needs to be stored temporarily.
- **Serial-to-parallel conversion:** Shift registers can be used to convert serial data into parallel data, making it easier to process.
- **Parallel-to-serial conversion:** Shift registers can also be used to convert parallel data into serial data, which is useful for transmitting data over a single communication line.
- **Data shifting:** Shift registers can be used to shift data from one register to another in a cyclic manner. This is useful in many applications, such as digital filtering, digital signal processing, and image processing.
- **Pattern generation**: Shift registers can be used to generate various digital patterns, such as pseudo-random sequences, codes, and waveforms.
- **Display interfaces:** Shift registers are commonly used in display interfaces to control the data flow to displays such as LEDs, LCDs, and OLEDs.
- **Input/output expansion:** Shift registers can be used to expand the number of inputs and outputs in a digital circuit.
- As ring counter and Johnson counter

Synchronous and Asynchronous Counter:

The combination of flip flop that performs the counting operation are known as counter. A counter is probably one of the most useful and versatile subsystems in a digital system. It is a sequential circuit that passes through predefined number of states.

- Counter are basically used to count number of clock pulse applied. It can also be used for frequency divider, time measurement, frequency measurement, range measurement, pulse width and waveform generator.
- With n-FF, maximum possible stage in the counter is 2^n.

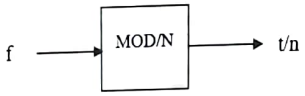
  $N<2^n$ or $n>\log_2 N$

  Where, N=number of state

  N=

  Counter are classified into two broad categories according to the way they are clocked:

  n = number of FF.
- Number of stages used in counter mean modulus of counter, i.e., if MOD is 5, then counter =5 stages and if MOD is n, then counter =n stage.



- For example, a decade counter is applied with frequency of 10 MHz, then output frequency is:

  $F_{out} = t_{in}/_{10} = 10/10 = 1MHz$

a. Asynchronous (ripple or Serial) Counter:

- In this counter, the first flip flop is clocked by the external clock pulse and then each successive flip flop is clocked by the output of the preceding flip flop.
- Asynchronous counter is simple and straightforward in operation and its construction usually requires a minimum number of hardware.
- It does not have speed limitation.
- FFs are not clocked simultaneously
- It is very simple.
- The speed of operation is very slow
- The power consumption is less
- Minimum hardware is required
- The cost is less

b. Synchronous (Parallel)Counter:

- In synchronous counter, the clock input is connected to all of the flip flops so that they are clocked simultaneously.
- An increase in speed of operation can be achieved by use of synchronous counter.
- FFs are clocked simultaneously
- It is very complex
- The speed of operation is fast
- The power consumption is more
- It requires more hardware
- The cost is more.

Comparison between synchronous and asynchronous counter:

Asynchronous/Ripple/Serial	Synchronous/Ring/Parallel
Different FFs are applied with different clock	All FFs are applied with the same clock
It is slower	It is faster
There is a fixed count sequence that is: up or down.	Any count sequence is possible
Decoding error are present	No decoding error are present
These are also called ripple counter or serial counter	These are also called ring or parallel counter.
Time delay of all flip flop+ time delay of combinational circuit.	Time delay of one flip flop + time delay of combinational circuit.

Up/down synchronous counters:

It is defined as a bidirectional counter that is capable of counting either up or down. An input (control) line up/$\overline{down}$ (or simply up) specifies the direction of counting.

up/$\overline{down}$ =1→count upward

up/$\overline{down}$ =0→ count downward

BCD Counter:

- It is an MSI device which counts the BCD sequence.

Ring Counter:

- This counter counts a prescribed number of pules and returns back to zero

Mod-n Counter

- This counter counts the pules till (n-1)

2.4 MICROPROCESSOR: PROGRAMMING

Internal Architecture of Microprocessor

The 8085A (commonly known as 8085) is an 8-bit general purpose microprocessor capable of addressing 64K of memory.

The device has 40 pins, require a +5V single power supply and can operate with a 3-MHZ, single phase clock.

The Intel 8085 A is a complete 8-bit parallel central processing unit. The main components of 8085A are array of registers, the arithmetic logic unit, the encoder/decoder, and timing and control circuits linked by an internal data bus. The block diagram is shown in Figure.

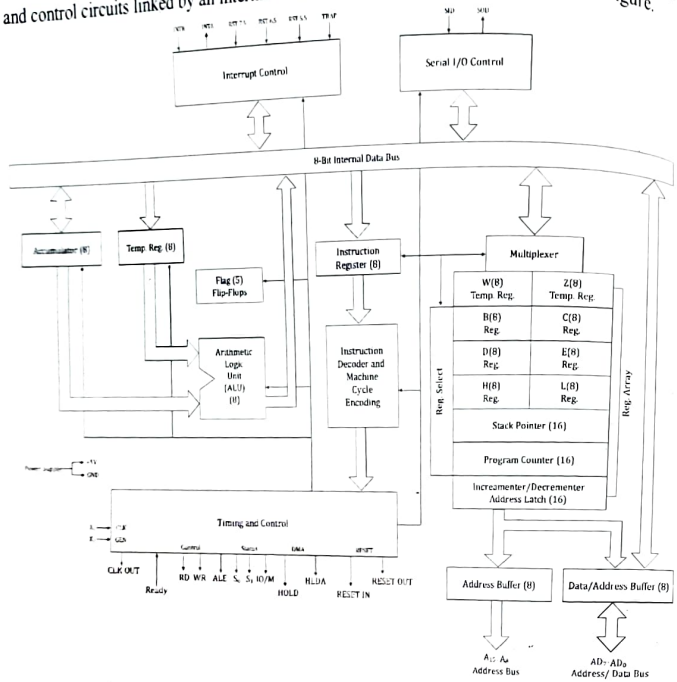


Figure: The 8085 A Microprocessor Functional Block Diagram

a) Arithmetical and Logic Unit (ALU)

The Arithmetic Logic Unit (ALU) is the central component of the computing system and performs various arithmetic and logic operations. The ALU comprises several components, including the accumulator, the temporary register, arithmetic and logic circuits, and five flags. The temporary register is used to temporarily store data during an arithmetic or logic operation, while the final result is stored in the accumulator. Additionally, the flags (which are flip-flops) are set or reset based on the outcome of the operation.

b) Accumulator (register A)

- It is one of the general-purpose registers of microprocessor also called as A register.
- The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU).
- This register is used to store 8-bit data and to perform arithmetic and logical operations.
- The result of an operation is stored in the accumulator. The user can access this register by giving appropriate instructions (commands).

c) Temporary registers (W & Z)

It is also called as operand register (8 bit). It provides operands to ALU. ALU can store immediate result in temporary register. It is not accessible by user.

d) Instruction register (IR)

- It is an 8 bit register not accessible to the programmer.
- It receives the operation codes of instruction from internal data bus and passes to the instruction decoder which decodes so that microprocessor knows which type of operation is to be performed.

e) Register Array

(Scratch pad registers B, C, D, and E): It is an 8-bit register accessible to the programmers.

- Data can be stored upon it during program execution.
- These can be used individually as 8-bit registers or in pair BC, DE as 16-bit registers.
- The data can be directly added or transferred from one to another.
- Their contents may be incremented or decremented and combined logically with the content of the accumulator.

f) Register H & L

- They are 8-bit registers that can be used in same manner as scratch pad registers.

g) Stack Pointer (SP)

- It is a 16-bit register used as a memory pointer.
- A stack is nothing but the portion of RAM (Random access memory).
- It points to a memory location in R/W memory, called the stack.
- The beginning of the stack is defined by loading a 16-bit address in the stack pointer.
- The stack is usually accessed in Last in First out (LIFO) fashion.

h) Program Counter (PC)

- It contains the address of the next instruction to be fetched from Memory.
- Program counter is a special purpose register.

Flag

The Flag register is a Special Purpose Register. Depending upon the value of result after any arithmetic and logical operation the flag bits become set (1) or reset (0). In 8085 microprocessors, flag register consists of 8 bits and only 5 of them are useful. The different flags are described as:

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

a) **Carry Flag/ Borrow Flag**

- If the last operation generates a carry its status will 1 otherwise 0. It can handle the carry or Borrow from one word to another.
- Carry Flag is also known as Borrow Flag.
- During subtraction (A-B), if A>B it becomes reset and if (A<B) it becomes set 1-carry out from MSB bit on addition or borrow into MSB bit on subtraction 0-no carry out or borrow into MSB bit

b) **Zero**

- If the result of last operation is zero, its status will be 1 otherwise 0. It is often used in loop control and in searching for particular data value.
- After any arithmetical or logical operation if the result is 0 (00) H, the zero flag becomes set i.e. 1, otherwise it becomes reset i.e. 0. 00H zero flag is 1. from $01H_{to}$ FFH zero flag is 0
- 1- zero result, 0- non-zero result

c) **Sign**

- If the most significant bit (MSB) of the result of the last operation is 1 (negative), then its status will be 1 otherwise 0.
- After any operation if the MSB (B (7)) of the result is 1, it indicates the number is negative and the sign flag becomes set, i.e. 1. If the MSB is 0, it indicates the number is positive and the sign flag becomes reset i.e. 0. from 00H to 7F, sign flag is 0
  from 80H to FF, sign flag is 1
- MSB is 1 (negative),0- MSB is 0 (positive)

d) **Parity**

- If the result of the last operation has even number of 1's (even parity), its status will be 1 otherwise 0.
- If after any arithmetic or logical operation the result has even parity, an even number of 1 bit, the parity register becomes set i.e. 1, otherwise it becomes reset i.e. 0.

e) **Auxiliary carry**

- If the last operation generates a carry from the lower half word (lower nibble), its status will be 1 otherwise 0. Used for performing BCD arithmetic.

Timing and Control Unit

- It is used to generate timing and control signals which are necessary for the execution of instructions. It is used to control data flow between CPU and peripherals (including memory).

- It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits
  -Control Signals: READY, RD', WR', ALE
- Status Signals: S0, S1, IO/$\overline{M}$
- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

Interrupt Controls:

- The various interrupt controls signal (INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP) are used to interrupt a microprocessor.

Serial I/O controls:

- It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

Characteristics (Features) of 8085A Microprocessor and its Signals

The all the signals associated with 8085 can be classified into 6 groups:

BUS:

The 8085 has 16 signal lines that are used as the address bus; however, these lines are split into two segments A_{15}-A_8 and AD_7- AD_0. The eight signals A_{15}-A_8 are unidirectional and used as high order bus.

a) **Data Bus:** The signal lines AD7- AD0 are bidirectional, they serve a dual purpose. They are used the low order address bus as well as data bus.

b) **Control and status signals:** This group of signals includes two control signals ($\overline{RD}$ and $\overline{WR}$), three status signals (IO/$\overline{M}$, S1 and S0) to identify the nature of the operation, and one special signal (ALE) to indicate the beginning of the operation.

c) **ALE- Address Latch Enable:** This is a positive going pulse generated every time the 8085 begins an operation (Machine Cycle): It indicates that the bits AD7-AD0 are address bits. This signal is used primarily to latch the low-order address from the multiplexed bus and generate a separate set of eight address lines A7 –A0.

- **$\overline{RD}$- Read:** this is a read control signal (active low). This signal indicates that the selected I/O or memory device is to be read and data are available on the data bus.
- **$\overline{WR}$ Write:** This is a write control signal (active low). This signal indicates that the data on the data bus are to be written into a selected memory or I/O location.
- **IO/$\overline{M}$:** This is a status signal used to differentiate between I/O and memory operations. When it is high, it indicates an I/O operation; When it is low indicates a
- **Memory Operation:** This signal is combined with $\overline{RD}$ (Read) and $\overline{WR}$ (Write) to generate I/O and memory signals.
- **S1 and S0:** This status signals, similar to IO/ $\overline{M}$, can identify various operations, but they are rarely used in small systems.

Supply and Clock Frequency:

a) **VCC:** +5V power supply

b) **VSS:** Ground reference

c) **X1 and X2:** A crystal (RC or LC network) is connected at these two pins for frequency OUT: It can be used as the system clock for other devices.

Externally Initiated Signals:

- **INTR (Input):** interrupt request, used as a general-purpose interrupt.
- **$\overline{\text{INTA}}$ (Output):** This is used to acknowledge an Interrupt.
- **RST 7.5, 6.5, 5.5 (Inputs):** These are vectored interrupts that transfer the program control to specific memory locations. They have higher priorities than INTR interrupt. Among these three, the priority order is 7.5, 6.5, and 5.5.
- **TRAP (input):** This is a non-Maskable interrupt with highest priority.
- **HOLD (input):** This signal indicates that a peripheral such as a DMA (Direct Memory Access) controller is requesting use of Address and data bus.
- **HLDA (output):** Hold Acknowledge: This signal acknowledges the HOLD request
- **READY (Input):** This signal is used to delay the microprocessor Read or Write cycles until a slow- responding peripheral is ready to send or accept data. When this signal goes low, the microprocessor waits for an integral number of clock cycles until it goes high.
- **$\overline{\text{RESETIN}}$:** When the signal on this pin goes low, the program counter is set to zero, the buses are tri-stated, and MPU is reset.
- **RESET OUT:** This signal indicates that the MPU is being reset. The signal can be used to reset other devices.

Serial I/O ports:

The 8085 has two signals to implement the serial transmission: SID (Serial Input Data) and SOD (Serial Output Data). In serial transmission, data bits are sent over a single line, one bit at a time, such as the transmission over telephone lines.

2.5 MICROPROCESSOR SYSTEM

Parallel Interface. Introduction to Programmable Peripheral Interface (PPI), Serial Interface, Synchronous and Asynchronous Transmission, Serial Interface Standards. Introduction to Direct Memory Access (DMA) and DMA Controllers. (AExE0205)

Parallel Interface:

The parallel interface is a method of communicating between a microprocessor and other peripheral devices such as memory, input/output (I/O) devices, and other processors. In a parallel interface, multiple bits of data are transferred simultaneously on multiple parallel lines. This allows for a faster transfer of data compared to a serial interface, where data is transferred one bit at a time on a single line.

In a microprocessor-based system, the parallel interface is used to connect the microprocessor to external devices such as memory, I/O devices, and other processors. The microprocessor sends commands and data to these devices using the parallel interface, and it also receives data from these devices in the same manner.

Introduction to Programmable Peripheral Interface (PPI):

Programmable Peripheral Interface (PPI) is a digital interface standard used in microprocessor-based systems to communicate between peripheral devices and a microprocessor. It is a parallel interface, meaning that multiple bits of data are transferred simultaneously on multiple lines, allowing for faster data transfer compared to serial interfaces.

PPI is used to connect peripheral devices such as input/output (I/O) devices, memory, and other processors to the microprocessor. The microprocessor sends commands and data to these devices using the PPI, and it also receives data from these devices in the same manner.

Serial Interface:

A serial interface is a method of communicating between a microprocessor and peripheral devices such as memory, input/output (I/O) devices, and other processors. In a serial interface, data is transferred one bit at a time on a single line. This is in contrast to a parallel interface, where multiple bits of data are transferred simultaneously on multiple parallel lines.

In a microprocessor-based system, the serial interface is used to connect the microprocessor to external devices such as memory, I/O devices, and other processors. The microprocessor sends commands and data to these devices using the serial interface, and it also receives data from these devices in the same manner.

Synchronous and Asynchronous Transmission:

In microprocessor-based systems, synchronous and asynchronous transmission are used to communicate between the microprocessor and peripheral devices such as memory, input/output (I/O) devices, and other processors.

Synchronous transmission is used when a high data transfer rate is required and when a shared clock signal between the microprocessor and peripheral devices is available. This method is often used in applications where real-time communication is required, such as in industrial control systems, where the accurate timing of data is critical.

Asynchronous transmission is used when a shared clock signal is not available or when a simple and flexible communication solution is desired. This method is often used in applications where a fast data transfer rate is not a critical requirement, and where a simple and cost-effective communication solution is desired.

Serial Interface Standard:

There are several serial interface standards that are commonly used in microprocessor-based systems to communicate between the microprocessor and peripheral devices such as memory, input/output (I/O) devices, and other processors. Some of the most commonly used serial interface standards are:

Universal Asynchronous Receiver/Transmitter (UART): UART is a simple and widely used serial interface standard that is often used in embedded systems and other applications where a fast data transfer rate is not a critical requirement.

Inter-Integrated Circuit (I2C): I2C is a serial communication protocol that is used to connect peripheral devices to a microprocessor. It is widely used in embedded systems and other applications where multiple devices need to communicate with a single microprocessor.

Serial Peripheral Interface (SPI): SPI is a full-duplex serial communication protocol that is used to connect peripheral devices to a microprocessor. It is widely used in embedded systems and other applications where a fast data transfer rate is required.

RS-232: RS-232 is a standard for serial communication transmission of data. It is used to connect peripheral devices to a microprocessor and is widely used in industrial control systems, scientific instruments, and other applications where a fast data transfer rate is not a critical requirement.

RS-422 and RS-485: RS-422 and RS-485 are serial communication standards that are used to connect peripheral devices to a microprocessor. They are commonly used in industrial control systems, scientific instruments, and other applications where a fast data transfer rate is required and where the communication distance between devices is long.

Introduction to Direct Memory Access and DMA Controller:

Direct Memory Access (DMA) is a method used in microprocessor-based systems to transfer data directly between memory and peripheral devices without involving the microprocessor. This allows for faster data transfer and reduces the workload of the microprocessor, freeing it up to perform other tasks.

The DMA controller is responsible for managing the transfer of data between memory and peripheral devices. It communicates directly with the memory and peripheral devices, and it also communicates with the microprocessor to set up the transfer and to obtain the necessary information about the transfer, such as the source and destination addresses, the size of the transfer, and the direction of the transfer.

2.6 INTERRUPT OPERATIONS

Interrupt:

In microprocessor architecture, an interrupt is a mechanism that allows a processor to temporarily stop executing its current task and attend to a high-priority request from another device, referred to as an interrupt request (IRQ).

Interrupts are a way for devices to communicate with the processor and request its attention, thereby allowing the processor to handle multiple tasks at the same time.

Interrupts can be triggered by a variety of events, such as the arrival of new data from a peripheral, a timer expiring, or an error occurring. When an interrupt is triggered, the processor stops its current task, saves its current state, and begins executing a special

interrupt service routine (ISR) associated with the interrupt. The ISR performs the necessary actions to service the interrupt, and when it is complete, the processor returns to the task it was previously executing, restoring its saved state

Interrupt Service Routine (ISR):

Interrupt service routine (ISR) is actually a call back function (program) in case of software or device driver (I/O device) in case of hardware. When an interrupt is acknowledged by the processor, the routine or program which is running currently gets paused or interrupted, and the ISR program gets executed. An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. Whenever an interrupt occurs, the controller completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler.

- An Interrupt Service Routine (ISR) is a piece of software code that is executed by the hardware in response to an interrupt.
- The ISR evaluates the interrupt and decides how to handle it. It performs the necessary actions to service the interrupt and then returns a logical value indicating the outcome.
- The main objective of an ISR is to process the interrupt and return control to the primary program. To prevent slowing down the device's performance and that of lower priority ISRs, an ISR must be executed quickly.
- Like procedures, the final instruction in an ISR should be a return statement.

A small program or a routine that when executed, services the corresponding interrupting source is called an ISR.

TRAP

It is a non-maskable interrupt, having the highest priority among all interrupts. By default, it is enabled until it gets acknowledged. In case of failure, it executes as ISR and sends the data to backup memory. This interrupt transfers the control to the location 0024H.

RST7.5

It is a maskable interrupt, having the second highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH address.

A Maskable Interrupt is an interrupt signal that has the second highest priority among all interrupts. When this interrupt is triggered, the microprocessor saves the current content of the Program Counter (PC) register onto the stack and branches to the address 003CH.

RST 6.5

It is a maskable interrupt, having the third highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H address.

RST 5.5

It is a maskable interrupt. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH address.

INTR

It is a maskable interrupt, having the lowest priority among all interrupts. It can be disabled by resetting the microprocessor.

When INTR signal goes high, the following events can occur –

- The microprocessor continuously checks the status of the INTR signal while executing each instruction.
- If the INTR signal is active (high), the microprocessor finishes the current instruction and sends a low-active Interrupt Acknowledge signal.
- After receiving the interrupt, the microprocessor saves the address of the next instruction to the stack and begins executing the received instruction.

Interrupt Processing

Interrupt processing refers to the process by which the microprocessor responds to an interrupt signal. When an interrupt is triggered, the microprocessor temporarily pauses its current task, saves the current state of the program and execution context, and starts executing the Interrupt Service Routine (ISR) associated with the interrupt. The ISR is responsible for processing the interrupt and determining the appropriate action to take. Once the ISR has completed its task, it returns control to the main program, and the microprocessor resumes its previous task. Interrupt processing is a crucial aspect of microprocessing, as it enables the microprocessor to handle external events and respond to them in a timely and efficient manner.

MULTIPLE CHOICE QUESTIONS

1. **Hexadecimal can express in one digit what is done by following binary number of binary digits**
   A. One B. Four
   C. Two D. Eight
2. **NAND gates are preferred over others because these have lower fabrication area**
   A. Have lower fabrication area
   B. Can be used to make any gate
   C. Consume lest electronic power
   D. Provide maximum density in a chip
3. **According to Demorgan's theorem A + B + C + D =**
   A. $\overline{A}\overline{B}\overline{C}\overline{D}$ B. AB +CD
   C. A + BC + D D. A + B + C +D
4. **According to Demorgan's theorem $\overline{A}\overline{B}\overline{C}\overline{D}$ =**
   A. $\overline{A} + \overline{B} + \overline{C} + \overline{D}$ B. A + B + C + D
   C. A + B + C + D D. A +B + CD
5. **The logic unit shown below is as the type**
   A. AND B. NAND
   C. OR D. NOT
6. **The truth table shown below is for**

Inputs		Outputs
a	b	
0	0	0
0	1	1
1	0	1
1	1	1

   A. NAND B. OR
   C. AND D. NOT
7. **Odd parity of word can be conveniently tested by**
   A. OR gate B. NOR gate
   C. AND gate D. XOR gate
8. **A record at the end of file which contains control total is**
   A. Pointers B. Trunk
   C. Trailer D. Trunkey
9. **Binary means .....**
   A. Three B. Ten
   C. Four D. Two
10. **The digits used in a binary number system are .....and.....**
    A. 9 and 0 B. 1 and 2
    C. 0 and1 D. 3 and 4
11. **Names, numbers and other information needed to solve a problem are called .....**
    A. Program B. Data
    C. Instruction D. Controls
12. **The ..... is a sequence of instructions that tells the computer how to process the data**
    A. Data B. Controls
    C. Program D. Instruction
13. **Computer ICs work reliably because they based on...design**
    A. Top-button B. Two-stage
    C. System D. Two-status
14. **When a transistor is cut off or saturated, transistor.....have almost no effect**
    A. Wave B. Stage
    C. Variations D. Circuits