

# UNIT -I

## **BINARY SYSTEMS**

- Analog Vs Digital
- Digital Systems
- Binary numbers
- Number base conversions
- Compliments
- Octal and Hexadecimal Numbers
- Signed Binary Numbers

## **ANALOG Vs DIGITAL:**

To learn and understand about the digital logic design, the initial knowledge we require is to differentiate between analog and digital. The following are few that differentiate between analog and digital.

- Analog information is made up of a continuum of values within a given range.
- At its most basic, digital information can assume only one of two possible values: one/zero, on/off, high/low, true/false, etc.
- Digital Information is less susceptible to noise than analog information
- Exact voltage values are not important, only their class (1 or 0)
- The complexity of operations is reduced, thus it is easier to implement them with high accuracy in digital form.

## **DIGITAL SYSTEMS**

**Digital** means electronic technology that generates, stores, and processes data in terms of two states: positive and non-positive. Positive is expressed or represented by the number 1 and non-positive by the number 0.

A „**digital system**“ is a data technology that uses discrete (discontinuous) values represented by high and low states known as bits. By contrast, non-digital (or analog) systems use a continuous range of values to represent information. Although digital representations are discrete, the information represented can be either discrete, such as numbers, letters or icons, or continuous, such as sounds, images, and other measurements of continuous systems.

## **BINARY**

Binary describes a numbering scheme in which there are only two possible values for each digit: 0 and 1. The term also refers to any digital encoding/decoding system in which there are exactly two possible states. In digital data memory, storage, processing, and communications, the 0 and 1 values are sometimes called "low" and "high," respectively.

## **BINARY NUMBER/BINARY NUMBER SYSTEM**

The binary number system is a numbering system that represents numeric values using two unique digits (0 and 1). Most of the computing devices use binary numbering to represent electronic circuit voltage state, (i.e., on/off switch), which considers 0 voltage input as off and 1 input as on.

This is also known as the base-2 number system (The base-2 system is a positional notation with a radix of 2), or the binary numbering system. Few examples of binary numbers are as follows:

- 10
- 111
- 10101
- 11110

## COMPLIMENTS

Compliments are used in digital computers to simplify the subtraction operation and for logical manipulation. Simplifying operations leads to simpler, less expensive circuits to implement the operations.

There are 2 types of complements for each base  $r$  system.

- (1) The radix complement
- (2) Diminished radix complement

**Radix complement:** Also referred to as the  $r$ 's complement.

**Diminished radix complement:** Also referred to as  $(r-1)$ 's complement.

## OCTAL NUMBERS

The **Octal Number System** is another type of computer and digital base number system. The **Octal Numbering System** is very similar in principle to the previous hexadecimal numbering system except that in Octal, a binary number is divided up into groups of only 3 bits, with each group or set of bits having a distinct value of between 000 (0) and 111 (7). Octal numbers therefore have a range of just "8" digits, (0, 1, 2, 3, 4, 5, 6, 7) making them a Base-8 numbering system and therefore,  $q$  is equal to "8".

**HEXADECIMAL NUMBERING SYSTEM:** The one main disadvantage of binary numbers is that the binary string equivalent of a large decimal base-10 number can be quite long. When working with large digital systems, such as computers, it is common to find binary numbers consisting of 8, 16 and even 32 digits which makes it difficult to both read and write without producing errors especially when working with lots of 16 or 32-bit binary numbers. One common way of overcoming this problem is to arrange the binary numbers into groups or sets of four bits (4-bits). These groups of 4-bits uses another type of numbering system also commonly used in computer and digital systems called **Hexadecimal Numbers**

The “Hexadecimal” or simply “Hex” numbering system uses the **Base of 16** system and are a popular choice for representing long binary values because their format is quite compact and much easier to understand compared to the long binary strings of 1’s and 0’s.

Being a Base-16 system, the hexadecimal numbering system therefore uses 16 (sixteen) different digits with a combination of numbers from 0 through to 15. In other words, there are 16 possible digit symbols.

<b>Decimal</b>	<b>Binary</b>	<b>Octal</b>	<b>Hexadecimal</b>
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

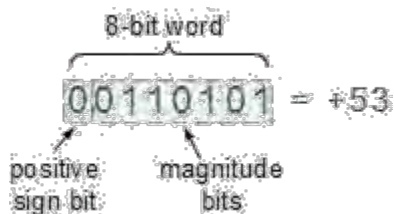
## **SIGNED BINARY NUMBERS**

In mathematics, positive numbers (including zero) are represented as unsigned numbers. That is we do not put the +ve sign in front of them to show that they are positive numbers. However, when dealing with negative numbers we do use a -ve sign in front of the number to show that the number is negative in value and different from a positive unsigned value and the

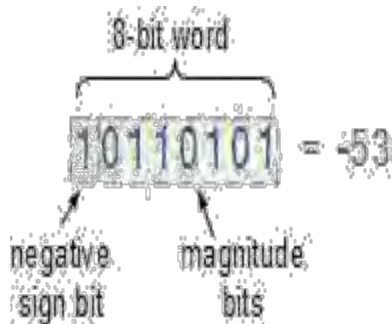
same is true with **signed binary numbers**. However, in digital circuits there is no provision made to put a plus or even a minus sign to a number, since digital systems operate with binary numbers that are represented in terms of “0”s and “1”s.

So to represent a positive (N) and a negative (-N) binary number we can use the binary numbers with sign. For signed binary numbers the most significant bit (MSB) is used as the sign. If the sign bit is “0”, this means the number is positive. If the sign bit is “1”, then the number is negative. The remaining bits are used to represent the magnitude of the binary number in the usual unsigned binary number format.

### Positive Signed Binary Numbers.



### Negative Signed Binary Numbers



## BINARY CODES

In the coding, when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as a code. The digital data is represented, stored and transmitted as group of binary bits. This group

is also called as **binary code**. The binary code is represented by the number as well as alphanumeric letter.

### Advantages of Binary Code

Following is the list of advantages that binary code offers.

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.
- Binary codes make the analysis and designing of digital circuits if we use the binary codes.
- Since only 0 & 1 are being used, implementation becomes easy.

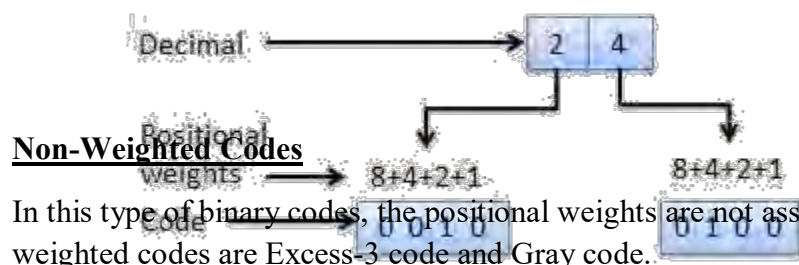
### Classification of binary codes

The codes are broadly categorized into following four categories.

- Weighted Codes
- Non-Weighted Codes
- Binary Coded Decimal Code
- Alphanumeric Codes
- Error Detecting Codes
- Error Correcting Codes

### Weighted Codes

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.



### Excess-3 code

The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding (0011)<sub>2</sub> or (3)<sub>10</sub> to each code word in 8421. The excess-3 codes are obtained as follows.



Decimal	BCD				Excess-3			
	8	4	2	1	BCD + 0011			
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

### Gray Code

It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position. It has a very special feature that, only one bit will change each time the decimal number is incremented as shown in fig. As only one bit changes at a time, the gray code is called as a unit distance code. The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.

### Application of Gray code

- Gray code is popularly used in the shaft position encoders.
- A shaft position encoder produces a code word which represents the angular position of the shaft.

### Binary Coded Decimal (BCD) 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. In the BCD, with four bits we can



represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.

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

### **Advantages of BCD Codes**

- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only

### **Disadvantages of BCD Codes**

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is little more complicated.
- BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary

### **Alphanumeric codes**

A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.

The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items. The following three alphanumeric codes are very commonly used for the data representation.

- American Standard Code for Information Interchange (ASCII).

- Extended Binary Coded Decimal Interchange Code (EBCDIC).
- Five bit Baudot Code.

ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

### **Error Codes**

There are binary code techniques available to detect and correct data during data transmission.

### **NUMBER BASE CONVERSIONS**

There are many methods or techniques which can be used to convert code from one format to another. We'll demonstrate here the following

- Binary to BCD Conversion
- BCD to Binary Conversion
- BCD to Excess-3
- Excess-3 to BCD

Binary to BCD Conversion

Steps

- **Step 1** -- Convert the binary number to decimal.
- **Step 2** -- Convert decimal number to BCD.

Example – convert  $(11101)_2$  to BCD.

Step 1 – Convert to Decimal

Binary Number –  $11101_2$

Calculating Decimal Equivalent –

Step	Binary Number	Decimal Number

Step 1	11101 <sub>2</sub>	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101 <sub>2</sub>	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101 <sub>2</sub>	29 <sub>10</sub>

Binary Number – 11101<sub>2</sub> = Decimal Number – 29<sub>10</sub>

Step 2 – Convert to BCD

Decimal Number – 29<sub>10</sub>

Calculating BCD Equivalent. Convert each digit into groups of four binary digits equivalent

Step	Decimal Number	Conversion
Step 1	29 <sub>10</sub>	0010 <sub>2</sub> 1001 <sub>2</sub>
Step 2	29 <sub>10</sub>	00101001 <sub>BCD</sub>

Result

$(11101)_2 = (00101001)_{BCD}$

BCD to Binary Conversion

## Steps

- **Step 1** -- Convert the BCD number to decimal.
- **Step 2** -- Convert decimal to binary.

Example – convert  $(00101001)_{\text{BCD}}$  to Binary.

Step 1 - Convert to BCD

BCD Number –  $(00101001)_{\text{BCD}}$

Calculating Decimal Equivalent. Convert each four digit into a group and get decimal equivalent for each group.

Step	BCD Number	Conversion
Step 1	$(00101001)_{\text{BCD}}$	$0010_2 \ 1001_2$
Step 2	$(00101001)_{\text{BCD}}$	$2_{10} \ 9_{10}$
Step 3	$(00101001)_{\text{BCD}}$	$29_{10}$

BCD Number –  $(00101001)_{\text{BCD}}$  = Decimal Number –  $29_{10}$

Step 2 - Convert to Binary

Used long division method for decimal to binary conversion.

Decimal Number –  $29_{10}$

Calculating Binary Equivalent

Step	Operation	Result	Remainder
Step 1	$29 / 2$	14	1
Step 2	$14 / 2$	7	0
Step 3	$7 / 2$	3	1
Step 4	$3 / 2$	1	1
Step 5	$1 / 2$	0	1

As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the least significant digit (LSD) and the last remainder becomes the most significant digit (MSD).

Decimal Number –  $29_{10}$  = Binary Number –  $11101_2$

### Result

$(00101001)_{BCD} = (11101)_2$

BCD to Excess-3

Steps

- **Step 1** -- Convert BCD to decimal.
-

- **Step 2** -- Add  $(3)_{10}$  to this decimal number.
- **Step 3** -- Convert into binary to get excess-3 code.

Example – convert  $(1001)_{\text{BCD}}$  to Excess-3.

Step 1 – Convert to decimal

$$(1001)_{\text{BCD}} = 9_{10}$$

Step 2 – Add 3 to decimal

$$(9)_{10} + (3)_{10} = (12)_{10}$$

Step 3 – Convert to Excess-3

$$(12)_{10} = (1100)_2$$

**Result**

$$(1001)_{\text{BCD}} = (1100)_{\text{XS-3}}$$

Excess-3 to BCD Conversion

Steps

- **Step 1** -- Subtract  $(0011)_2$  from each 4 bit of excess-3 digit to obtain the corresponding BCD code.

Example – convert  $(10011010)_{\text{XS-3}}$  to BCD.

Given XS-3 number = 1 0 0 1 1 0 1 0

Subtract  $(0011)_2 =$  0 0 1 1 0 0 1 1

$$\begin{array}{r} \text{-----} \\ \text{BCD} = 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1 \end{array}$$

**Result**

$$(10011010)_{\text{XS-3}} = (01100111)_{\text{BCD}}$$



# UNIT –II

## **BOOLEAN ALGEBRA :**

- Basic Definitions
- Axiomatic Definition of Boolean Algebra



- Basic Theorems and properties of Boolean Algebra
- Boolean Functions
- Canonical and Standard Forms, Other Logic Operations
- Digital Logic Gates
- Integrated Circuits

**Boolean Algebra:** Boolean algebra, like any other deductive mathematical system, may be defined with a set of elements, a set of operators, and a number of unproved axioms or postulates. A *set* of elements is any collection of objects having a common property. If  $S$  is a set and  $x$  and  $y$  are certain objects, then  $x \in S$  denotes that  $x$  is a member of the set  $S$ , and  $y \notin S$  denotes that  $y$  is not an element of  $S$ . A set with a denumerable number of elements is specified by braces:  $A = \{1, 2, 3, 4\}$ , i.e. the elements of set  $A$  are the numbers 1, 2, 3, and 4. A *binary operator* defined on a set  $S$  of elements is a rule that assigns to each pair of elements from  $S$  a unique element from  $S$ . \_ Example: In  $a * b = c$ , we say that  $*$  is a binary operator if it specifies a rule for finding  $c$  from the pair  $(a, b)$  and also if  $a, b, c \in S$ .

**CLOSURE:** The Boolean system is *closed* with respect to a binary operator if for every pair of Boolean values, it produces a Boolean result. For example, logical AND is closed in the Boolean system because it accepts only Boolean operands and produces only Boolean results.

\_ A set  $S$  is closed with respect to a binary operator if, for every pair of elements of  $S$ , the binary operator specifies a rule for obtaining a unique element of  $S$ .

For example, the set of natural numbers  $N = \{1, 2, 3, 4, \dots\}$  is closed with respect to the binary operator plus (+) by the rule of arithmetic addition, since for any  $a, b \in N$  we obtain a unique  $c \in N$  by the operation  $a + b = c$ .

**ASSOCIATIVE LAW:** A binary operator  $*$  on a set  $S$  is said to be associative whenever  $(x * y) * z = x * (y * z)$  for all  $x, y, z \in S$ , for all Boolean values  $x, y$  and  $z$ .

**COMMUTATIVE LAW:**

A binary operator  $*$  on a set  $S$  is said to be commutative whenever  $x * y = y * x$  for all  $x, y, z \in S$ .

**IDENTITY ELEMENT:**

A set  $S$  is said to have an identity element with respect to a binary operation  $*$  on  $S$  if there exists an element  $e \in S$  with the property  $e * x = x * e = x$  for every  $x \in S$

## BASIC IDENTITIES OF BOOLEAN ALGEBRA

*Postulate 1 (Definition):* A Boolean algebra is a closed algebraic system containing a set  $K$  of two or more elements and the two operators  $\cdot$  and  $+$  which refer to logical AND and logical OR

- $x + 0 = x$
- $x \cdot 0 = 0$
- $x + 1 = 1$
- $x \cdot 1 = x$
- $x + x = x$
- $x \cdot x = x$
- $x + x' = 1$
- $x \cdot x' = 0$
- $x + y = y + x$
- $xy = yx$
- $x + (y + z) = (x + y) + z$
- $x(yz) = (xy)z$

- $x(y + z) = xy + xz$
- $x + yz = (x + y)(x + z)$
- $(x + y)' = x'y'$
- $(xy)' = x' + y'$
- $(x')' = x$

### DeMorgan's Theorem

(a)	$(a + b)' = a'b'$
(b)	$(ab)' = a' + b'$
Generalized DeMorgan's Theorem	
(a)	$(a + b + \dots z)' = a'b' \dots z'$
(b)	$(a.b \dots z)' = a' + b' + \dots z',,$

### AXIOMATIC DEFINITION OF BOOLEAN ALGEBRA:

1. Closure
  - a. Closure with respect to (wrt) OR (+)
  - b. Closure with respect to AND (•)
2. Identity
  - a. Identity element wrt to OR : 0
  - b. Identity element wrt to AND : 1
3. Commutative Property
  - a. Commutative Property wrt to OR :  $x + y = y + x$
  - b. Commutative Property wrt to AND :  $x \cdot y = y \cdot x$
4. Distributive Property

a.  $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

b.  $x + (y \cdot z) = (x + y)(x + z)$

## 5. Existence of Complement




a.  $x + x' = 1$

b.  $x \cdot x' = 0$

## LOGIC GATES

Formal logic: In formal logic, a statement (proposition) is a declarative sentence that is either true(1) or false (0). It is easier to communicate with computers using formal logic.

- Boolean variable: Takes only two values – either true (1) or false (0). They are used as basic units of formal logic.
- Boolean algebra: Deals with binary variables and logic operations operating on those variables.
- Logic diagram: Composed of graphic symbols for logic gates. A simple circuit sketch that represents inputs and outputs of Boolean functions.

Name	Graphic symbol	Algebraic function	Truth table															
Inverter	A  x	$x = A'$	<table><tr><td>A</td><td>x</td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	x	0	1	1	0									
A	x																	
0	1																	
1	0																	
AND	A  B x	$x = AB$	<table><tr><td>A</td><td>B</td><td>x</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> <p>True if both are true.</p>	A	B	x	0	0	0	0	1	0	1	0	0	1	1	1
A	B	x																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR	A  B x	$x = A + B$	<table><tr><td>A</td><td>B</td><td>x</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table> <p>True if either one is true.</p>	A	B	x	0	0	0	0	1	1	1	0	1	1	1	1
A	B	x																
0	0	0																
0	1	1																
1	0	1																
1	1	1																

## INTEGRATED CIRCUIT

An integrated circuit or monolithic integrated circuit (also referred to as an IC, a chip, or a microchip) is a set of electronic circuits on one small plate ("chip") of semiconductor material, normally silicon. This can be made much smaller than a discrete circuit made from independent electronic components. ICs can be made very compact, having up to several billion transistors and other electronic components in an area the size of a human fingernail.

## LEVEL OF INTEGRATION

### 1.SSI : Small Scale Integration

It has less than 100 components(about 10gates)

## 2.MSI: Medium Scale Integration

It contains less than 500 components or have more than 10 but less than 100 gates.

## 3.LSI: Large scale integration

Number of components is between 500 and 300000 or have more than 100gates.

## 4.VLSI:very large scale integration. process of creating an integrated circuit (IC)

by combining thousands of transistors into a single chip.

## **DIGITAL LOGIC FAMILIES**

**Transistor–transistor logic (TTL)** is a class of digital circuits built from bipolar junction transistors (BJT) and resistors. It is called *transistor–transistor logic* because both the logic gating function (e.g., AND) and the amplifying function are performed by transistors (contrast with resistor–transistor logic (RTL) and diode–transistor logic (DTL)).

## **Emitter coupled logic(ECL)**

Emitter-coupled logic (ECL) is the fastest logic circuit family available for conventional logic-system design.<sup>4</sup> High speed is achieved by operating all bipolar transistors out of saturation, thus avoiding storage-time delays, and by keeping the logic signal swings relatively small (about 0.8 V or less), thus reducing the time required to charge and discharge the various load and parasitic capacitances.

Complementary Metal oxide semiconductor (CMOS): Technology for constructing integrated circuits. CMOS technology is used in microprocessors, microcontrollers, static RAM, and other digital logic circuits. CMOS technology is also used for several analog circuits such as image sensors (CMOS sensor), data converters, and highly integrated transceivers for many types of communication.

## INTRODUCTION

Minimization of switching functions is to obtain logic circuits with least circuit complexity. This goal is very difficult since how a minimal function relates to the implementation technology is

important. For example, If we are building a logic circuit that uses discrete logic made of small scale Integration ICs(SSIs) like 7400 series, in which basic building block are constructed and are available for use. The goal of minimization would be to reduce the number of ICs and not the logic gates. For example, If we require two 6 and gates and 5 Or gates,we would require 2 AND ICs(each has 4 AND gates) and one OR IC. (4 gates). On the other hand if the same logic could be implemented with only 10 nand gates, we require only 3 ICs. Similarly when we design logic on Programmable device, we may implement the design with certain number of gates and remaining gates may not be used. Whatever may be the criteria of minimization we would be guided by the following:

- Boolean algebra helps us simplify expressions and circuits
- Karnaugh Map: A graphical technique for simplifying a Boolean expression into either form: minimal sum of products (MSP)
- minimal product of sums (MPS)
- Goal of the simplification.
- There are a minimal number of product/sum terms
- Each term has a minimal number of literals
- Circuit-wise, this leads to a minimal two-level implementation

### **K-map Simplification**

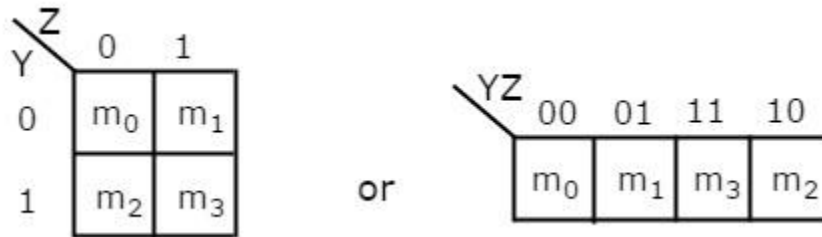
- Imagine a two-variable sum of minterms
- $x''y'' + x''y$
- Both of these minterms appear in the top row of a Karnaugh map, which means that they both contain the literal  $x''$

### **K-Maps for 2 to 5 Variables**

K-Map method is most suitable for minimizing Boolean functions of 2 variables to 5 variables. Now, let us discuss about the K-Maps for 2 to 5 variables one by one.

## 2 Variable K-Map

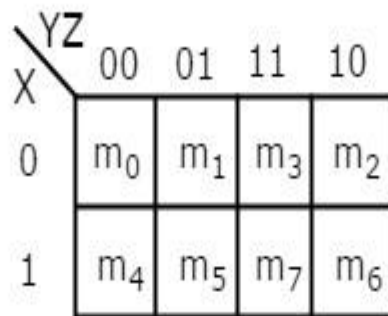
The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows **2 variable K-Map**.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are  $\{(m_0, m_1), (m_2, m_3), (m_0, m_2) \text{ and } (m_1, m_3)\}$ .

## 3 Variable K-Map

The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows **3 variable K-Map**.



- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are  $\{(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) \text{ and } (m_2, m_0, m_6, m_4)\}$ .
- The possible combinations of grouping 2 adjacent min terms are  $\{(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m_7) \text{ and } (m_2, m_6)\}$ .
- If  $x=0$ , then 3 variable K-map becomes 2 variable K-map.

#### 4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows **4 variable K-Map**.

WX \ YZ	YZ			
	00	01	11	10
00	$m_0$	$m_1$	$m_3$	$m_2$
01	$m_4$	$m_5$	$m_7$	$m_6$
11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
10	$m_8$	$m_9$	$m_{11}$	$m_{10}$

- There is only one possibility of grouping 16 adjacent min terms.
- Let  $R_1, R_2, R_3$  and  $R_4$  represents the min terms of first row, second row, third row and fourth row respectively. Similarly,  $C_1, C_2, C_3$  and  $C_4$  represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are  $\{(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)\}$ .
- If  $w=0$ , then 4 variable K-map becomes 3 variable K-map.

#### 5 Variable K-Map

The number of cells in 5 variable K-map is thirty-two, since the number of variables is 5. The following figure shows **5 variable K-Map**.



		V=0						V=1			
		YZ						YZ			
WX		00	01	11	10	WX		00	01	11	10
	00	m <sub>0</sub>	m <sub>1</sub>	m <sub>3</sub>	m <sub>2</sub>		00	m <sub>16</sub>	m <sub>17</sub>	m <sub>19</sub>	m <sub>18</sub>
	01	m <sub>4</sub>	m <sub>5</sub>	m <sub>7</sub>	m <sub>6</sub>		01	m <sub>20</sub>	m <sub>21</sub>	m <sub>23</sub>	m <sub>22</sub>
	11	m <sub>12</sub>	m <sub>13</sub>	m <sub>15</sub>	m <sub>14</sub>		11	m <sub>28</sub>	m <sub>29</sub>	m <sub>31</sub>	m <sub>30</sub>
	10	m <sub>8</sub>	m <sub>9</sub>	m <sub>11</sub>	m <sub>10</sub>		10	m <sub>24</sub>	m <sub>25</sub>	m <sub>27</sub>	m <sub>26</sub>

- There is only one possibility of grouping 32 adjacent min terms.
- There are two possibilities of grouping 16 adjacent min terms. i.e., grouping of min terms from m<sub>0</sub> to m<sub>15</sub> and m<sub>16</sub> to m<sub>31</sub>.
- If v=0, then 5 variable K-map becomes 4 variable K-map.

In the above all K-maps, we used exclusively the min terms notation. Similarly, you can use exclusively the Max terms notation.

### Minimization of Boolean Functions using K-Maps

If we consider the combination of inputs for which the Boolean function is „1“, then we will get the Boolean function, which is in **standard sum of products** form after simplifying the K-map.

Similarly, if we consider the combination of inputs for which the Boolean function is „0“, then we will get the Boolean function, which is in **standard product of sums** form after simplifying the K-map.

### Example

Let us **simplify** the following Boolean function,  $f(W, X, Y, Z) = WX'Y' + WY + W'YZ'$  using K-map.

The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require **4 variable K-map**. The **4 variable K-map** with ones corresponding to the given product terms is shown in the following figure.

WX \ YZ	00	01	11	10
00				1
01				1
11			1	1
10	1	1	1	1

Here, 1s are placed in the following cells of K-map.

- The cells, which are common to the intersection of Row 4 and columns 1 & 2 are corresponding to the product term,  $WX'Y'$ .
- The cells, which are common to the intersection of Rows 3 & 4 and columns 3 & 4 are corresponding to the product term,  $WY$ .
- The cells, which are common to the intersection of Rows 1 & 2 and column 4 are corresponding to the product term,  $YZ'$ .

There are no possibilities of grouping either 16 adjacent ones or 8 adjacent ones. There are three possibilities of grouping 4 adjacent ones. After these three groupings, there is no single one left as ungrouped. So, we no need to check for grouping of 2 adjacent ones. The **4 variable K-map** with these three **groupings** is shown in the following figure.

WX \ YZ	00	01	11	10	
00				1	..... $YZ'$
01				1	
11			1	1	..... $WY$
10	1	1	1	1	$WX'$ .....

Here, we got three prime implicants  $WX'$ ,  $WY$  &  $YZ'$

Therefore, the **simplified Boolean function** is

$$f = WX' + WY + YZ'$$



# UNIT-III

## **Combinational Logic**

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of input variables, logic gates, and output variables

# Combinational Circuits

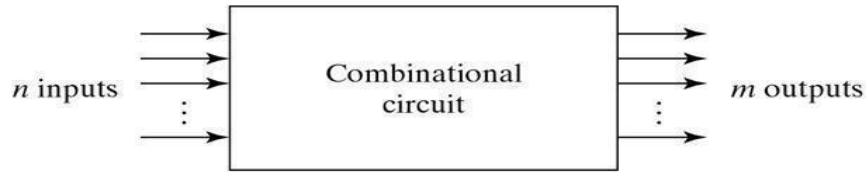


Fig. Block Diagram of Combinational Circuit

## Analysis procedure

To obtain the output Boolean functions from a logic diagram, proceed as follows:

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

## Design Procedure:

1. The problem is stated
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

## Binary Adder

The most basic arithmetic operation is addition. The circuit, which performs the addition of two binary numbers is known as **Binary adder**. First, let us implement an adder, which performs the addition of two bits.

## Half Adder

Half adder is a combinational circuit, which performs the addition of two binary numbers A and B are of **single bit**. It produces two outputs sum, S & carry, C.

The **Truth table** of Half adder is shown below.

Inputs		Outputs	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

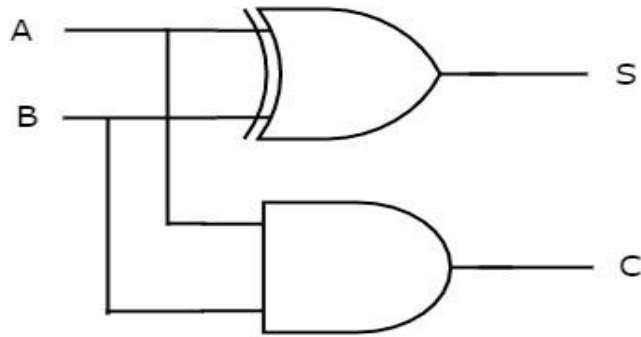
Let, sum, S is the Least significant bit and carry, C is the Most significant bit of the resultant sum. For first three combinations of inputs, carry, C is zero and the value of S will be either zero or one based on the **number of ones** present at the inputs. But, for last combination of inputs, carry, C is one and sum, S is zero, since the resultant sum is two.

From Truth table, we can directly write the **Boolean functions** for each output as

$$S=A\oplus B$$

$$C=AB$$

We can implement the above functions with 2-input Ex-OR gate & 2-input AND gate. The **circuit diagram** of Half adder is shown in the following figure.



In the above circuit, a two input Ex-OR gate & two input AND gate produces sum, S & carry, C respectively. Therefore, Half-adder performs the addition of two bits.

### Full Adder

Full adder is a combinational circuit, which performs the **addition of three bits** A, B and  $C_{in}$ . Where, A & B are the two parallel significant bits and  $C_{in}$  is the carry bit, which is generated from previous stage. This Full adder also produces two outputs sum, S & carry,  $C_{out}$ , which are similar to Half adder.

The **Truth table** of Full adder is shown below.

Inputs			Outputs	
A	B	$C_{in}$	$C_{out}$	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

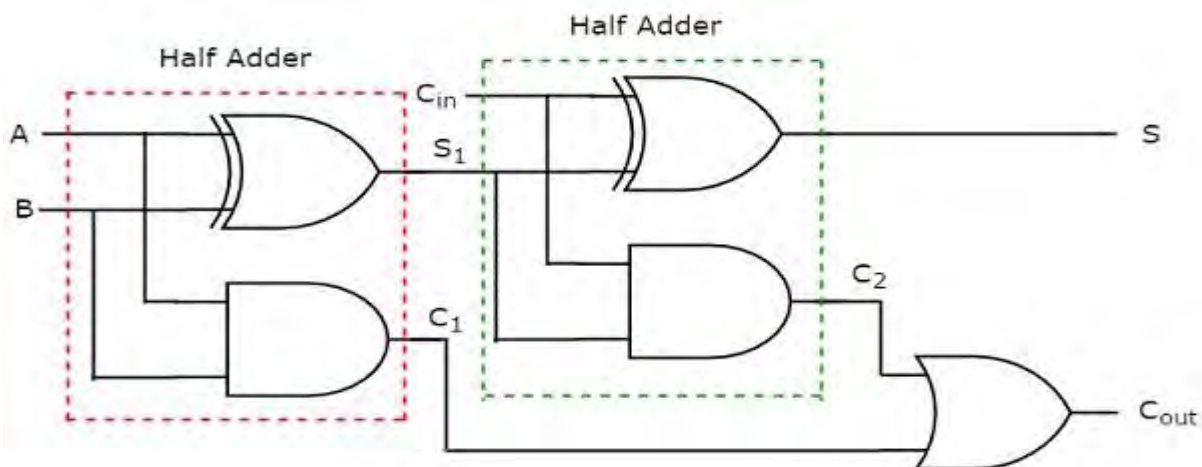
Let, sum, S is the Least significant bit and carry,  $C_{out}$  is the Most significant bit of resultant sum. It is easy to fill the values of outputs for all combinations of inputs in the truth table. Just count the **number of ones** present at the inputs and write the equivalent binary number at outputs. If  $C_{in}$  is equal to zero, then Full adder truth table is same as that of Half adder truth table.

We will get the following **Boolean functions** for each output after simplification.

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + (A \oplus B)C_{in}$$

The sum, S is equal to one, when odd number of ones present at the inputs. We know that Ex-OR gate produces an output, which is an odd function. So, we can use either two 2-input Ex-OR gates or one 3-input Ex-OR gate in order to produce sum, S. We can implement carry,  $C_{out}$  using two 2-input AND gates & one OR gate. The **circuit diagram** of Full adder is shown in the following figure.



This adder is called as **Full adder** because for implementing one Full adder, we require two Half adders and one OR gate. If  $C_{in}$  is zero, then Full adder becomes Half adder. We can verify it easily from the above circuit diagram or from the Boolean functions of outputs of Full adder.

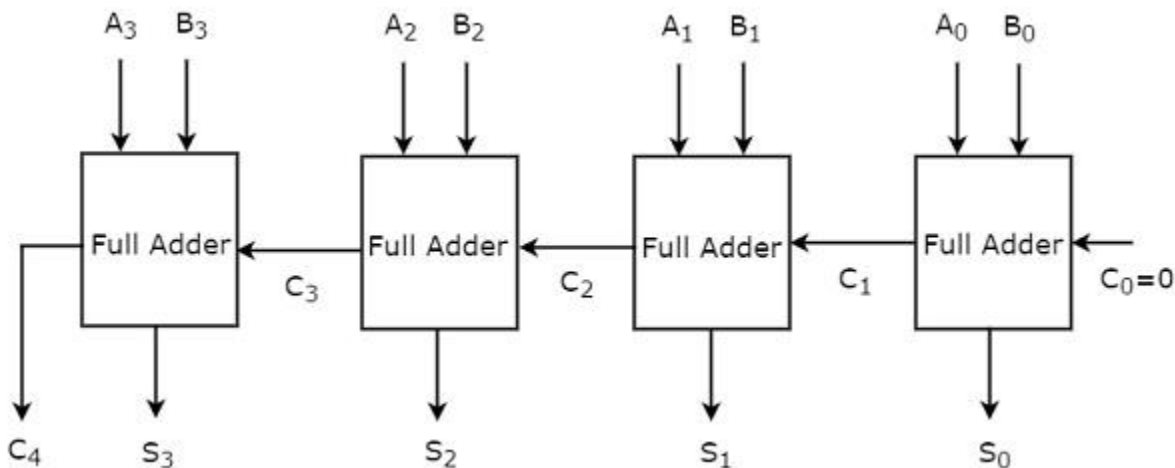


## 4-bit Binary Adder

The 4-bit binary adder performs the **addition of two 4-bit numbers**. Let the 4-bit binary numbers,  $A=A_3A_2A_1A_0$  and  $B=B_3B_2B_1B_0$ . We can implement 4-bit binary adder in one of the two following ways.

- Use one Half adder for doing the addition of two Least significant bits and three Full adders for doing the addition of three higher significant bits.
- Use four Full adders for uniformity. Since, initial carry  $C_{in}$  is zero, the Full adder which is used for adding the least significant bits becomes Half adder.

For the time being, we considered second approach. The **block diagram** of 4-bit binary adder is shown in the following figure.



Here, the 4 Full adders are cascaded. Each Full adder is getting the respective bits of two parallel inputs A & B. The carry output of one Full adder will be the carry input of subsequent higher order Full adder. This 4-bit binary adder produces the resultant sum having at most 5 bits. So, carry out of last stage Full adder will be the MSB.

In this way, we can implement any higher order binary adder just by cascading the required number of Full adders. This binary adder is also called as **ripple carry (binary) adder** because the carry propagates (ripples) from one stage to the next stage.

## Binary Subtractor

The circuit, which performs the subtraction of two binary numbers is known as **Binary subtractor**. We can implement Binary subtractor in following two methods.

- Cascade Full subtractors

- 2's complement method

In first method, we will get an n-bit binary subtractor by cascading „n“ Full subtractors. So, first you can implement Half subtractor and Full subtractor, similar to Half adder & Full adder. Then, you can implement an n-bit binary subtractor, by cascading „n“ Full subtractors. So, we will be having two separate circuits for binary addition and subtraction of two binary numbers.

In second method, we can use same binary adder for subtracting two binary numbers just by doing some modifications in the second input. So, internally binary addition operation takes place but, the output is resultant subtraction.

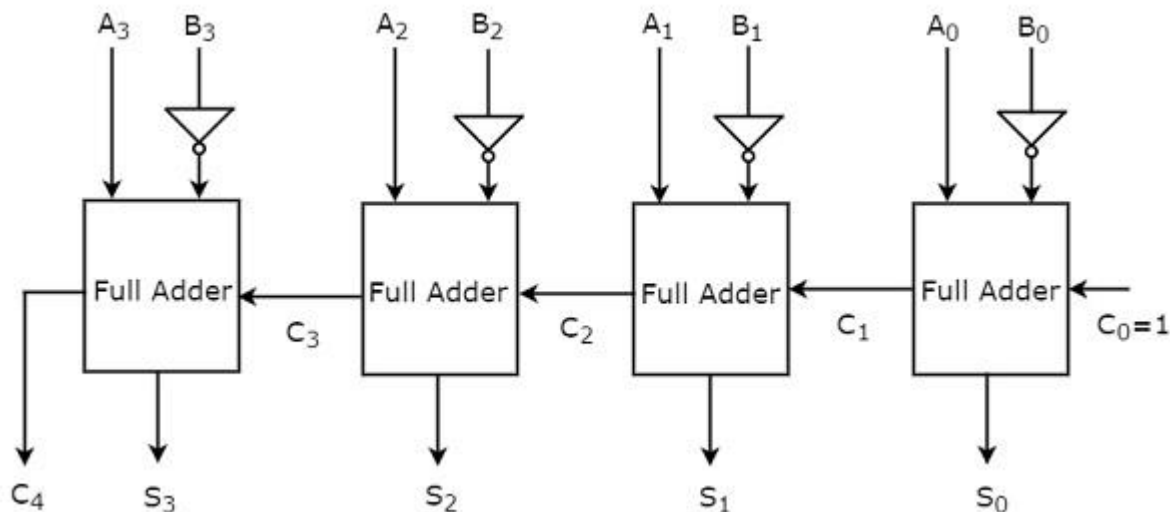
We know that the subtraction of two binary numbers A & B can be written as,

$$A - B = A + (2's \text{ complement of } B)$$

$$\Rightarrow A - B = A + (1's \text{ complement of } B) + 1$$

#### 4-bit Binary Subtractor:

The 4-bit binary subtractor produces the **subtraction of two 4-bit numbers**. Let the 4bit binary numbers,  $A = A_3A_2A_1A_0$  and  $B = B_3B_2B_1B_0$ . Internally, the operation of 4-bit Binary subtractor is similar to that of 4-bit Binary adder. If the normal bits of binary number A, complemented bits of binary number B and initial carry (borrow),  $C_{in}$  as one are applied to 4-bit Binary adder, then it becomes 4-bit Binary subtractor. The **block diagram** of 4-bit binary subtractor is shown in the following figure.



This 4-bit binary subtractor produces an output, which is having at most 5 bits. If Binary number A is greater than Binary number B, then MSB of the output is zero and the remaining bits hold the magnitude of A-B. If Binary number A is less than Binary number B, then MSB of the output is one. So, take the 2's complement of output in order to get the magnitude of A-B.

### Binary Adder / Subtractor

The circuit, which can be used to perform either addition or subtraction of two binary numbers at any time is known as **Binary Adder / subtractor**. Both, Binary adder and Binary subtractor contain a set of Full adders, which are cascaded. The input bits of binary number A are directly applied in both Binary adder and Binary subtractor.

There are two differences in the inputs of Full adders that are present in Binary adder and Binary subtractor.

- The input bits of binary number B are directly applied to Full adders in Binary adder, whereas the complemented bits of binary number B are applied to Full adders in Binary subtractor.
- The initial carry,  $C_0 = 0$  is applied in 4-bit Binary adder, whereas the initial carry (borrow),  $C_0 = 1$  is applied in 4-bit Binary subtractor.

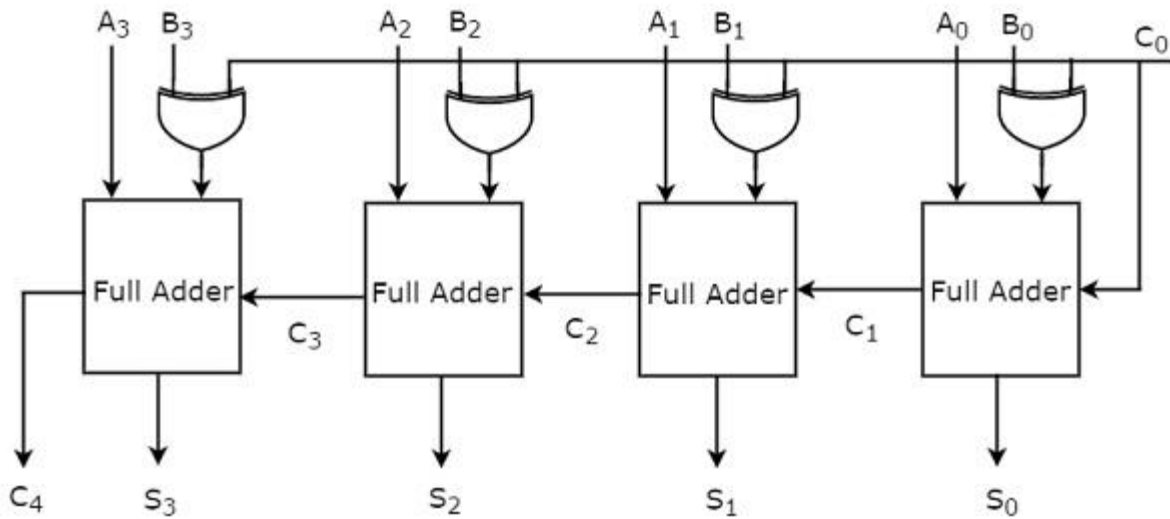
We know that a **2-input Ex-OR gate** produces an output, which is same as that of first input when other input is zero. Similarly, it produces an output, which is complement of first input when other input is one.

Therefore, we can apply the input bits of binary number B, to 2-input Ex-OR gates. The other input to all these Ex-OR gates is  $C_0$ . So, based on the value of  $C_0$ , the Ex-OR gates produce either the normal or complemented bits of binary number B.

### 4-bit Binary Adder / Subtractor

The 4-bit binary adder / subtractor produces either the addition or the subtraction of two 4-bit numbers based on the value of initial carry or borrow,  $C_0$ . Let the 4-bit binary numbers,  $A=A_3A_2A_1A_0$  and  $B=B_3B_2B_1B_0$ . The operation of 4-bit Binary adder / subtractor is similar to that of 4-bit Binary adder and 4-bit Binary subtractor.

Apply the normal bits of binary numbers A and B & initial carry or borrow,  $C_0$  from externally to a 4-bit binary adder. The **block diagram** of 4-bit binary adder / subtractor is shown in the following figure.



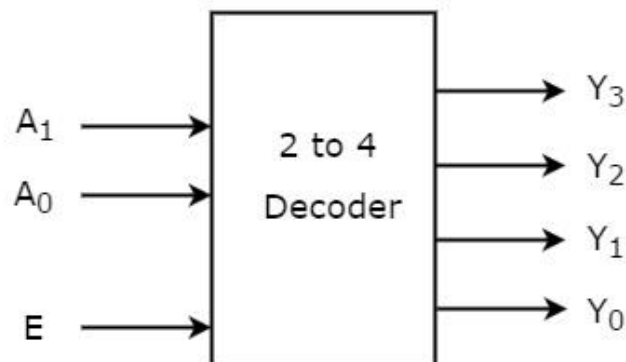
If initial carry,  $C_0$  is zero, then each full adder gets the normal bits of binary numbers A & B. So, the 4-bit binary adder / subtractor produces an output, which is the **addition of two binary numbers** A & B.

If initial borrow,  $C_0$  is one, then each full adder gets the normal bits of binary number A & complemented bits of binary number B. So, the 4-bit binary adder / subtractor produces an output, which is the **subtraction of two binary numbers** A & B.

**Decoder** is a combinational circuit that has „n“ input lines and maximum of  $2^n$  output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of „n“ input variables (lines), when it is enabled.

### 2 to 4 Decoder:

Let 2 to 4 Decoder has two inputs A<sub>1</sub> & A<sub>0</sub> and four outputs Y<sub>3</sub>, Y<sub>2</sub>, Y<sub>1</sub> & Y<sub>0</sub>. The **block diagram** of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be „1“ for each combination of inputs when enable, E is „1“. The **Truth table** of 2 to 4 decoder is shown below.

Enable	Inputs		Outputs			
E	A <sub>1</sub>	A <sub>0</sub>	Y <sub>3</sub>	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the **Boolean functions** for each output as

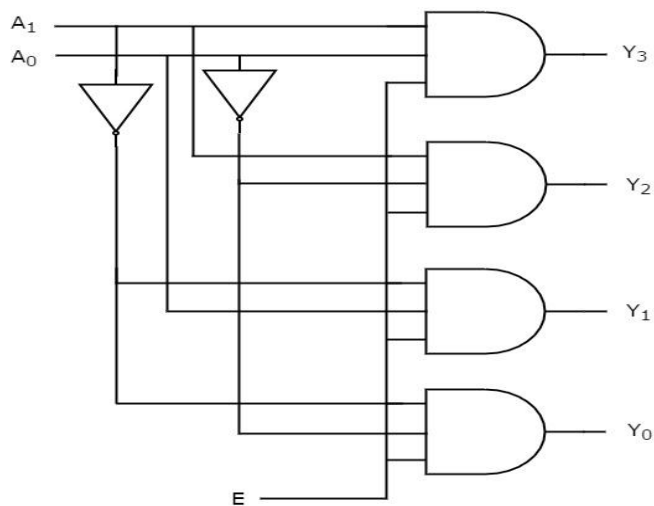
$$Y_3 = E \cdot A_1 \cdot A_0 \quad Y_3 = E \cdot A_1 \cdot A_0$$

$$Y_2 = E \cdot A_1 \cdot A_0' \quad Y_2 = E \cdot A_1 \cdot A_0'$$

$$Y_1 = E \cdot A_1' \cdot A_0 \quad Y_1 = E \cdot A_1' \cdot A_0$$

$$Y_0 = E \cdot A_1' \cdot A_0'$$

The **circuit diagram** of 2 to 4 decoder is shown in the following figure.



Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables  $A_1$  &  $A_0$ , when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

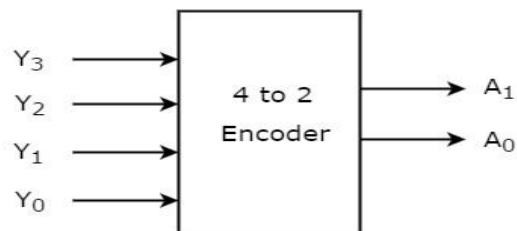
Similarly, 3 to 8 decoder produces eight min terms of three input variables  $A_2$ ,  $A_1$  &  $A_0$  and 4 to 16 decoder produces sixteen min terms of four input variables  $A_3$ ,  $A_2$ ,  $A_1$  &  $A_0$ .

### Encoder:

An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of  $2^n$  input lines and „n“ output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes  $2^n$  input lines with „n“ bits. It is optional to represent the enable signal in encoders.

### 4 to 2 Encoder

Let 4 to 2 Encoder has four inputs  $Y_3$ ,  $Y_2$ ,  $Y_1$  &  $Y_0$  and two outputs  $A_1$  &  $A_0$ . The **block diagram** of 4 to 2 Encoder is shown in the following figure.



The **Truth table** of 4 to 2 encoder is shown below.

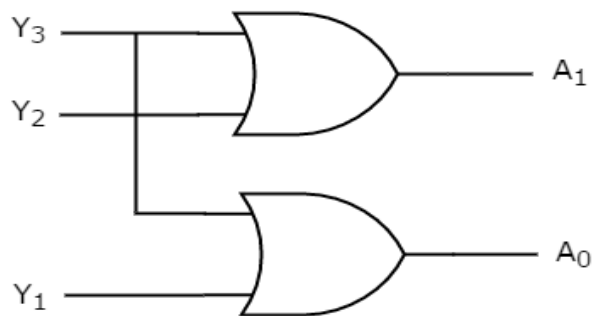
Inputs				Outputs	
$Y_3$	$Y_2$	$Y_1$	$Y_0$	$A_1$	$A_0$
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From Truth table, we can write the **Boolean functions** for each output as

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

We can implement the above two Boolean functions by using two input OR gates. The **circuit diagram** of 4 to 2 encoder is shown in the following figure.

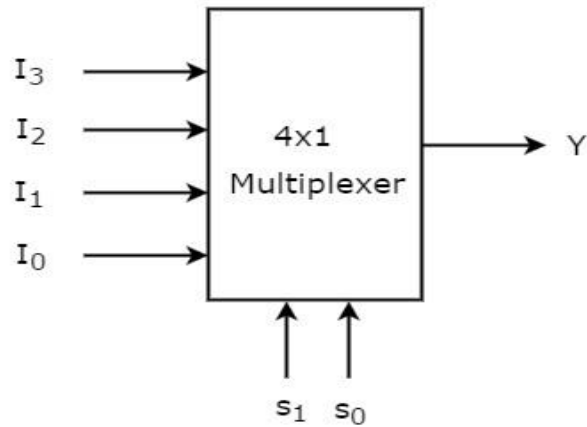


**Multiplexer** is a combinational circuit that has maximum of  $2^n$  data inputs, „n“ selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

Since there are „n“ selection lines, there will be  $2^n$  possible combinations of zeros and ones. So, each combination will select only one data input.

### 4x1 Multiplexer:

4x1 Multiplexer has four data inputs  $I_3$ ,  $I_2$ ,  $I_1$  &  $I_0$ , two selection lines  $s_1$  &  $s_0$  and one output  $Y$ . The **block diagram** of 4x1 Multiplexer is shown in the following figure.



One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

Selection Lines		Output
$S_1$	$S_0$	$Y$
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

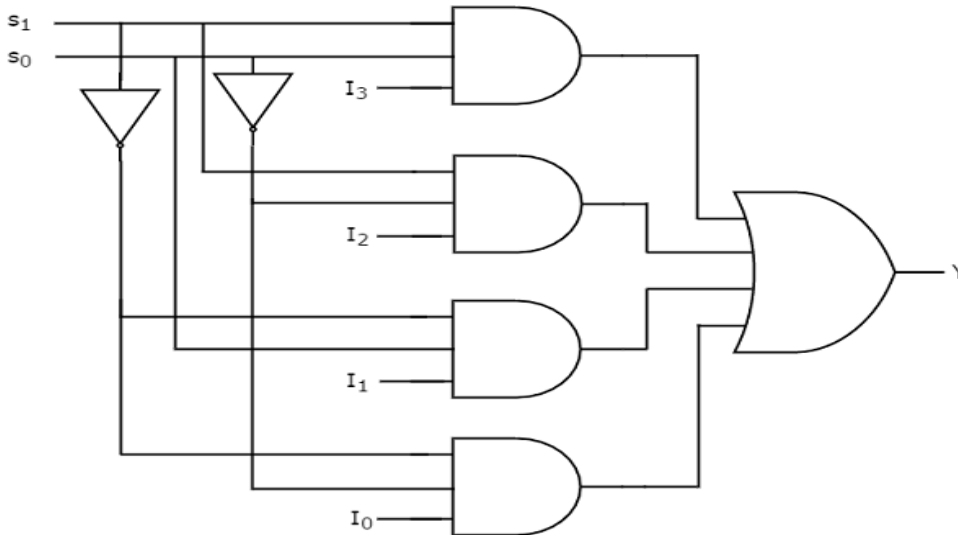
From Truth table, we can directly write the **Boolean function** for output,  $Y$  as

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_2$$

We can implement this Boolean function using Inverters, AND gates & OR gate.



The **circuit diagram** of 4x1 multiplexer is shown in the following figure.

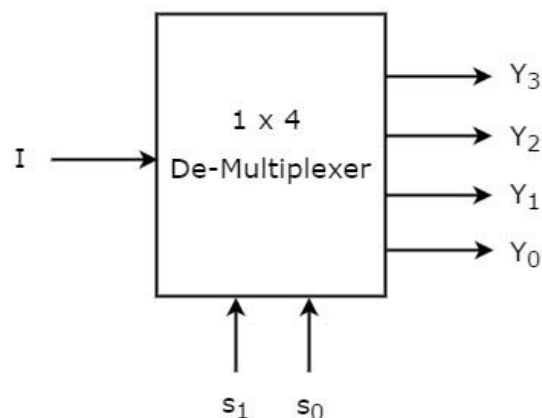


**De-Multiplexer** is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, „n“ selection lines and maximum of  $2^n$  outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are „n“ selection lines, there will be  $2^n$  possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.

#### 1x4 De-Multiplexer :

1x4 De-Multiplexer has one input I, two selection lines,  $s_1$  &  $s_0$  and four outputs  $Y_3$ ,  $Y_2$ ,  $Y_1$  &  $Y_0$ . The **block diagram** of 1x4 De-Multiplexer is shown in the following figure.



The single input „I“ will be connected to one of the four outputs,  $Y_3$  to  $Y_0$  based on the values of selection lines  $s_1$  &  $s_0$ . The **Truth table** of 1x4 De-Multiplexer is shown below.

Selection Inputs		Outputs			
$S_1$	$S_0$	$Y_3$	$Y_2$	$Y_1$	$Y_0$
0	0	0	0	0	<b>I</b>
0	1	0	0	<b>I</b>	0
1	0	0	<b>I</b>	0	0
1	1	<b>I</b>	0	0	0

From the above Truth table, we can directly write the **Boolean functions** for each output as

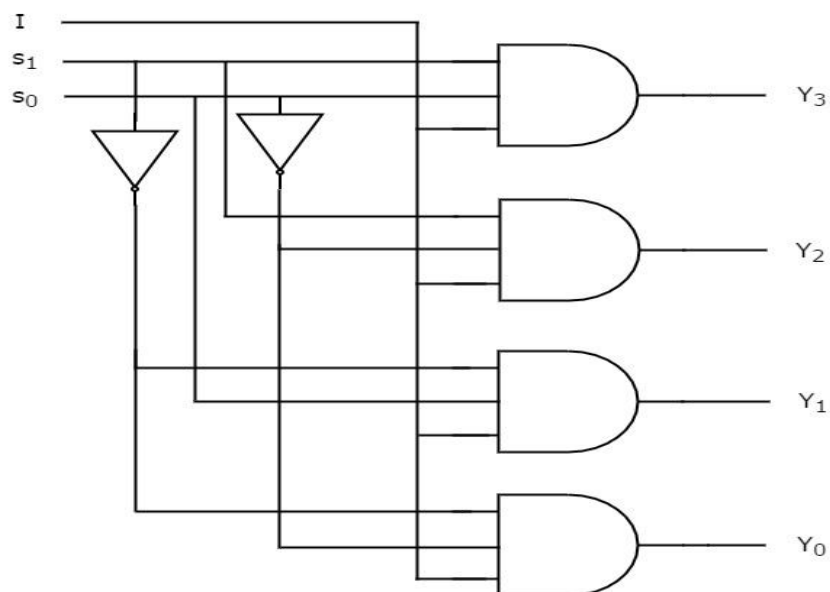
$$Y_3 = s_1 s_0 I \quad Y_3 = s_1 s_0 I$$

$$Y_2 = s_1 s_0' I \quad Y_2 = s_1 s_0' I$$

$$Y_1 = s_1' s_0 I \quad Y_1 = s_1' s_0 I$$

$$Y_0 = s_1' s_0' I \quad Y_0 = s_1' s_0' I$$

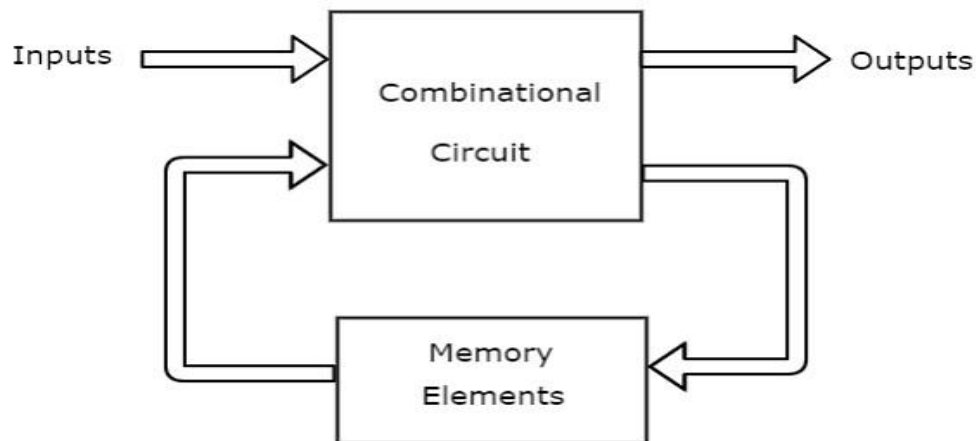
We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure.



# UNIT -IV

## Sequential Circuits

The following figure shows the **block diagram** of sequential circuit.



This sequential circuit contains a set of inputs and output(s). The output(s) of sequential circuit depends not only on the combination of present inputs but also on the previous output(s). Previous output is nothing but the **present state**. Therefore, sequential circuits contain combinational circuits along with memory (storage) elements. Some sequential circuits may not contain combinational circuits, but only memory elements.

Following table shows the **differences** between combinational circuits and sequential circuits.

Combinational Circuits	Sequential Circuits
Outputs depend only on present inputs.	Outputs depend on both present inputs and present state.
Feedback path is not present.	Feedback path is present.
Memory elements are not required.	Memory elements are required.
Clock signal is not required.	Clock signal is required.
Easy to design.	Difficult to design.

## Types of Sequential Circuits

Following are the two types of sequential circuits –

- Asynchronous sequential circuits
- Synchronous sequential circuits

### Asynchronous sequential circuits

If some or all the outputs of a sequential circuit do not change (affect) with respect to active transition of clock signal, then that sequential circuit is called as **Asynchronous sequential circuit**. Therefore, most of the outputs of asynchronous sequential circuits are **not in synchronous** with either only positive edges or only negative edges of clock signal.

### Synchronous sequential circuits

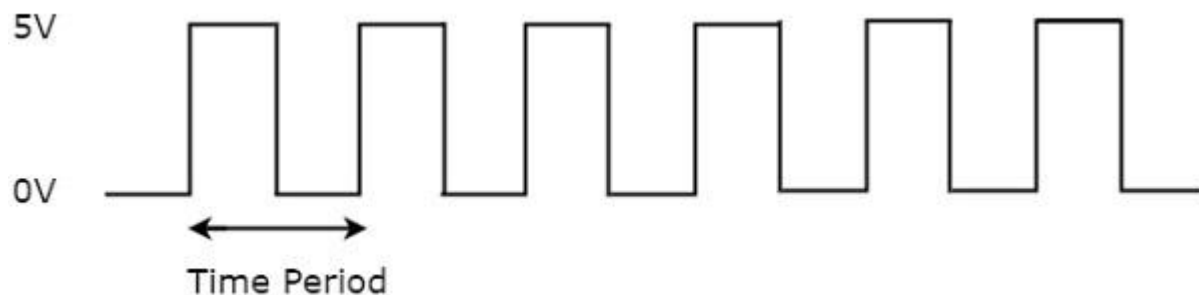
If all the outputs of a sequential circuit change (affect) with respect to active transition of clock signal, then that sequential circuit is called as **Synchronous sequential circuit**. That means, all the outputs of synchronous sequential circuits change (affect) at the same time. Therefore, the outputs of synchronous sequential circuits are in synchronous with either only positive edges or only negative edges of clock signal.

### Clock Signal and Triggering

In this section, let us discuss about the clock signal and types of triggering one by one.

#### Clock signal

Clock signal is a periodic signal and its ON time and OFF time need not be the same. We can represent the clock signal as a **square wave**, when both its ON time and OFF time are same. This clock signal is shown in the following figure.



This signal stays at logic High (5V) for some time and stays at logic Low (0V) for equal amount of time. This pattern repeats with some time period. In this case, the **time period** will be equal to either twice of ON time or twice of OFF time.

## Types of Triggering

Following are the two possible types of triggering that are used in sequential circuits.

- Level triggering
- Edge triggering

### Level triggering

There are two levels, namely logic High and logic Low in clock signal. Following are the two **types of level triggering**.

- Positive level triggering
- Negative level triggering

### Edge triggering

There are two types of transitions that occur in clock signal. That means, the clock signal transitions either from Logic Low to Logic High or Logic High to Logic Low.

Following are the two **types of edge triggering** based on the transitions of clock signal.

- Positive edge triggering
- Negative edge triggering

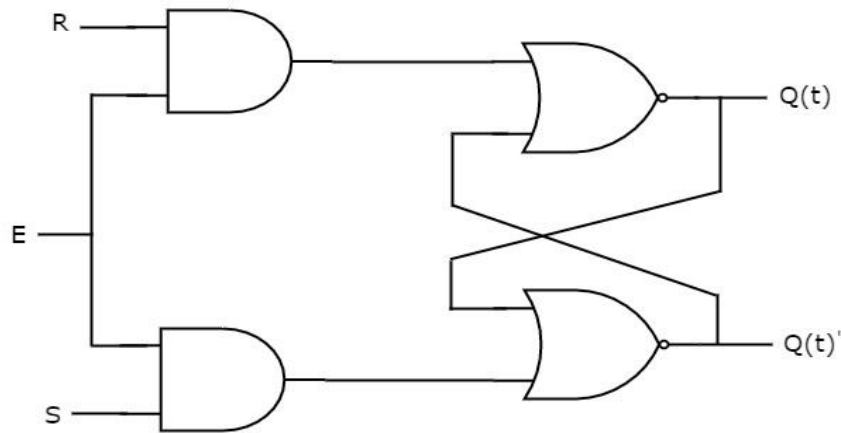
There are two types of memory elements based on the type of triggering that is suitable to operate it.

- Latches
- Flip-flops

Latches operate with enable signal, which is **level sensitive**. Whereas, flip-flops are edge sensitive. We will discuss about flip-flops in next chapter. Now, let us discuss about SR Latch & D Latch one by one.

### SR Latch

SR Latch is also called as **Set Reset Latch**. This latch affects the outputs as long as the enable, E is maintained at „1“. The **circuit diagram** of SR Latch is shown in the following figure.



This circuit has two inputs S & R and two outputs  $Q(t)$  &  $Q(t)'$ . The **upper NOR gate** has two inputs R & complement of present state,  $Q(t)'$  and produces next state,  $Q(t+1)$  when enable, E is „1“.

Similarly, the **lower NOR gate** has two inputs S & present state,  $Q(t)$  and produces complement of next state,  $Q(t+1)'$  when enable, E is „1“.

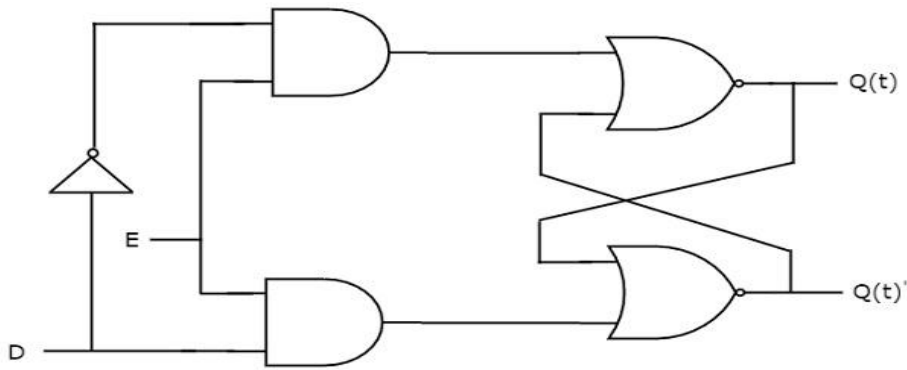
We know that a **2-input NOR gate** produces an output, which is the complement of another input when one of the input is „0“. Similarly, it produces „0“ output, when one of the input is „1“.

The following table shows the **state table** of SR latch.

S	R	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	-

### D Latch

There is one drawback of SR Latch. That is the next state value can't be predicted when both the inputs S & R are one. So, we can overcome this difficulty by D Latch. It is also called as Data Latch. The **circuit diagram** of D Latch is shown in the following figure.



The following table shows the **state table** of D latch.

D	Q(t+1)
0	0
1	1

In first method, **cascade two latches** in such a way that the first latch is enabled for every positive clock pulse and second latch is enabled for every negative clock pulse. So that the combination of these two latches become a flip-flop.

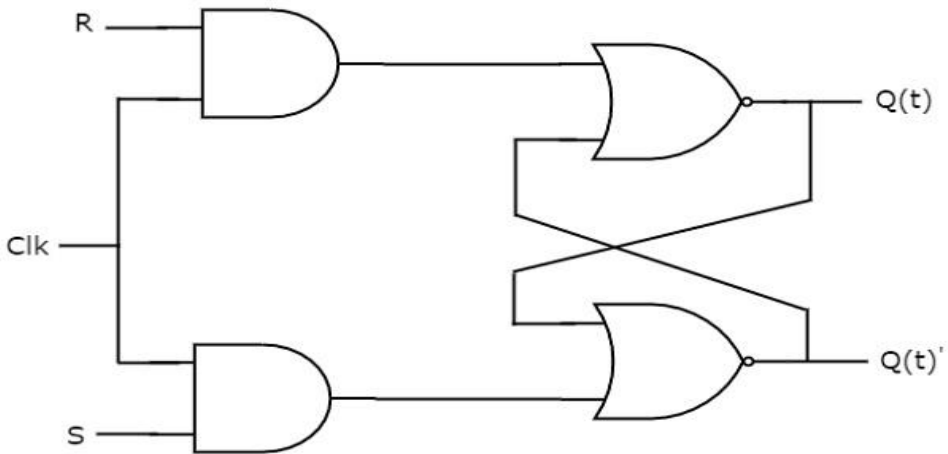
In second method, we can directly implement the flip-flop, which is edge sensitive. In this chapter, let us discuss the following **flip-flops** using second method.

- SR Flip-Flop
- D Flip-Flop
- JK Flip-Flop
- T Flip-Flop

### SR Flip-Flop

SR flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, SR latch operates with enable signal. The **circuit diagram** of SR flip-flop is shown in the following figure.





The following table shows the **state table** of SR flip-flop.

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

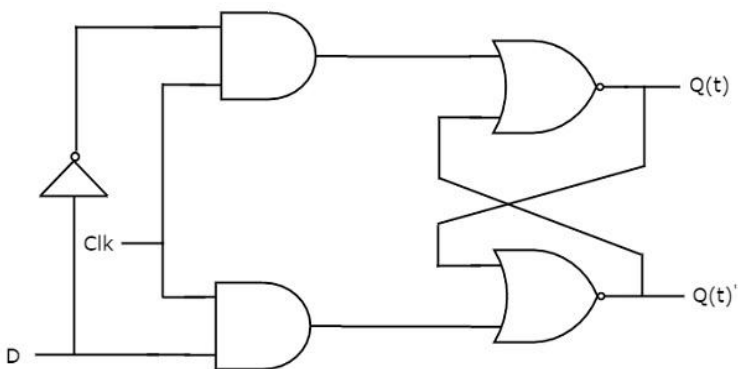
The following table shows the **characteristic table** of SR flip-flop.

Present Inputs		Present State	Next State
S	R	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0

0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

### D Flip-Flop

D flip-flop operates with only positive clock transitions or negative clock transitions. Whereas, D latch operates with enable signal. That means, the output of D flip-flop is insensitive to the changes in the input, D except for active transition of the clock signal. The **circuit diagram** of D flip-flop is shown in the following figure.



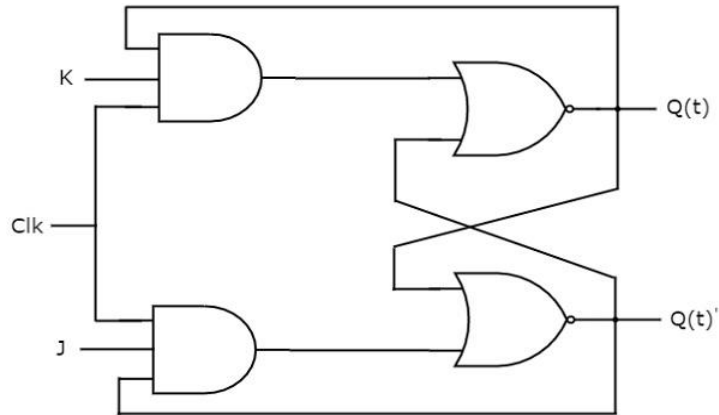
The following table shows the **state table** of D flip-flop.

D	Q(t+1)
0	0
1	1

From the above state table, we can directly write the next state equation as  $Q(t+1)=D$

## JK Flip-Flop

JK flip-flop is the modified version of SR flip-flop. It operates with only positive clock transitions or negative clock transitions. The **circuit diagram** of JK flip-flop is shown in the following figure.



The following table shows the **state table** of JK flip-flop.

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q(t)'

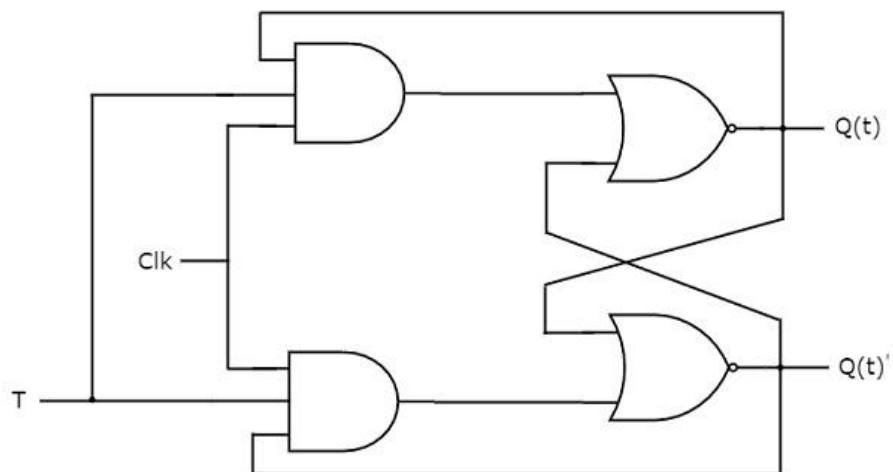
The following table shows the **characteristic table** of JK flip-flop.

Present Inputs		Present State	Next State
J	K	Q(t)	Q(t+1)
0	0	0	0

0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

### T Flip-Flop

T flip-flop is the simplified version of JK flip-flop. It is obtained by connecting the same input „T“ to both inputs of JK flip-flop. It operates with only positive clock transitions or negative clock transitions. The **circuit diagram** of T flip-flop is shown in the following figure.



The following table shows the **state table** of T flip-flop.

D	Q(t+1)
0	Q(t)
1	Q(t)'

The following table shows the **characteristic table** of T flip-flop.

Inputs	Present State	Next State
T	Q(t)	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

From the above characteristic table, we can directly write the **next state equation** as

$$Q(t+1) = T'Q(t) + TQ(t)'Q(t+1) = T'Q(t) + TQ(t)'$$

$$\Rightarrow Q(t+1) = T \oplus Q(t)$$

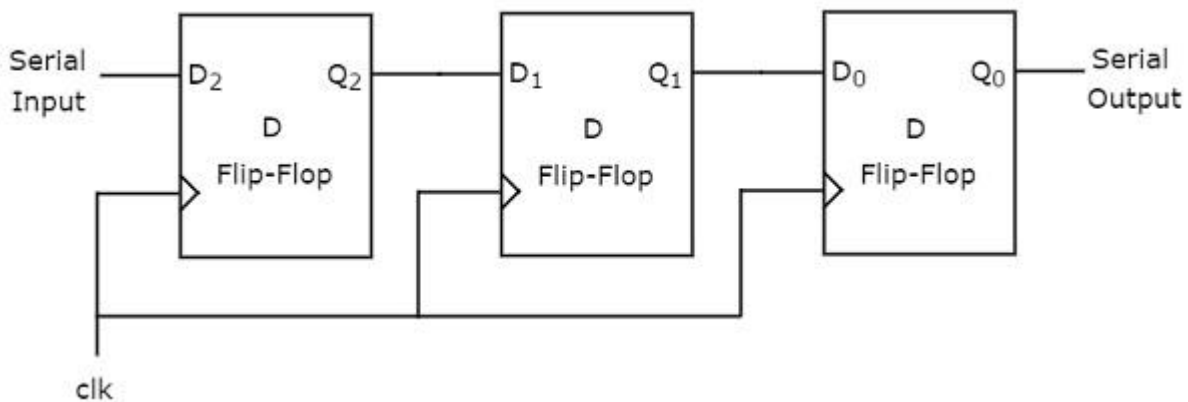
### shift register:

If the register is capable of shifting bits either towards right hand side or towards left hand side is known as **shift register**. An „N“ bit shift register contains „N“ flip-flops. Following are the four types of shift registers based on applying inputs and accessing of outputs.

- Serial In - Serial Out shift register
- Serial In - Parallel Out shift register
- Parallel In - Serial Out shift register
- Parallel In - Parallel Out shift register

### Serial In - Serial Out (SISO) Shift Register

The shift register, which allows serial input and produces serial output is known as Serial In – Serial Out (**SISO**) shift register. The **block diagram** of 3-bit SISO shift register is shown in the following figure.

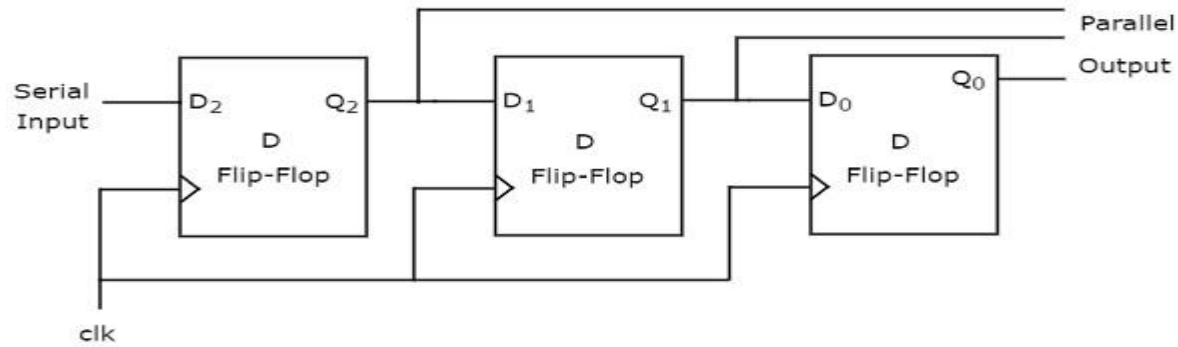


This block diagram consists of three D flip-flops, which are **cascaded**. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we can receive the bits serially from the output of right most D flip-flop. Hence, this output is also called as **serial output**.

### Serial In - Parallel Out (SIPO) Shift Register

The shift register, which allows serial input and produces parallel output is known as Serial In – Parallel Out (**SIPO**) shift register. The **block diagram** of 3-bit SIPO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**. For every positive edge triggering of clock signal, the data shifts from one stage to the next. In this case, we can access the outputs of each D flip-flop in parallel. So, we will get **parallel outputs** from this shift register.

### Design of Asynchronous and Synchronous Circuits:

The synchronous sequential circuits change (affect) their states for every positive (or negative) transition of the clock signal based on the input. So, this behavior of synchronous sequential circuits can be represented in the graphical form and it is known as **state diagram**.

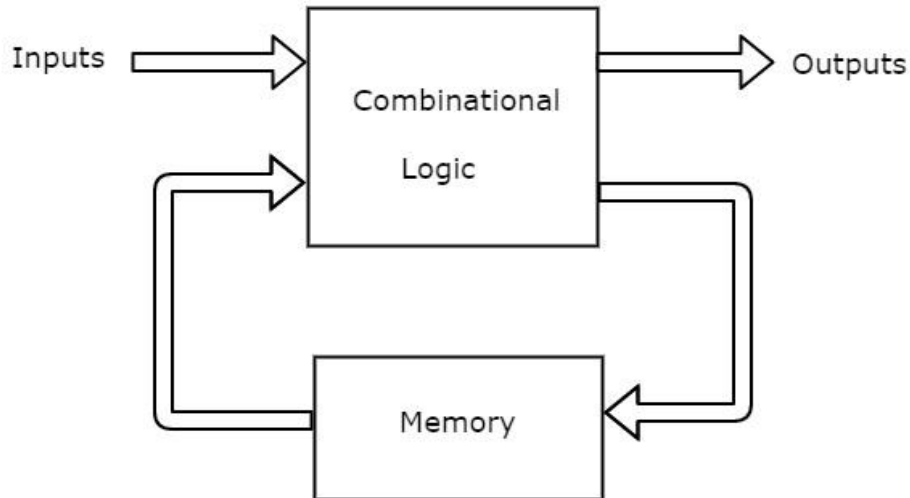
A synchronous sequential circuit is also called as **Finite State Machine (FSM)**, if it has finite number of states. There are two types of FSMs.

- Mealy State Machine
- Moore State Machine

Now, let us discuss about these two state machines one by one.

### Mealy State Machine

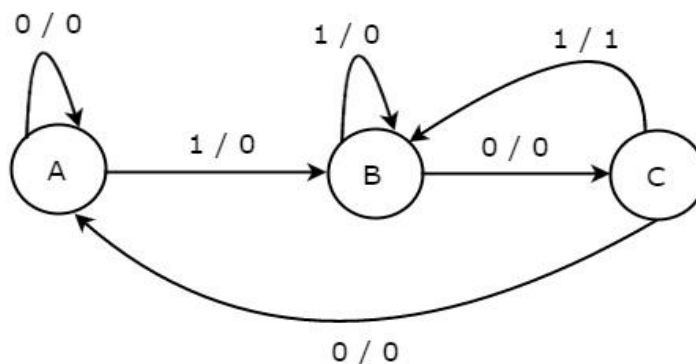
A Finite State Machine is said to be Mealy state machine, if outputs depend on both present inputs & present states. The **block diagram** of Mealy state machine is shown in the following figure.



As shown in figure, there are two parts present in Mealy state machine. Those are combinational logic and memory. Memory is useful to provide some or part of previous outputs (**present states**) as inputs of combinational logic.

So, based on the present inputs and present states, the Mealy state machine produces outputs. Therefore, the outputs will be valid only at positive (or negative) transition of the clock signal.

The **state diagram** of Mealy state machine is shown in the following figure.

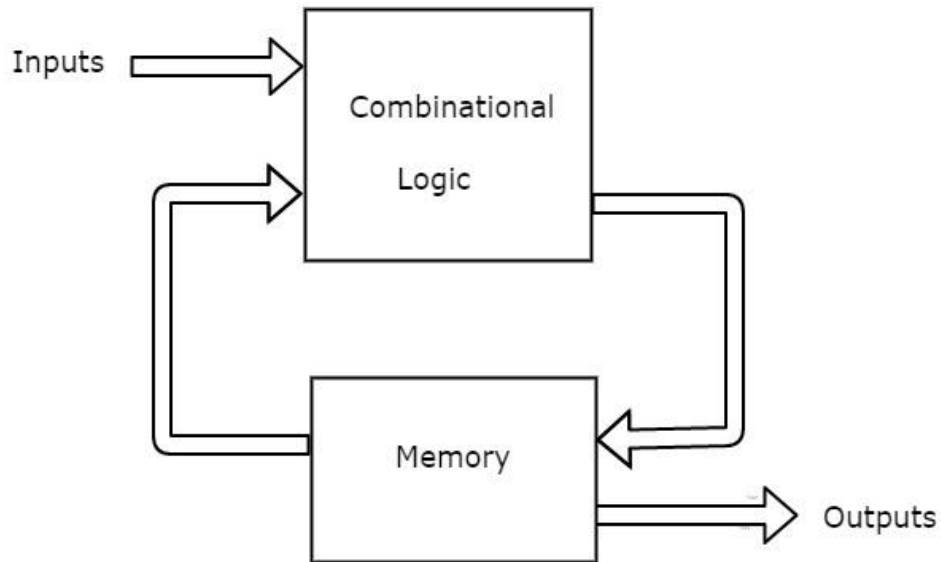


In the above figure, there are three states, namely A, B & C. These states are labelled inside the circles & each circle corresponds to one state. Transitions between these states are represented with directed lines. Here, 0 / 0, 1 / 0 & 1 / 1 denotes **input / output**. In the above figure, there are two transitions from each state based on the value of input, x.

### Moore State Machine

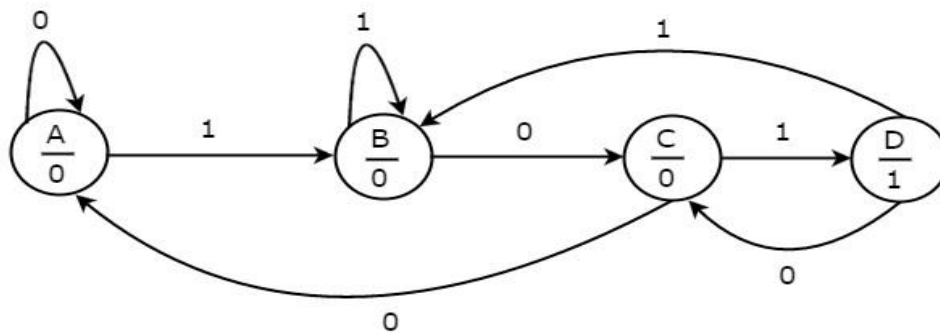
A Finite State Machine is said to be Moore state machine, if outputs depend only on present states. The **block diagram** of Moore state machine is shown in the following figure.





As shown in figure, there are two parts present in Moore state machine. Those are combinational logic and memory. In this case, the present inputs and present states determine the next states. So, based on next states, Moore state machine produces the outputs. Therefore, the outputs will be valid only after transition of the state.

The **state diagram** of Moore state machine is shown in the following figure.



In the above figure, there are four states, namely A, B, C & D. These states and the respective outputs are labeled inside the circles. Here, only the input value is labeled on each transition. In the above figure, there are two transitions from each state based on the value of input, x.

# UNIT -V

# Memory:

## READ-ONLY MEMORY

Read-only memory (ROM) is a type of storage medium that permanently stores data on personal computers (PCs) and other electronic devices. It contains the programming needed to start a PC, which is essential for boot-up; it performs major input/output tasks and holds programs or software instructions.

Because ROM is read-only, it cannot be changed; it is permanent and non-volatile, meaning it also holds its memory even when power is removed. By contrast, random access memory (RAM) is volatile; it is lost when power is removed.

There are numerous ROM chips located on the motherboard and a few on expansion boards. The chips are essential for the basic input/output system (BIOS), boot up, reading and writing to peripheral devices, basic data management and the software for basic processes for certain utilities.

## RANDOM ACCESS MEMORY

RAM (random access memory) is the place in a computing device where the operating system (OS), application programs and data in current use are kept so they can be quickly reached by the device's processor. RAM is much faster to read from and write to than other kinds of storage in a computer, such as a hard disk drive (HDD), solid-state drive (SSD) or optical drive. Data remains in RAM as long as the computer is running. When the computer is turned off, RAM loses its data. When the computer is turned on again, the OS and other files are once again loaded into RAM, usually from an HDD or SSD.

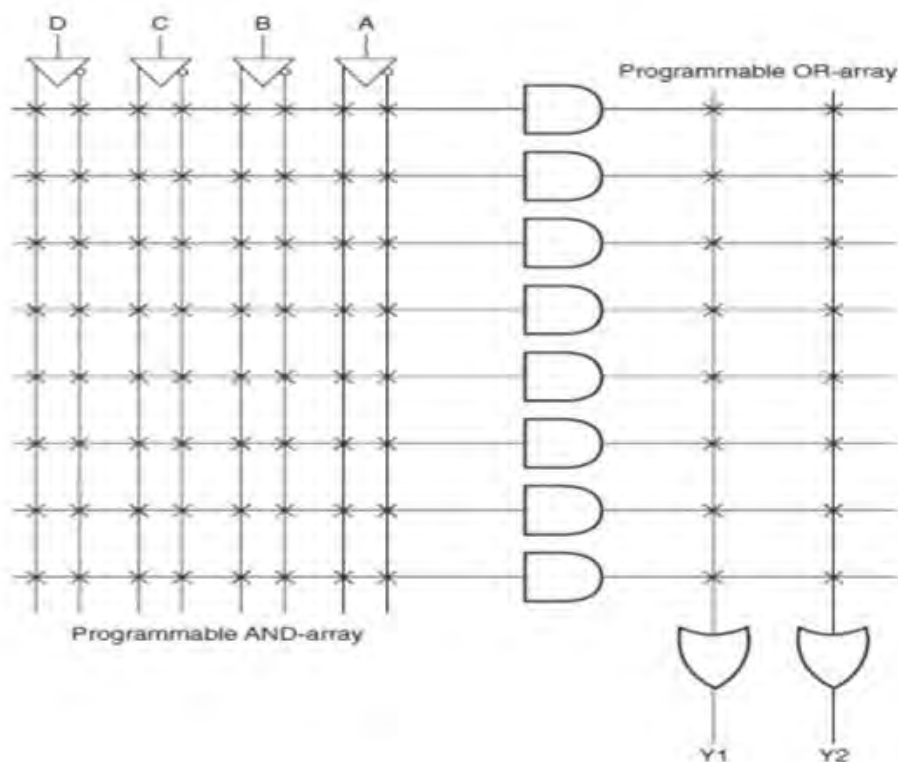
## RAM TYPES

**(1)Dynamic random access memory:** DRAM is what makes up the typical computing device RAM and, as noted above, requires constant power to hold on to stored data.

**(2)Static random access memory.:**SRAM doesn't need constant power to hold on to data, but the way the memory chips are made means they are much larger and thousands of times more expensive than an equivalent amount of DRAM. However, SRAM is significantly faster than DRAM. The price and speed differences mean SRAM is mainly used in small amounts as cache memory inside a device's processor.

## PROGRAMMABLE LOGIC ARRAY

A programmable logic array (PLA) has a programmable AND array at the inputs and programmable OR array at the outputs. The PLA has a programmable AND array instead of hard-wired AND array. The number of AND gates in the programmable AND array are usually much less and the number of inputs of each of the OR gates equal to the number of AND gates. The OR gate generates an arbitrary Boolean function of minterms equal to the number of AND gates. Figure below shows the PLA architecture with four input lines, a programmable array of eight AND gates at the input and a programmable array of two OR gates at the output.



### ADVANTAGES

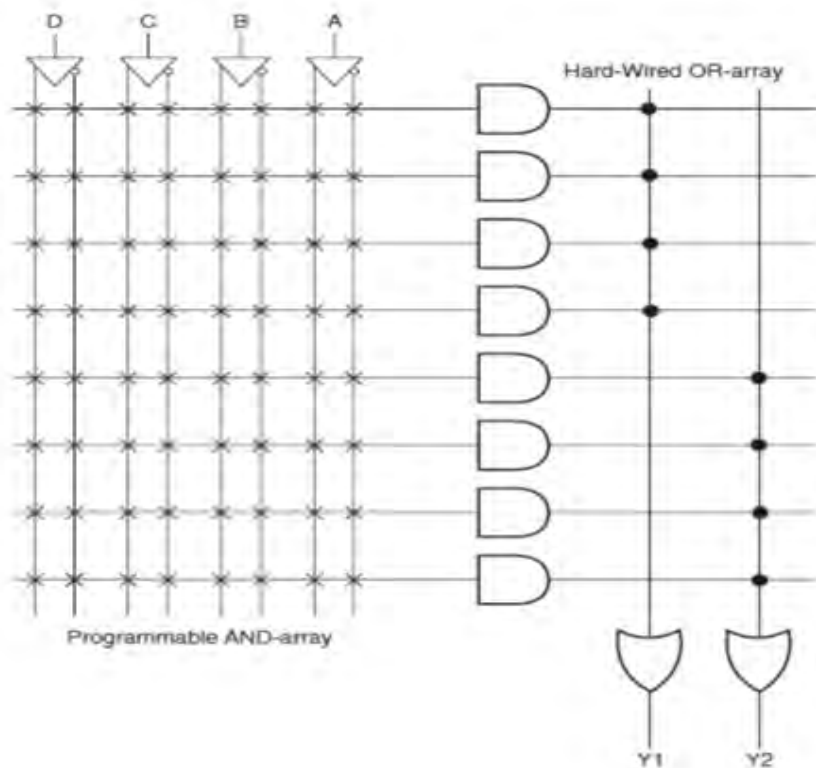
PLA architecture more efficient than a PROM.

### DISADVANTAGE

PLA architecture has two sets of programmable fuses due to which PLA devices are difficult to manufacture, program and test.

## PROGRAMMABLE ARRAY LOGIC

Programmable array logic (PAL) has a programmable AND array at the input and a fixed OR array at the output. The programmable AND array of a PAL architecture is same as that of the PLA architecture. The number of programmable AND gates in PAL architecture are smaller than the number of minterms. The OR array is fixed and the AND outputs are divided between OR gates.



## Memory decoding:

**Memory decoding** :n The equivalent logic of a binary cell that stores one bit of information is shown below. Read/Write = 0, select = 1, input data to S-R latch Read/Write = 1, select = 1, output data from S-R latch.

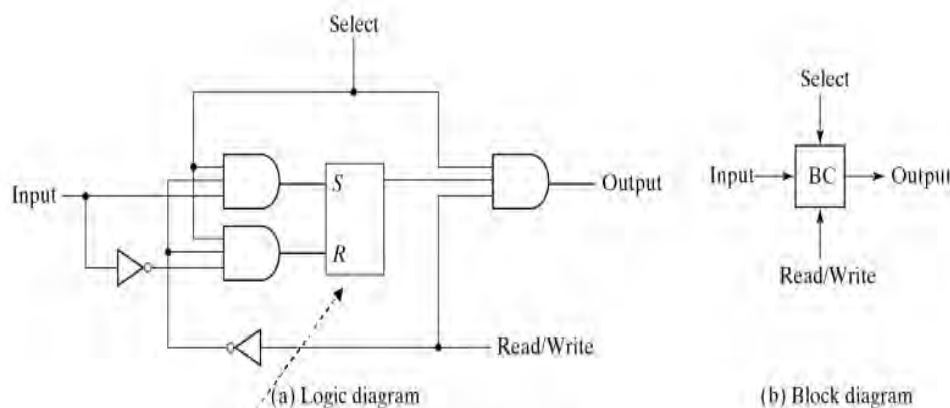


Fig. 7-5 Memory Cell

## Cache memory:

Cache memory is a small amount of fast memory

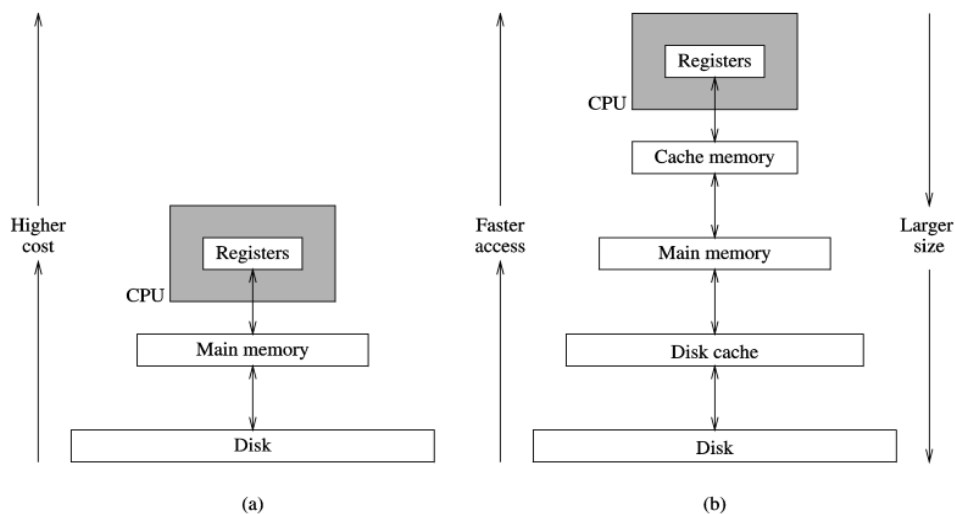
- \* Placed between two levels of memory hierarchy

- » To bridge the gap in access times

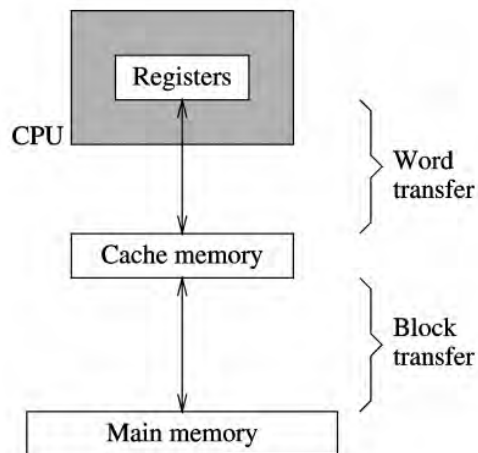
- Between processor and main memory (our focus)

- Between main memory and disk (disk cache)

- \* Expected to behave like a large amount of fast memory



- Transfer between main memory and cache
  - \* In units of blocks
  - \* Implements spatial locality
- Transfer between main memory and cache
  - \* In units of words
- Need policies for
  - \* Block placement
  - \* Mapping function
  - \* Block replacement
  - \* Write policies



- Determines how memory blocks are mapped to cache lines
- Three types
  - \* Direct mapping
    - » Specifies a single cache line for each memory block
  - \* Set-associative mapping
    - » Specifies a set of cache lines for each memory block
  - \* Associative mapping
    - » No restrictions
      - Any cache line can be used for any memory block

## Levels of memory Hierarchy:

