

# Unit.1 Introduction

# What is Signal?

---

- ✓ A signal is a physical quantity (sound, light, voltage, current) that carries **information**
  - The power cable supplies power but no information (not a signal)
  - A USB cable carries information (files)
- ✓ Examples of quantities used as digital information signals
  - Voltage: 5V (logic 1), 0V (logic 0) in digital circuits
  - Magnetic field orientation in magnetic hard disks
  - Pits and lands on the CD surface reflect the light from the laser differently, and that difference is encoded as binary data

# Analog Vs Digital

---

## Analog Signal

- Continuous
- Infinite range of values
- More exact values, but more difficult to work with

## Digital Signal

- Discrete
- Finite range of values (2).
- Not as exact as analog, but easier to work with

### Example:

A digital thermostat in a room displays a temperature of 30 degree(in celcius). An analog thermometer measures the room temperature at 72.482 degree(in celcius). The analog value is continuous and more accurate, but the digital value is more than adequate for the application and significantly easier to process electronically.

# Example of Analog Vs Digital System

---



Digital advantages:  
Battery life  
Programmability  
Accuracy



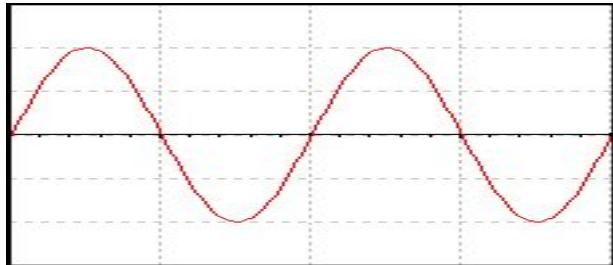
# The World is Analog

---

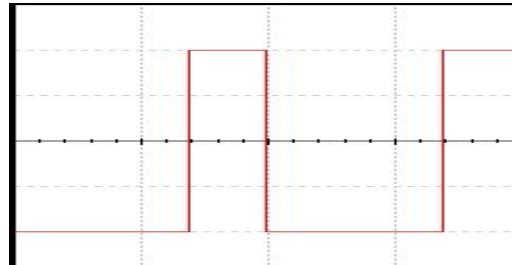
- ✓ The world we live in is analog.
- ✓ We are analog.
- ✓ Any inputs we can perceive are analog.
- ✓ For example,
  - sounds are analog signals; they are continuous time and continuous value.
  - Our ears listen to analog signals and we speak with analog signals.
  - Images, pictures, and video are all analog at the source and our eyes are analog sensors.
  - Measuring our heartbeat, tracking our activity, all requires processing analog sensor information.

# Examples of Analog Signal

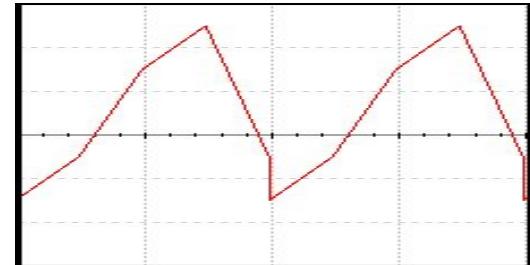
- ✓ An analog signal can be any time-varying signal.
- ✓ Minimum and maximum values can be either positive or negative.
- ✓ They can be periodic (repeating) or non-periodic.
- ✓ Sine waves and square waves are two common analog signals.
- ✓ Note that this square wave is not a digital signal because its minimum value is negative.
- ✓ Video and Audio



Sine wave

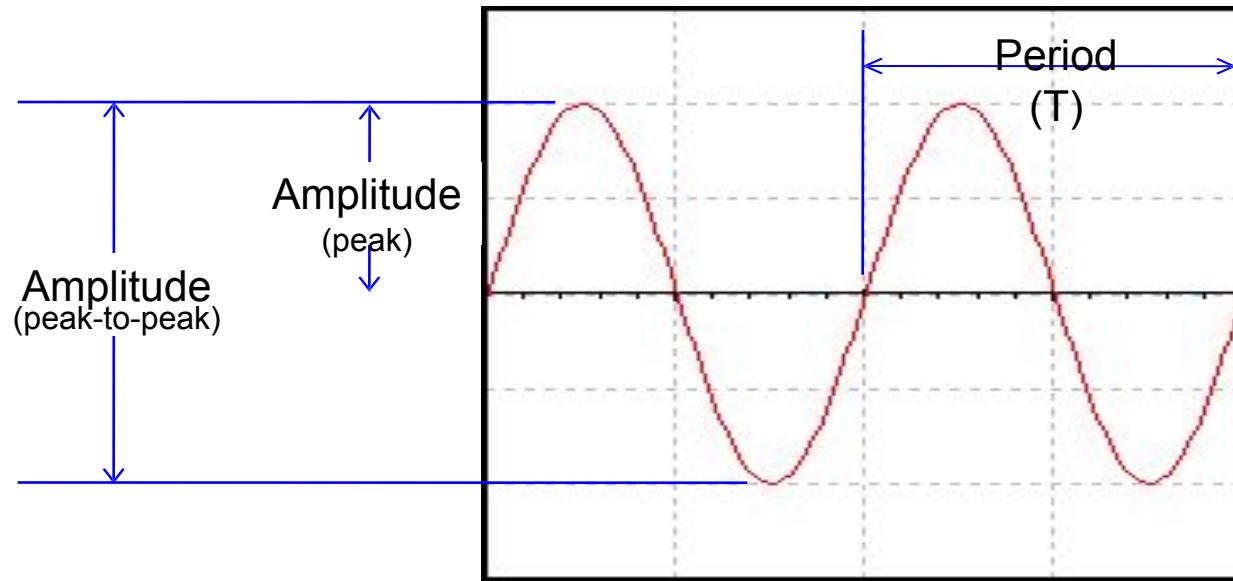


Square wave(not digital)



Random-Periodic

# Parts of Analog Signal



Frequency:  
 $F = \frac{1}{T}$   
Hz

# Pros and Cons Analog Signal

## □ Advantages

- ✓ Major advantages of the Analog signal is infinite amount of data.
- ✓ Density is much higher.
- ✓ Easy processing.

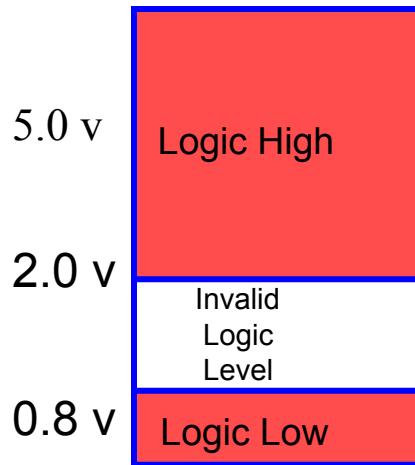
## □ Disadvantages

- ✓ Unwanted noise in recording.
- ✓ If we transmit data at long distance then unwanted disturbance is there.
- ✓ Generation loss is also a big con of analog signals.

# Logic Levels

Before examining digital signals, we must define logic levels. A logic level is a voltage level that represents a defined digital state.

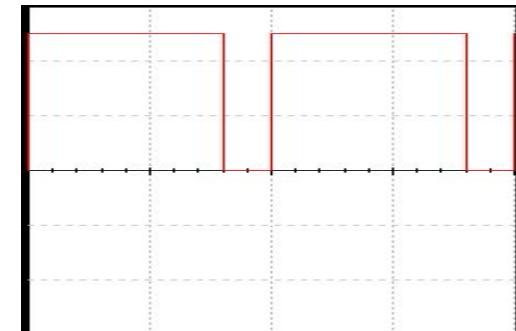
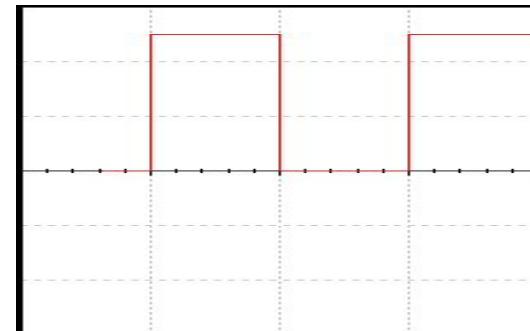
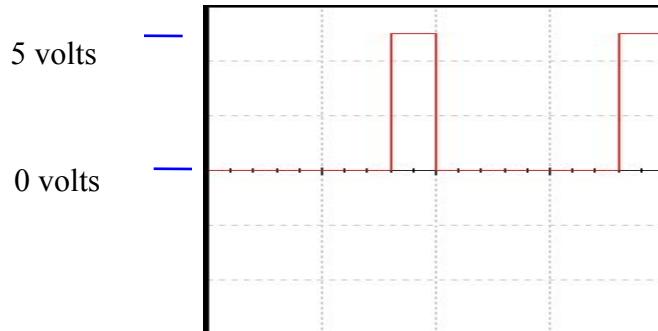
- ✓ Logic HIGH: The higher of two voltages, typically 5 volts
- ✓ Logic LOW: The lower of two voltages, typically 0 volts



Logic Level	Voltage	True/False	On/Off	0/1
HIGH	5 volts	True	On	1
LOW	0 volts	False	Off	0

# Examples of Digital Signal

- ✓ Digital signal are commonly referred to as square waves or clock signals.
- ✓ Their minimum value must be 0 volts, and their maximum value must be 5 volts.
- ✓ They can be periodic (repeating) or non-periodic.
- ✓ The time the signal is high ( $t_H$ ) can vary anywhere from 1% of the period to 99% of the period.
- ✓ Text and Integers.



# Parts of Digital Signal

## Amplitude:

For digital signals, this will ALWAYS be 5 volts.

## Period:

The time it takes for a periodic signal to repeat. (seconds)

## Frequency:

A measure of the number of occurrences of the signal per second.

## Time High ( $t_H$ ):

The time the signal is at 5 v.

## Time Low ( $t_L$ ):

The time the signal is at 0 v.

## Duty Cycle:

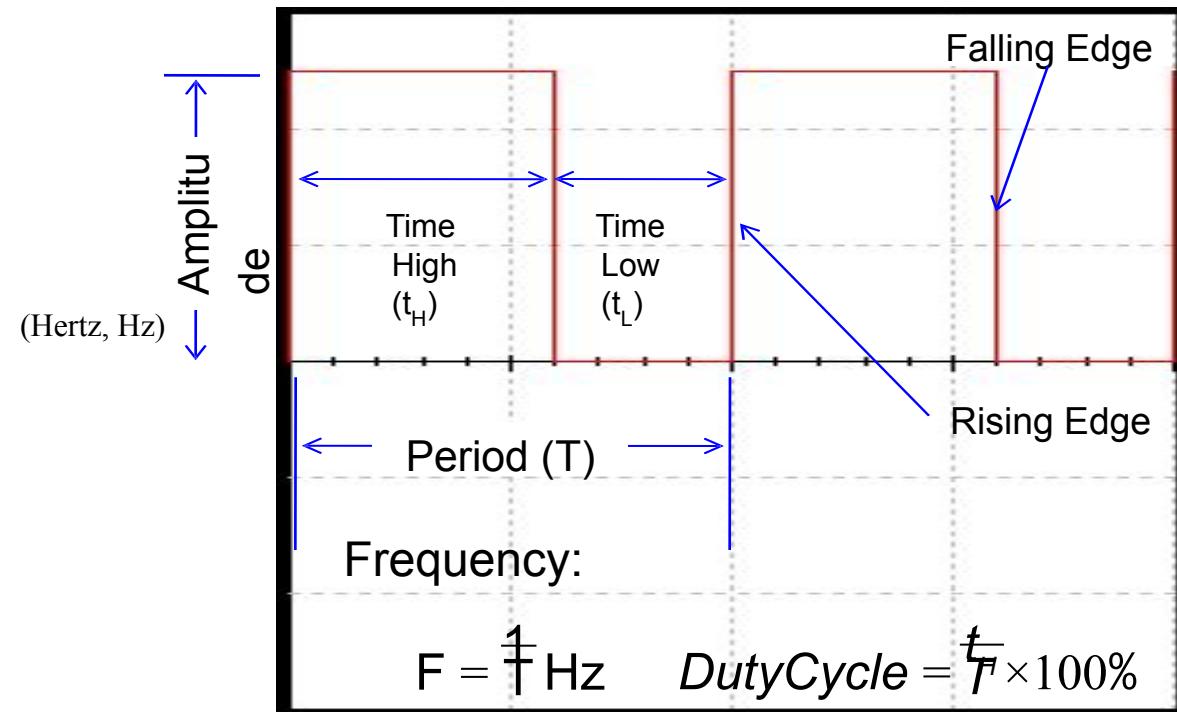
The ratio of  $t_H$  to the total period (T).

## Rising Edge:

A 0-to-1 transition of the signal.

## Falling Edge:

A 1-to-0 transition of the signal.



# Pros and Cons Digital Signal

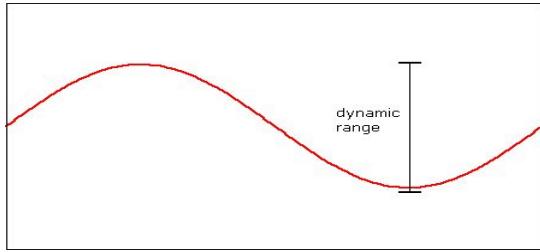
## □ Advantages

- ✓ Because of their digital nature they can travel faster in over digital lines.
- ✓ Ability to transfer more data as compared to analog.

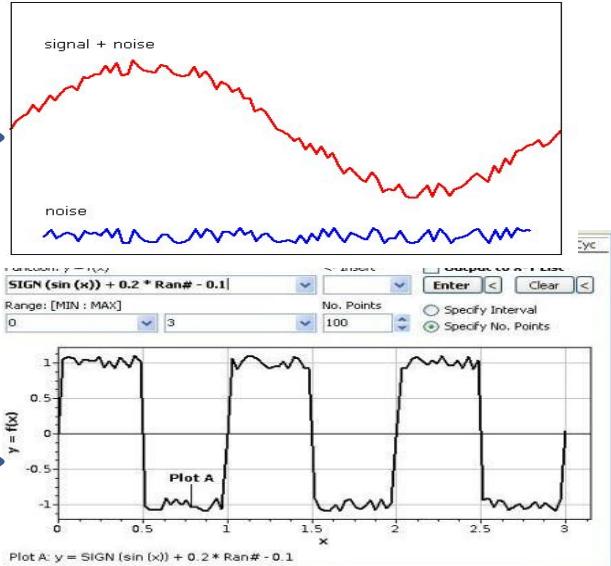
## □ Disadvantages

- ✓ Greater bandwidth is essential.
- ✓ Systems and processing is more complex.

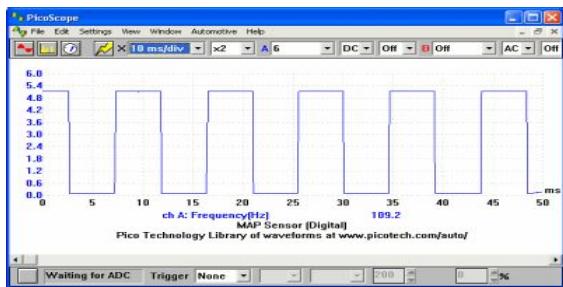
# Advantages of using digital over analog:



Noisy channel



Noisy channel



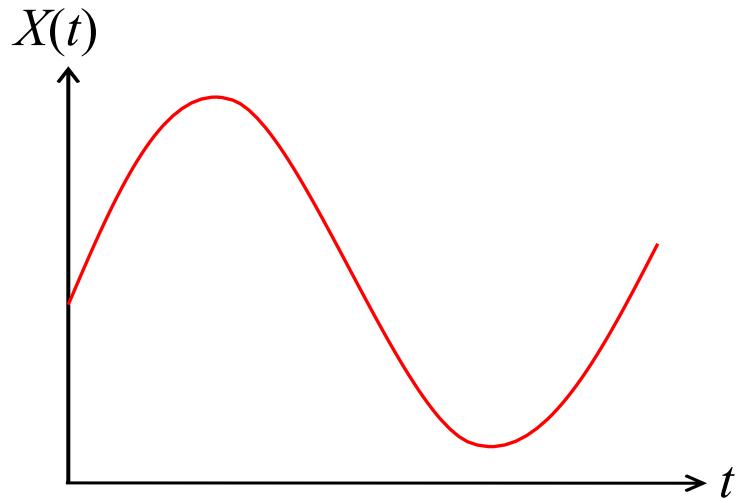
- ✓ Digital systems are less sensitive to noise
- ✓ As long as 0 is distinguishable from 1

# Analog Vs Digital

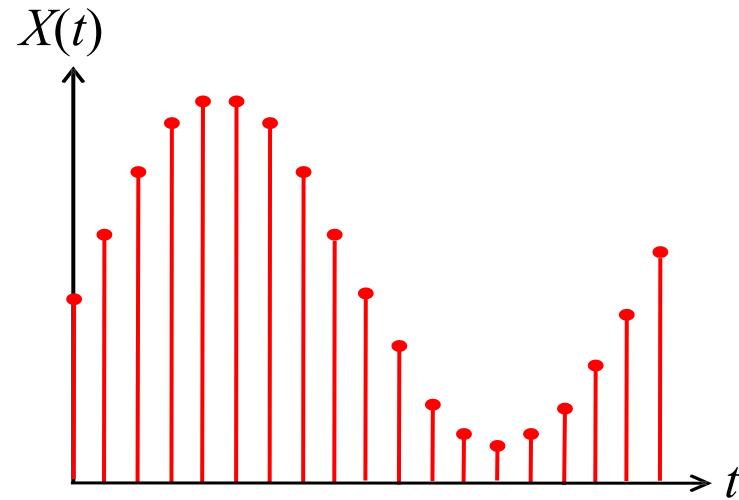
---

- Analog system
  - ✓ The physical quantities or signals may vary continuously over a specified range.
- Digital system
  - ✓ The physical quantities or signals can assume only discrete values.
  - ✓ Greater accuracy

# Analog Vs Digital



Analog signal



Digital signal

## Advantages of Digital System over Analog System

---

- ✓ Digital Systems are easier to design
- ✓ Information storage is easy
- ✓ Accuracy & Precision are greater
- ✓ Digital systems are more versatile
- ✓ Digital circuits are less affected by noise
- ✓ More digital circuitry can be fabricated on IC chips
- ✓ Reliability is more

# Thank You!

# **Unit-II**

# **Number System and Codes(6hrs)**

---

# Topics covered:

- Introduction to Different types of number system
- Binary to decimal and decimal to binary conversions, Octal, hexadecimal number system and conversions,
- Binary arithmetic, 1's complement and 9's complements,
- gray code, Excess 3 Code.
- instruction codes,
- alphanumeric characters,
- Modulo 2 system and 2's complement,
- Binary coded decimal (BCD) and hexadecimal codes,
- Parity method for error detection

# Number System

---

- ✓ A number system defines a set of values used to represent quantity.
- Positional number system.
- Non Positional number system

# Different Number Systems

---

✓ Decimal Number System

:Base/Radix 10

✓ Binary Number System

:Base/Radix 2

✓ Octal Number System

:Base/Radix 8

✓ Hexadecimal Number System

:Base/Radix 16

# Decimal Number System

---

- ✓ Decimal number system contains ten unique symbols  
0,1,2,3,4,5,6,7,8 and 9
- ✓ Since counting in decimal involves ten symbols, we can say that its base or radix is ten.
- ✓ It is a positional weighted system

# Decimal Number System

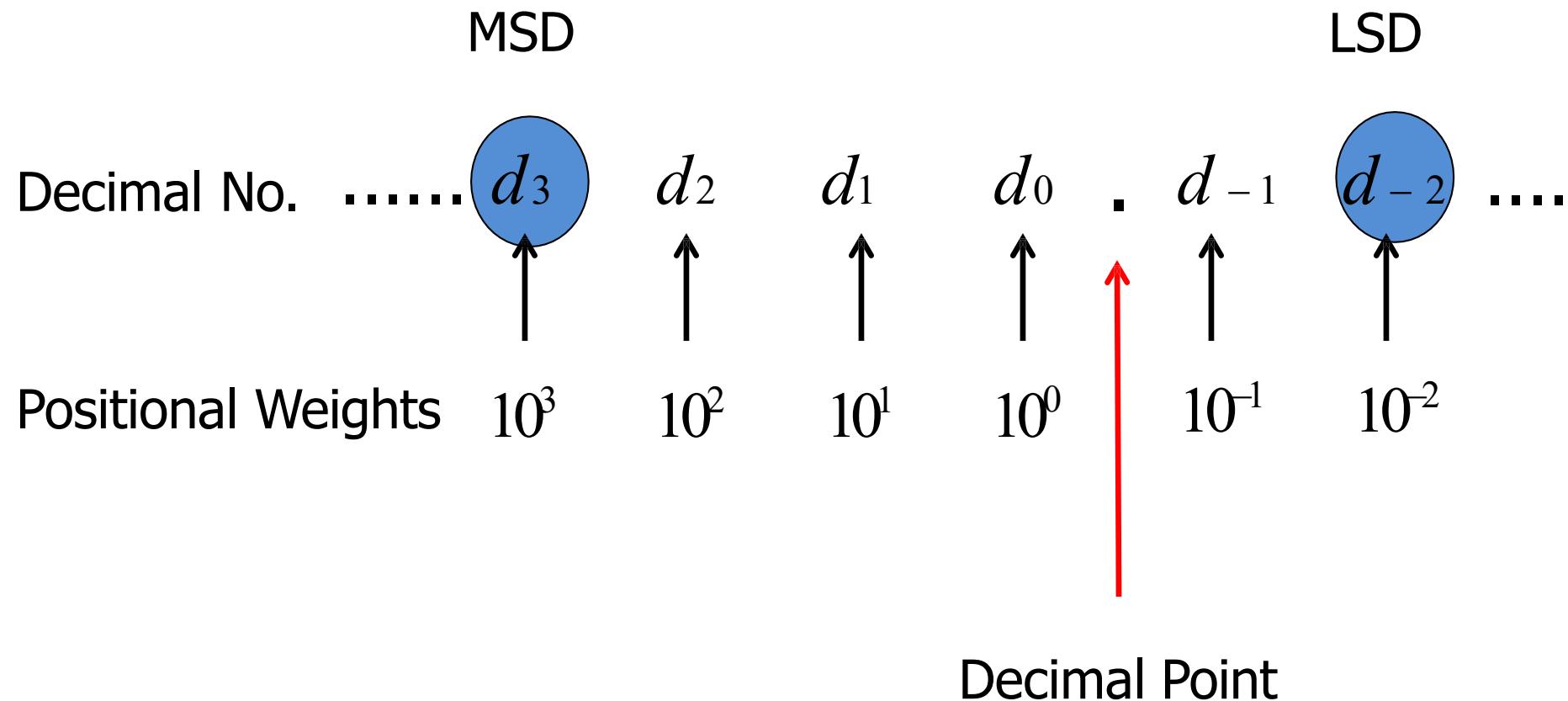
---

- ✓ In this system, any number (integer, fraction or mixed) of any magnitude can be represented by the use of these ten symbols only.
- ✓ Each symbols in the number is called a **“Digit”**

# Decimal Number System

---

## Structure:



# Decimal Number System

---

- ✓ **MSD:** The leftmost digit in any number representation, which has the greatest positional weight out of all the digits present in that number is called the “**Most Significant Digit**” (MSD)
- ✓ **LSD:** The rightmost digit in any number representation, which has the least positional weight out of all the digits present in that number is called the “**Least Significant Digit**” (LSD)

# Decimal Number System

---

## ➤ Examples

1214

1897

9875.54

# Binary Number System

---

- ✓ Binary number system is a positional weighted system
- ✓ It contains two unique symbols 0 and 1
- ✓ Since counting in binary involves two symbols, we can say that its base or radix is two.

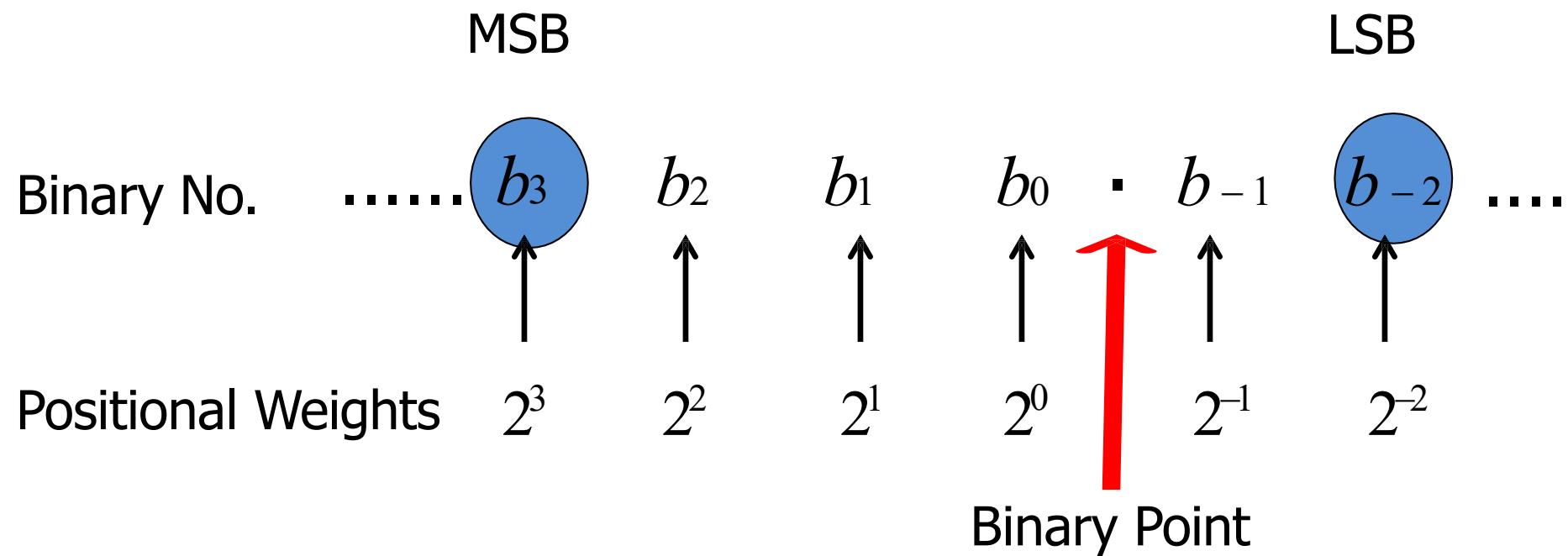
# Binary Number System

---

- ✓ A binary digit is called a “**Bit**”
- ✓ A binary number consists of a sequence of bits, each of which is either a 0 or a 1.
- ✓ The binary point separates the integer and fraction parts

# Binary Number System

## Structure:



# Binary Number System

---

✓ **MSB:** The leftmost bit in a given binary number with the highest positional weight is called the **“Most Significant Bit” (MSB)**

✓ **LSB:** The rightmost bit in a given binary number with the lowest positional weight is called the **“Least Significant Bit” (LSB)**

# Binary Number System

---

Decimal No.	Binary No.
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal No.	Binary No.
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

# Terms related to Binary Numbers

---

✓ **BIT:** The binary digits (0 and 1) are called bits.

- Single unit in binary digit is called “Bit”

- Example: 1  
              0

# Terms related to Binary Numbers

---

✓ **NIBBLE:** A nibble is a combination of 4 binary bits.

Example:      1110

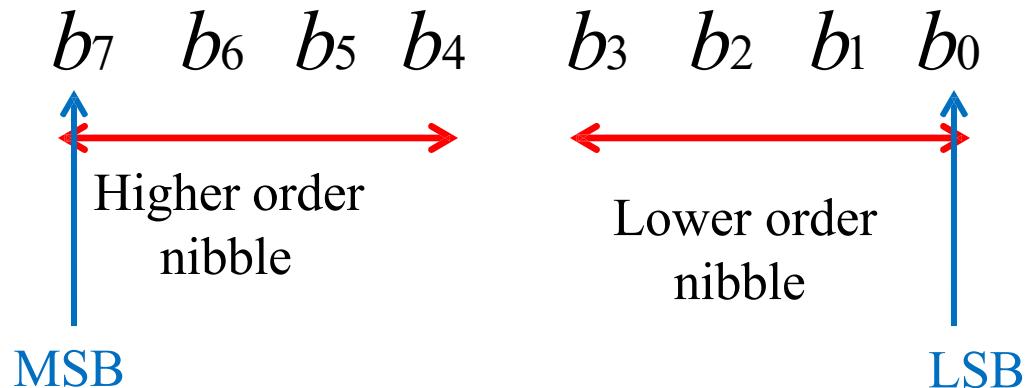
0000

1001

0101

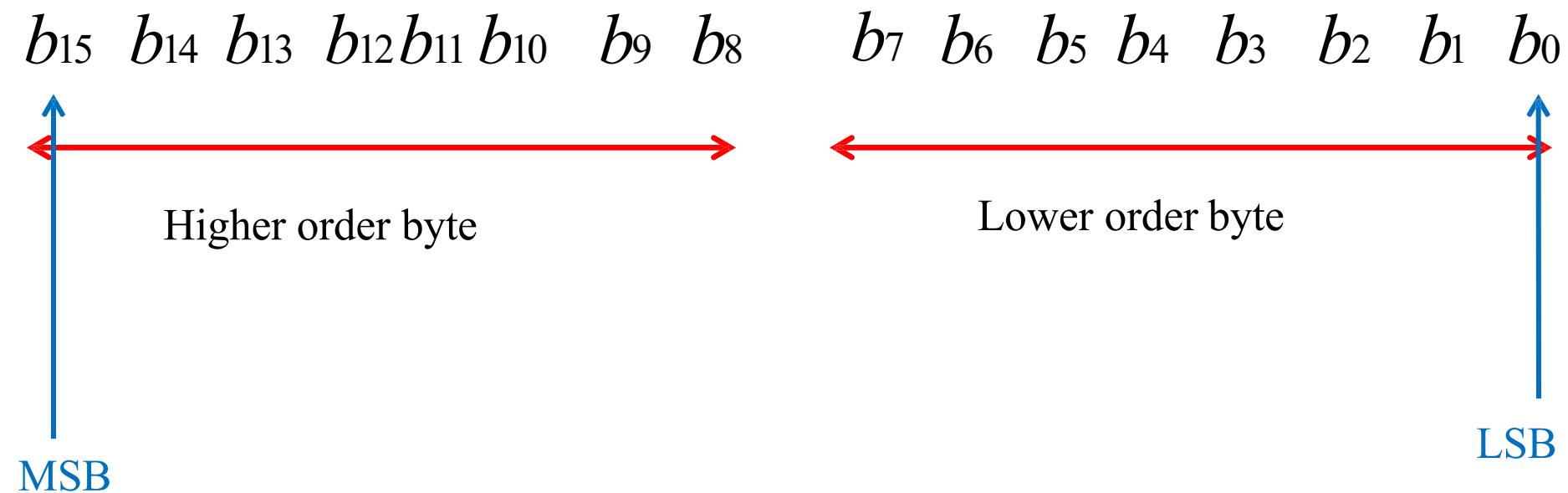
# Terms related to Binary Numbers

- ✓ **BYTE:** A byte is a combination of 8 binary bits.
- ✓ The number of distinct values represented by a byte is 256 ranging from 0000 0000 to 1111 1111.



# Terms related to Binary Numbers

✓ **WORD:** A word is a combination of 16 binary bits. Hence it consists of two bytes.



## Terms related to Binary Numbers

---

✓ **DOUBLE WORD:** A double word is exactly what its name implies, two words.

-It is a combination of 32 binary bits.

# Octal Number System

---

- ✓ Octal number system is a positional weighted system.
- ✓ It contains eight unique symbols 0,1,2,3,4,5,6 and 7
- ✓ Since counting in octal involves eight symbols, we can say that its base or radix is eight (8).

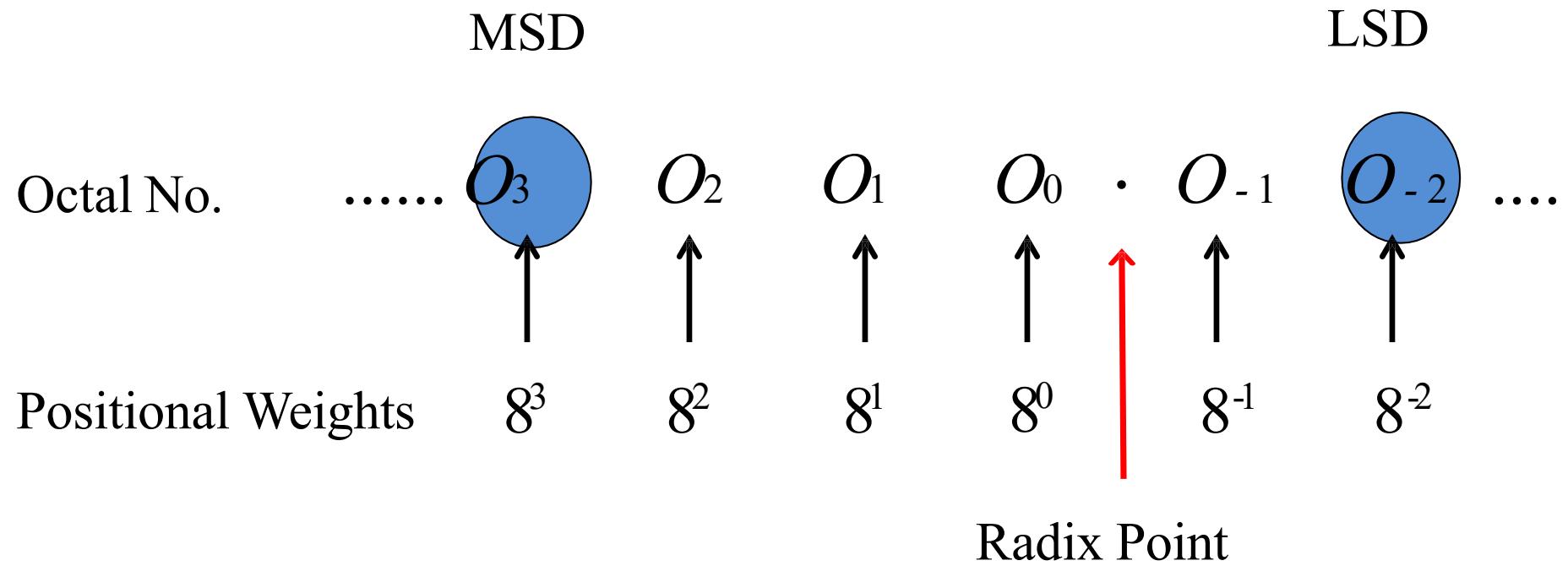
# Octal Number System

---

- ✓ The largest value of a digit in the octal system will be 7.
- ✓ That means the octal number higher than 7 will not be 8, instead of that it will be 10.

# Octal Number System

## **Structure:**



# Octal Number System

---

- ✓ Since its base  $8 = 2^3$ , every 3 bit group of binary can be represented by an octal digit.
- ✓ An octal number is thus  $1/3^{\text{rd}}$  the length of the corresponding binary number

# Octal Number System

---

Decimal No.	Binary No.	Octal No.
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

# Hexadecimal Number System (HEX)

---

- ✓ Binary numbers are long. These numbers are fine for machines but are too lengthy to be handled by human beings. So there is a need to represent the binary numbers concisely.
- ✓ One number system developed with this objective is the hexadecimal number system (or Hex)

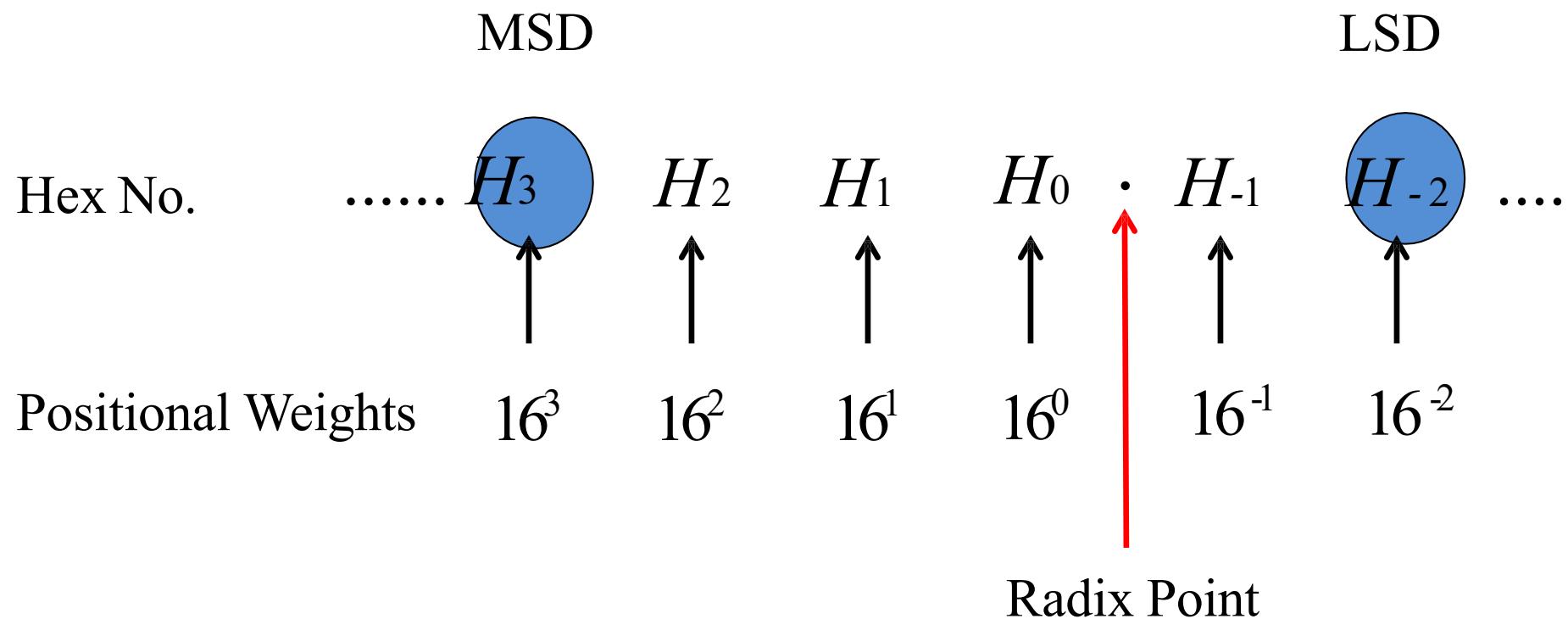
# Hexadecimal Number System (HEX)

---

- ✓ Hex number system is a positional weighted system
- ✓ It contains sixteen unique symbols  
0,1,2,3,4,5,6,7,8,9,A,B,C,D,E and F.
- ✓ Since counting in hex involves sixteen symbols, we can say that its base or radix is sixteen.

# Hexadecimal Number System (HEX)

## Structure:



# Hexadecimal Number System (HEX)

---

- ✓ Since its base  $16 = 2^4$ , every 4 bit group of binary can be represented by an hex digit.
- ✓ An hex number is thus  $1/4^{\text{th}}$  the length of the corresponding binary number
- ✓ The hex system is particularly useful for human communications with computer

# Hexadecimal Number System (HEX)

Decimal No.	Binary No.	Hex No.
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Decimal No.	Binary No.	Hex No.
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

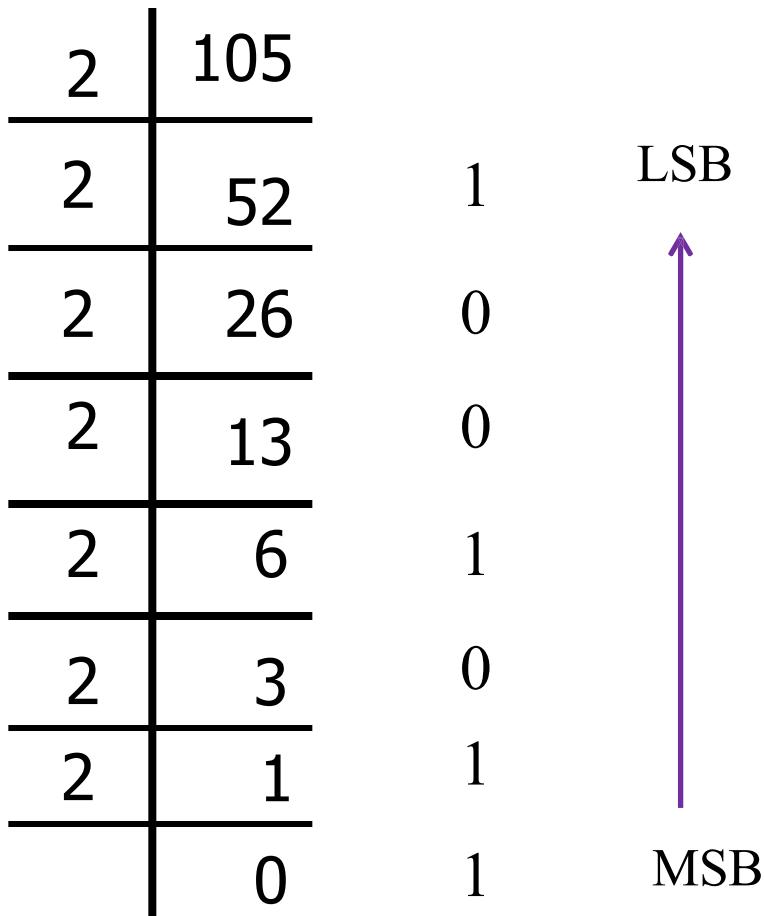
# **Conversion of Decimal number into Binary number (Integer Number)**

---

## **Procedure:**

1. Divide the decimal no by the base 2, noting the remainder.
2. Continue to divide the quotient by 2 until there is nothing left, keeping the track of the remainders from each step.
3. List the remainder values in reverse order to find the number's binary equivalent

# Example: Convert 105 decimal number in to it's equivalent binary number.



$$(105)_{10} = (1101001)_2$$

# Conversion of Decimal number into Binary number (Fractional Number)

---

## Procedure:

1. Multiply the given fractional number by base 2.
2. Record the carry generated in this multiplication as MSB.
3. Multiply only the fractional number of the product in step 2 by 2 and record the carry as the next bit to MSB.
4. Repeat the steps 2 and 3 up to 5 bits. The last carry will represent the LSB of equivalent binary number

# Example: Convert 0.42 decimal number in to it's equivalent binary number.

---

$$\begin{array}{rcl} 0.42 \times 2 & = & 0.84 \quad 0 \\ 0.84 \times 2 & = & 1.68 \quad 1 \\ 0.68 \times 2 & = & 1.36 \quad 1 \\ 0.36 \times 2 & = & 0.72 \quad 0 \\ 0.72 \times 2 & = & 1.44 \quad 1 \end{array}$$

↓

MSB

LSB

$$(0.42)_{10} = (0.01101)_2$$

## Exercise:

---

- Convert following Decimal Numbers in to its equivalent Binary Number:

$$(8476.47)_{10} = (?)_2$$

## Exercise:

---

- Convert following Decimal Numbers into its equivalent Binary Number:

$$(8476.47)_{10} = (?)_2$$

# Conversion of Decimal Number into Octal Number (Integer Number)

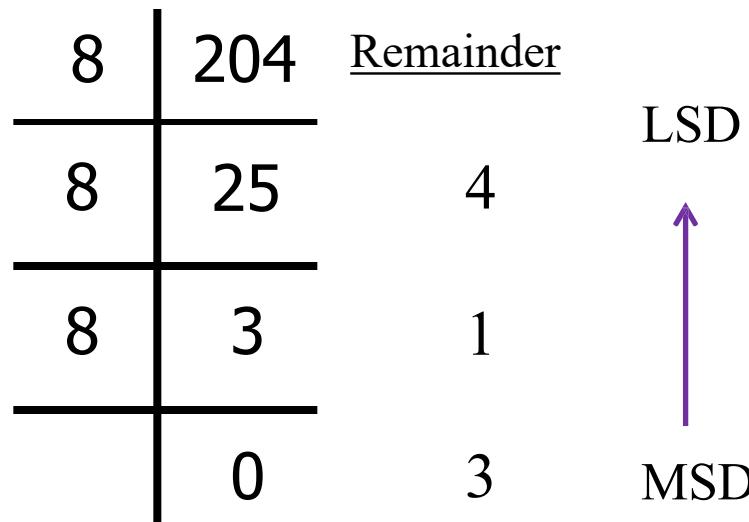
---

## Procedure:

1. Divide the decimal no by the base 8, noting the remainder.
2. Continue to divide the quotient by 8 until there is nothing left, keeping the track of the remainders from each step.
3. List the remainder values in reverse order to find the number's octal equivalent

# Example: Convert 204 decimal number in to it's equivalent octal number.

---



$$(204)_{10} = (314)_8$$

# Conversion of Decimal Number into Octal Number (Fractional Number)

## Procedure:

1. Multiply the given fractional number by base 8.
2. Record the carry generated in this multiplication as MSD.
3. Multiply only the fractional number of the product in step 2 by 8 and record the carry as the next bit to MSD.
4. Repeat the steps 2 and 3 up to 5 bits. The last carry will represent the LSD of equivalent octal number

# Example: Convert 0.6234 decimal number in to it's equivalent Octal number.

$$\begin{array}{rcl} 0.6234 \times 8 & = & 4.9872 \quad 4 \\ & & \text{MSD} \\ 0.9872 \times 8 & = & 7.8976 \quad 7 \\ & & \downarrow \\ 0.8976 \times 8 & = & 7.1808 \quad 7 \\ & & \downarrow \\ 0.1808 \times 8 & = & 1.4464 \quad 1 \\ & & \downarrow \\ 0.4464 \times 8 & = & 3.5712 \quad 3 \\ & & \text{LSD} \end{array}$$

$$(0.6234)_{10} = (0.47713)_8$$

## Exercise:

---

- Convert following Decimal Numbers in to its equivalent Octal Number:

$$(420.6)_{10} = (?)_8$$

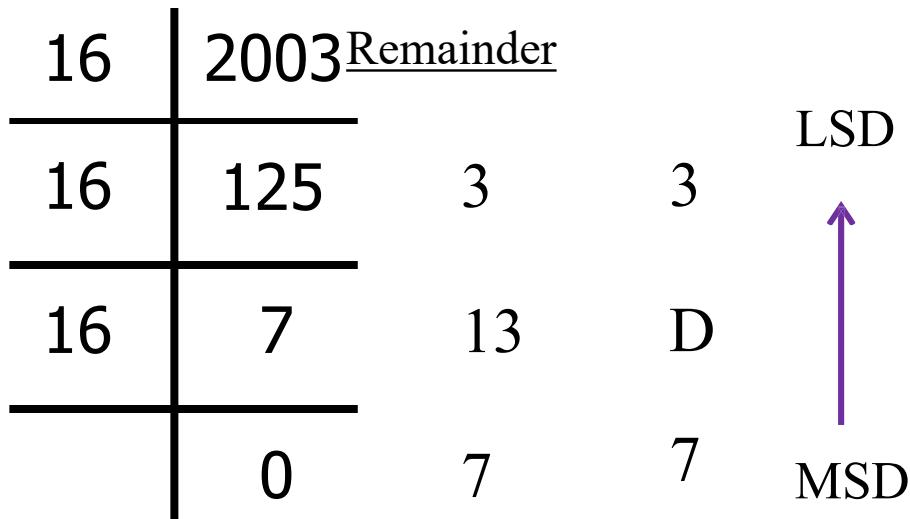
# **Conversion of Decimal Number into Hexadecimal Number (Integer Number)**

---

## **Procedure:**

1. Divide the decimal no by the base 16, noting the remainder.
2. Continue to divide the quotient by 16 until there is nothing left, keeping the track of the remainders from each step.
3. List the remainder values in reverse order to find the number's hex equivalent

# Example: Convert 2003 decimal number in to it's equivalent Hex number.



$$(2003)_{10} = (7D3)_{16}$$

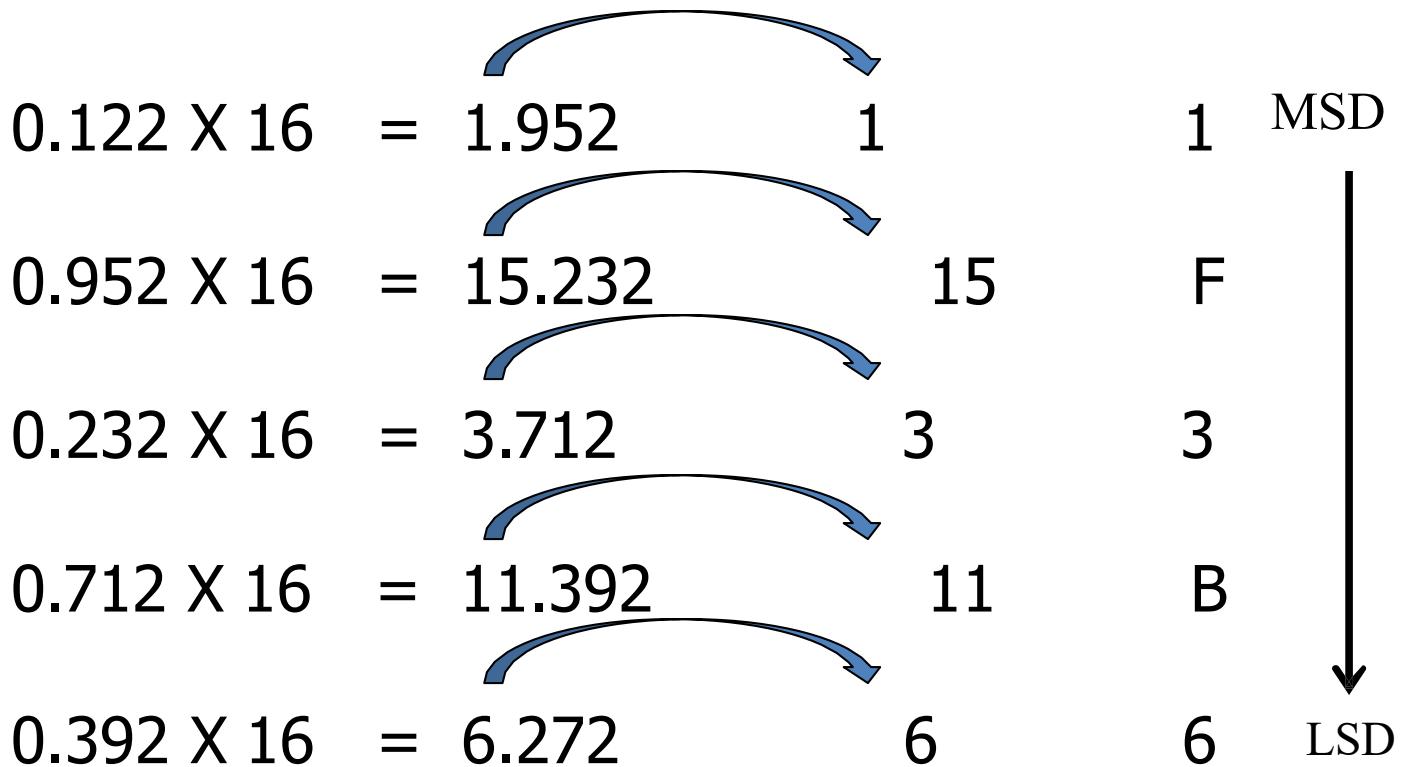
# Conversion of Decimal Number into Hexadecimal Number (Fractional Number)

---

## Procedure:

1. Multiply the given fractional number by base 16.
2. Record the carry generated in this multiplication as MSD.
3. Multiply only the fractional number of the product in step 2 by 16 and record the carry as the next bit to MSD.
4. Repeat the steps 2 and 3 up to 5 bits. The last carry will represent the LSD of equivalent hex number

# Example: Convert 0.122 decimal number in to it's equivalent Hex number.



$$(0.122)_{10} = 0.1F3B6)_{16}$$

## Exercise:

---

- Convert following Decimal Numbers in to its equivalent Hex Number:

$$(420.6)_{10} = (?)_{16}$$

# Conversion of Binary Number into Decimal Number

---

## Procedure:

1. Write down the binary number.
2. Write down the weights for different positions.
3. Multiply each bit in the binary number with the corresponding weight to obtain product numbers to get the decimal numbers.
4. Add all the product numbers to get the decimal equivalent

Example: Convert  $(1001.01)_2$  binary number in to it's equivalent decimal number.

---

Binary No.	1	0	0	1	.	1	1
Positional Weights	$2^3$	$2^2$	$2^1$	$2^0$	$2^{-1}$	$2^{-2}$	

$$= (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \cdot (1 \times 2^{-1}) + (1 \times 2^{-2})$$

$$= 8 + 0 + 0 + 1 \cdot 0.5 + 0.25$$

$$= 9.75$$

$$(1001.01)_2 = (9.75)_{10}$$

# Conversion of Octal Number into Decimal Number

---

## Procedure:

1. Write down the octal number.
2. Write down the weights for different positions.
3. Multiply each bit in the binary number with the corresponding weight to obtain product numbers to get the decimal numbers.
4. Add all the product numbers to get the decimal equivalent

## Example: Convert 365.24 octal number in to it's equivalent decimal number.

Octal No.

3	6	5	.	2	4
↑	↑	↑		↑	↑

Positional Weights

$8^2$	$8^1$	$8^0$	$8^{-1}$	$8^{-2}$
-------	-------	-------	----------	----------

$$= (3 \times 8^2) + (6 \times 8^1) + (5 \times 8^0) + (2 \times 8^{-1}) + (4 \times 8^{-2})$$

$$= 192 + 48 + 5 . 0.25 + 0.0625$$

$$= 245.3125$$

$$(365.24)_8 = (245.3125)_{10}$$

## Exercise

---

- Convert following Octal Numbers into its equivalent Decimal Number:

$$(273.56)_8 = (?)_{10}$$

## Example: Convert 273.56 octal number in to it's equivalent decimal number.

---

Octal No.

2	7	3	.	5	6
↑	↑	↑		↑	↑

Positional Weights

$8^2$	$8^1$	$8^0$	$8^{-1}$	$8^{-2}$
-------	-------	-------	----------	----------

$$= (2 \times 8^2) + (7 \times 8^1) + (3 \times 8^0) \cdot (5 \times 8^{-1}) + (6 \times 8^{-2})$$

$$\begin{aligned} &= 128 + 56 + 3 \cdot 0.625 \\ &\quad + 0.09375 \end{aligned}$$

$$= 187.71875$$

# Conversion of Hexadecimal Number into Decimal Number

---

## Procedure:

1. Write down the hex number.
2. Write down the weights for different positions.
3. Multiply each bit in the binary number with the corresponding weight to obtain product numbers to get the decimal numbers.
4. Add all the product numbers to get the decimal equivalent

## Example: Convert 5826 hex number in to it's equivalent decimal number.

---

Hex No.

5      8      2      6



Positional Weights

$16^3$        $16^2$        $16^1$        $16^0$

$$= (5 \times 16^3) + (8 \times 16^2) + (2 \times 16^1) + (6 \times 16^0)$$

$$= 20480 + 2048 + 32 + 6$$

$$= 22566$$

$$(5826)_{16} = (22566)_{10}$$

# Exercise

---

- Convert following Hexadecimal Numbers in to its equivalent Decimal Number:

$$(6B7)_{16} = (?)_{10}$$

# Conversion of Binary Number into Octal Number

---

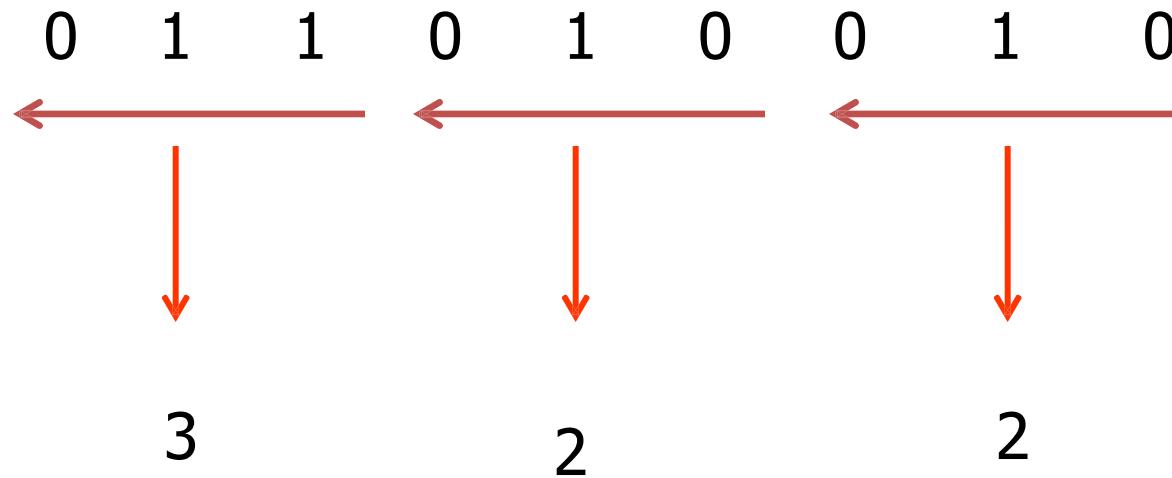
## Procedure:

1. Group the binary bits into groups of 3 starting from LSB.
2. Convert each group into its equivalent decimal.

As the number of bits in each group is restricted to 3, the decimal number will be same as octal number

Example: Convert 11010010 binary number in to it's equivalent octal number.

---



$$(11010010)_2 = (322)_8$$

## Exercise:

---

- Convert following Binary number into its equivalent Octal Number:

$$(1101.11)_2 = (?)_8$$

Ans:

# Conversion of Binary Number to Hexadecimal Number

---

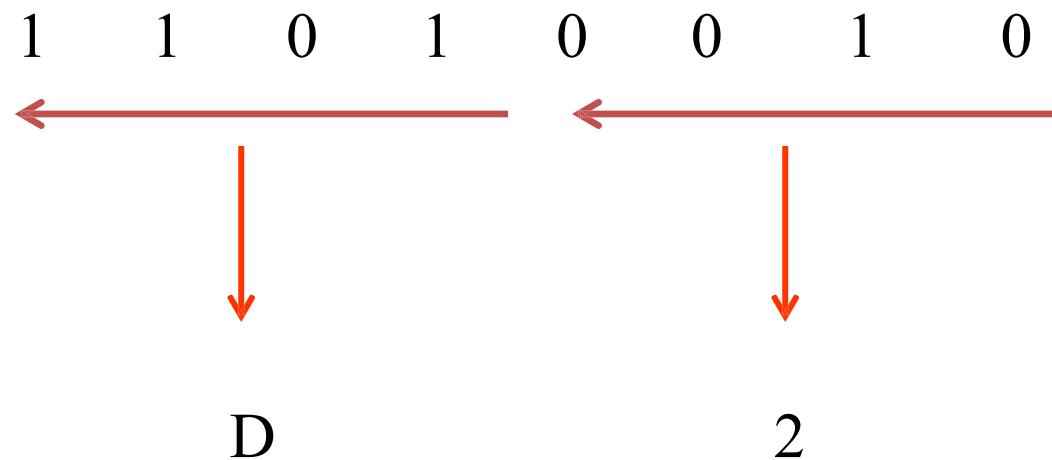
## Procedure:

1. Group the binary bits into groups of 4 starting from LSB.
2. Convert each group into its equivalent decimal.

As the number of bits in each group is restricted to 4, the decimal number will be same as hex number

# Example: Convert 11010010 binary number in to it's equivalent hex number.

---



$$(11010010)_2 = (\text{D}2)_{16}$$

## Exercise

---

- Convert following Binary Numbers into its equivalent Hexadecimal Number:

$$(10001.01)_2 = (?)_{16}$$

Ans:

# Conversion of Octal Number into Binary Number

---

- ✓ To get the binary equivalent of the given octal number we have to convert each octal digit into its equivalent 3 bit binary number

Example: Convert 364 octal number in to it's equivalent binary number.

---

3	6	4
↓	↓	↓
011	110	100

$$(364)_8 = (011110100)_2$$

OR

$$(364)_8 = (11110100)_2$$

## Exercise

---

- Convert following Octal Numbers in to its equivalent Binary Number:

$$(6534.04)_8 = (?)_2$$

# Conversion of Hexadecimal Number into Binary Number

---

✓ To get the binary equivalent of the given hex number we have to convert each hex digit into its equivalent 4 bit binary number

# Example: Convert AFB2 hex number in to it's equivalent binary number.

---

A	F	B	2
↓	↓	↓	↓
1010	1111	1011	0010

$$(AFB2)_{16} = (101011110110010)_2$$

## Exercise

---

- Convert following Hexadecimal Numbers in to its equivalent Binary Number:

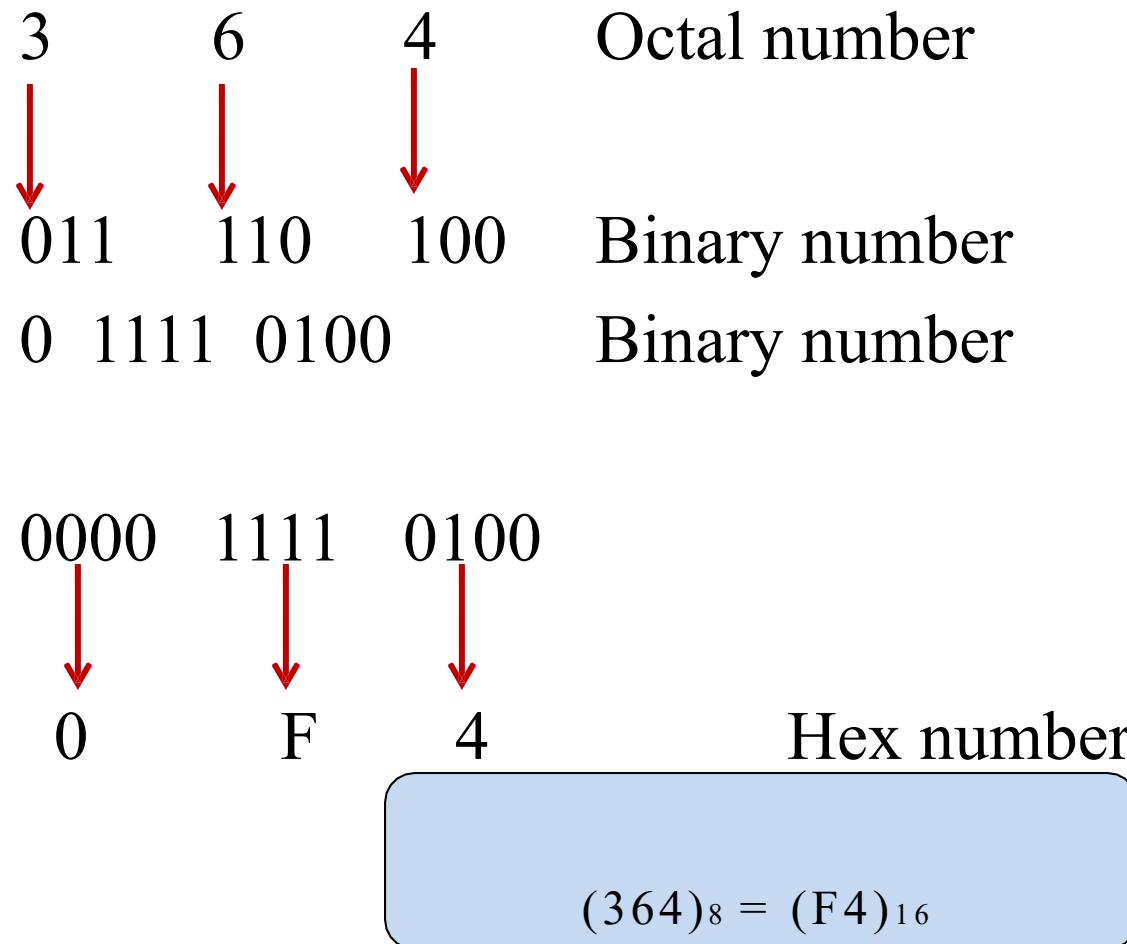
$$(8E47.AB)_{16} = (?)_2$$

## Conversion of Octal Number into Hexadecimal Number

---

✓ To get hex equivalent number of given octal number, first we have to convert octal number into its 3 bit binary equivalent and then convert binary number into its hex equivalent.

Example: Convert 364 octal number in to it's equivalent hex number.



# Exercise

---

- Convert following Octal Numbers in to its equivalent Hex Number:

$$(6534.04)_8 = (?)_{16}$$

# Conversion of Hexadecimal Number into Octal Number

---

✓ To get octal equivalent number of given hex number, first we have to convert hex number into its 4 bit binary equivalent and then convert binary number into its octal equivalent.

# Example: Convert 4CA hex number in to it's equivalent octal number.

Hex Number	Binary Number
4 ↓ 0100	C ↓ 1100

Binary Number	Octal Number
0100 1100 1010 — — — — ↓ ↓ ↓ ↓ 2 3 1 2	

$$(4CA)_{16} = (2312)_8$$

# Binary Addition

Example: Perform  $(1101.101)_2 + (111.011)_2$

$$\begin{array}{r} & 1 & 1 & 1 & 1 & & 1 & 1 \\ & 1 & 1 & 0 & 1 & . & 1 & 0 & 1 \\ + & 1 & 1 & 1 & 1 & . & 0 & 1 & 1 \\ \hline & 1 & 0 & 1 & 0 & 1 & . & 0 & 0 & 0 \end{array}$$

$$(1101.101)_2 + (111.011)_2 = (10101.000)_2$$

## Exercise

---

- Perform Binary Addition of following:

$$(1011)_2 + (1101)_2 + (1001)_2 + (1111)_2$$

# Binary Subtraction

---

- Following are the four most basic cases for binary subtraction

Subtraction				Borrow
0	-	0	=	0
0	-	1	=	1
1	-	0	=	1
1	-	1	=	0

# Binary Subtraction

Example: Perform  $(1010.010)_2 - (111.111)_2$

$$\begin{array}{r} & 1 & 10 & 1 & 1 & 10 \\ 0 & 10 & 0 & 10 & 10 & 0 & 10 \\ 1 & 0 & 1 & 0. & 0 & 1 & 0 \\ - & 1 & 1 & 1. & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 0. & 0 & 1 & 1 \end{array}$$

$$(1010.010)_2 - (111.111)_2 = (0010.011)_2$$

## Exercise

---

- Perform Binary Subtraction of following:

$$(10001.01)_2 - (1111.11)_2$$

# Binary Multiplication

---

- Following are the four most basic cases for binary multiplication

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

# Binary Multiplication

Example: Perform  $(1001)_2 * (1000)_2$

# Binary Multiplication

Example: Perform  $(1001)_2 * (1000)_2$

$$\begin{array}{r} & 1 & 0 & 0 & 1 \\ \times & 1 & 0 & 0 & 0 \\ \hline & 0 & 0 & 0 & 0 \\ & 0 & 0 & 0 & 0 & x \\ + & 1 & 0 & 0 & 1 & x & x \\ \hline & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}$$

$$(1001)_2 * (1000)_2 = (1001000)_2$$

## Exercise

---

- Perform Binary Multiplication of following:

$$(1101.11)_2 \times (101.1)_2$$

# Binary Division

Example: Perform  $(110110)_2 / (101)_2$

$$\begin{array}{r} 1 \ 0 \ 1 \ 0 \\ 101 \overline{)1 \ 1 \ 0 \ 1 \ 1 \ 0} \\ -1 \ 0 \ 1 \\ \hline * \ 0 \ 1 \ 1 \\ -0 \ 0 \ 0 \\ \hline * \ 1 \ 1 \ 1 \\ -1 \ 0 \ 1 \\ \hline * \ 1 \ 0 \ 0 \\ -0 \ 0 \ 0 \\ \hline 1 \ 0 \ 0 \end{array}$$

The diagram shows the binary division process. The divisor is 101, and the dividend is 110110. The quotient is 1010, and the remainder is 100. Red arrows point to the subtraction steps: from 110 to 101, from 011 to 000, and from 111 to 101. The final subtraction step from 100 to 000 is also indicated by a red arrow.

Remainder: 100  
Quotient: 1010

## Exercise

---

- Perform Binary Division of following:

$(1010)_2$  by  $(11)_2$

## 1's Complement

---

- The 1's complement of a number is obtained by simply complementing each bit of the number that is by changing all 0's to 1's and all 1's to 0's.
- This system is called as 1's complement because the number can be subtracted from 1 to obtain result

# 1's Complement

Example: Obtain 1's complement of the 1010

$$\begin{array}{r} 1 & 1 & 1 & 1 \\ - & & & \\ \hline 0 & 1 & 0 & 1 \end{array}$$

1's complement of the 1010 is 0101

# 1's Complement

Sr. No.	Binary Number	1's Complement
1	1101 0101	0010 1010
2	1001	0110
3	1011 1111	0100 0000
4	1101 1010 0001	0010 0101 1110
5	1110 0111 0101	0001 1000 1010
6	1011 0100 1001	0100 1011 0110
7	1100 0011 0010	0011 1100 1101
8	0001 0010 1000	1110 1101 0111

# Subtraction Using 1's Complement

- In 1's complement subtraction, add the 1's complement of subtrahend to the minuend.
- If there is carry out, bring the carry around and add it to LSB.
- If carry is present, the answer is positive and it is true binary form
- If there is no carry, the answer is negative and it is in 1's complement.

# Subtraction using 1's Complement

Example: Perform using 1' complement

$$(9)_{10} - (4)_{10}$$

Step 1: Take 1' complement of

$$\begin{aligned}(4)_{10} &= (0100)_2 \\ &= 1011\end{aligned}$$

Step 2: Add 9 with 1' complement of 4

$$\begin{array}{r} & 1 & 0 & 0 & 1 \\ + & 1 & 0 & 1 & 1 \\ \hline & 0 & 1 & 0 & 0 \end{array}$$

final carry → 1      Result

Step 3: If carry is generated add final carry to the result

# Subtraction using 1's Complement

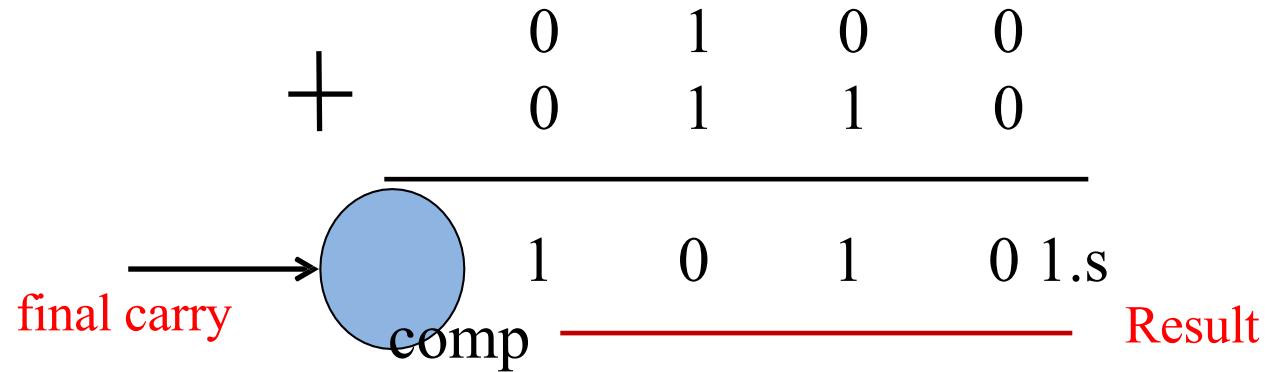
Example: Perform using 1' complement

$$(4)_{10} - (9)_{10}$$

Step 1: Take 1' complement of

$$\begin{aligned}(9)_{10} &= (1001)_2 \\ &= 0110\end{aligned}$$

Step 2: Add 4 with 1' complement of 9



Step 3: If carry is generated add final carry to the result

$$0 \quad 1 \quad 0 \quad 1$$

# Example Continue

$$\begin{array}{r} & 1 & 0 & 0 & 1 \\ + & 1 & 0 & 1 & 1 \\ \hline \end{array}$$

final carry  $\longrightarrow$  1 0 1 0 0 Result

↓

1 →

$$\begin{array}{r} 0 & 1 & 0 & 1 \\ \hline \end{array}$$

Final Result

When the final carry is produced the answer is positive and is in its true binary form

# 2's Complement

---

- ✓ The 2's complement of a number is obtained by adding 1 to the 1's complement of that number

# 2's Complement

Sr. No.	Binary Number	1's Complement	2's Complement
1	1101 0101	0010 1010	0011 1011
2	1001	0110	0111
3	1011 1111	0100 0000	0100 0001
4	1101 1010 0001	0010 0101 1110	0010 0101 1111
5	1110 0111 0101	0001 1000 1010	0001 1000 1011

# Subtraction Using 2's Complement

---

- ✓ In 2's complement subtraction, add the 2's complement of subtrahend to the minuend.
- ✓ If carry is generated then the result is positive and in its true binary form.
- ✓ If the carry is not produced, then the result is negative and in its 2's complement form.

\*Carry is always to be discarded

# Subtraction Using 2's Complement

Example: Perform using 2' complement  $(9)_{10} - (4)_{10}$

Step 1: Take 2' complement of  $(4)_{10} = (0100)_2$

$$= 1011 + 1 = 1100$$

Step 2: Add 9 with 2' complement of 4

$$\begin{array}{r} & 1 & 0 & 0 & 1 \\ + & 1 & 1 & 0 & 0 \\ \hline & 0 & 1 & 0 & 1 \end{array}$$

final carry → Discard      Final Result

If Carry is generated, discard carry. The result is positive and its true binary form.

## BCD or 8421 Code

---

- ✓ The smallest BCD number is (0000) and the largest is (1001). The next number to 9 will be 10 which is expressed as (0001 0000) in BCD.
- ✓ There are six illegal combinations 1010, 1011, 1100, 1101, 1110 and 1111 in this code i.e. they are not part of the 8421 BCD code

# Decimal to BCD Conversion

---

Sr. No.	Decimal Number	BCD Code
1	8	1000
2	47	0100 0111
3	345	0011 0100 0101
4	99	1001 1001
5	10	0001 0000

# Gray Code

---

- ✓ The gray code is non-weighted code.
- ✓ It is not suitable for arithmetic operations.
- ✓ It is a cyclic code because successive code words in this code differ in one bit position only i.e. unit distance code

<u>Binary</u>	<u>Gray code</u>
000	001
001	001
010	011
011	010

# Binary to Gray Code Conversion

---

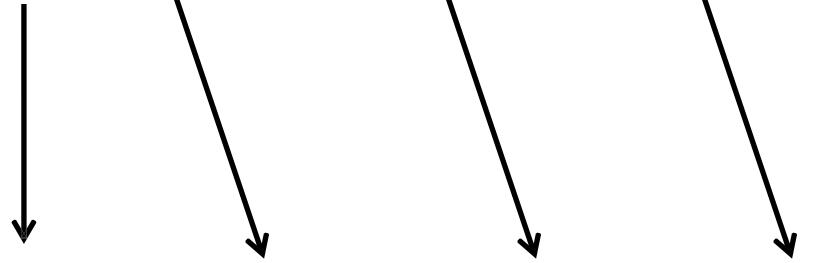
- Keep MSB of gray code same as MSB of binary code.
- Going from left to right, add adjacent pairs of binary code to get next gray code bit.
- Discard any carries

## Example 1: Convert 1011 Binary Number into Gray Code

---

Binary Number

1 →⊕← 0 →⊕← 1 →⊕← 1



Gray Code

1 1 1 0

# Binary and Corresponding Gray Codes

Decimal No.	Binary No.	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

# Gray Code to Binary Conversion

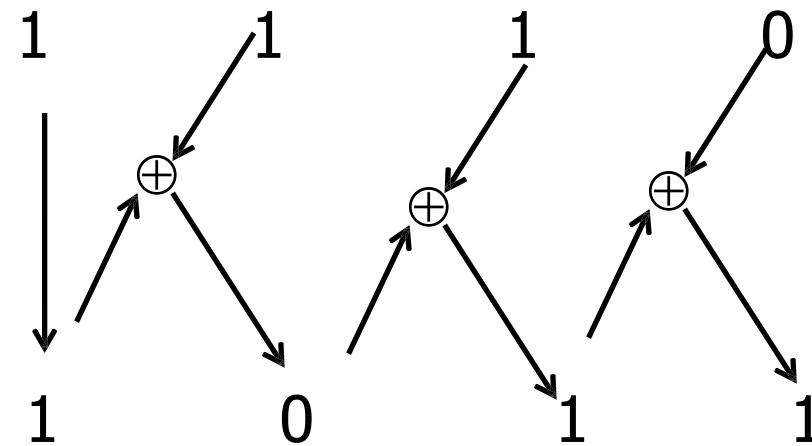
---

- Keep MSB of binary code same as MSB of gray code.
- Add each binary code generated to the gray code bit in the next adjacent.
- Discard carries.

## Example 1:: Convert 1110 Gray code into Binary Number.

---

Gray Code



Binary Number

# Excess-3 Code (XS-3)

Example 1: Obtain Xs-3 Code for 428 Decimal

$$\begin{array}{ccc} & 4 & \\ & 2 & \\ & 8 & \\ \\ \begin{array}{r} 0100 \\ + 0011 \\ \hline \end{array} & \begin{array}{r} 0010 \\ 0011 \\ \hline 0101 \end{array} & \begin{array}{r} 1000 \\ 0011 \\ \hline 1011 \end{array} \end{array}$$

# Excess-3 Code (XS-3)

Decimal No.	BCD Code	Excess-3 Code= BCD + Excess-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

# Excess 3 code is a self complementary code

Decimal Digit	BCD code	Excess-3 code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100



# Alphanumeric code

- Alphanumeric codes are also called character codes, are binary codes used to represent alphanumeric data. The codes write alphanumeric data, including letters of the alphabet, numbers, mathematical symbols and punctuation marks, in a form that is understandable and process able by a computer.
- Using these code we can interface, input output devices such as keyboards, monitors, printers etc. With computer.

# ASCII Codes

---

- ✓ The **American Standard Code for Information Interchange** is a character-encoding scheme originally based on the English alphabet.
- ✓ ASCII codes represent text in computers, communications equipment, and other devices that use text.
- ✓ Most modern character-encoding schemes are based on ASCII, though they support many additional characters.

# ASCII Codes

---

- ✓ ASCII includes definitions for 128 characters: 33 are non-printing control characters (many now obsolete) that affect how text and space is processed and 95 printable characters, including the space (which is considered an invisible graphic)
- ✓ Example: a=97

A=65

# BCD Addition

---

- ✓ The BCD addition is performed by individually adding the corresponding digits of the decimal number expressed in 4 bit binary groups starting from LSD.
- ✓ If there is no carry & the sum term is not an illegal code, no correction is needed.

# BCD Addition

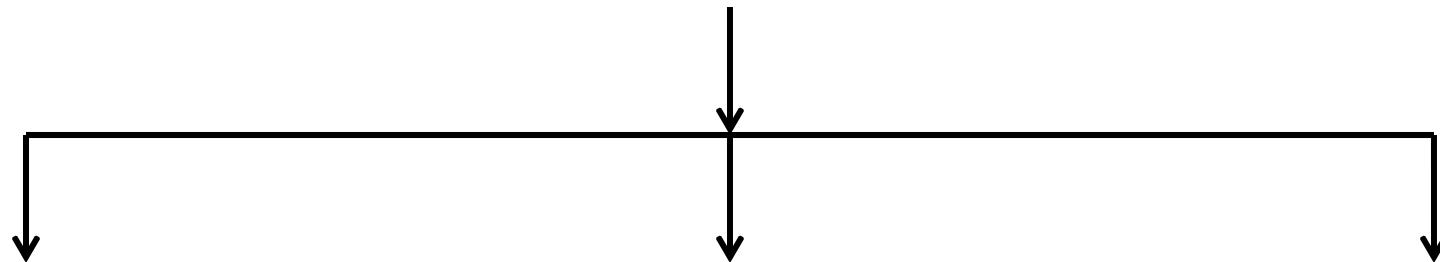
---

- ✓ If there is a carry out of one group to the next group or if the sum term is an illegal code then 6 i.e. 0110 is added to the sum term of that group and resulting carry is added to the next group.
- ✓ This is done to skip the six illegal states.

# BCD Addition

---

Addition of two BCD numbers



Sum $\leq$ 9, Carry=0

Sum $\leq$ 9, Carry=1

Sum $>$ 9, Carry=0



Answer is correct.  
No correction  
required.

Add 6 to the sum  
term to get the  
correct answer

Add 6 to the sum  
term to get the  
correct answer

# BCD Addition

---

Example: Perform in BCD  $(57)_{10} + (26)_{10}$

# BCD Addition

---

Example: Perform in BCD  $(57)_{10} + (26)_{10}$

$$\begin{array}{r} 57 \\ + 26 \\ \hline 83 \end{array} \quad \begin{array}{r} 0 1 0 1 0 1 1 1 \\ + 0 0 1 0 0 1 1 0 \\ \hline 0 0 1 1 1 1 1 0 \end{array}$$

Final Carry 0      Valid BCD Code      Invalid BCD Code

Thus we have to add 0110 in illegal BCD code

# Example

$$\begin{array}{r} 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ + & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Add 0110 in  
only invalid  
code

$$(57)_{10} + (26)_{10} = (83)_{10}$$

# Exercise

---

- Perform BCD Addition

$$(88.7)_{10} + (265.8)_{10}$$

# Parity method for error detection

- Parity bit: an extra bit included with message to make total number of 1's either odd or even.

message	P(odd)	P(even)
000	1	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	0	1

# Subtraction of decimal using 9's complement

- 9's complement of number is obtained by subtracting each decimal digit from 9.
- Example: 9's complement of 523 is: 476.

# Subtraction of decimal using 10's complement

- 10”s complement of number is obtained by adding 1 to the 9’s complement
- Example: 10’s complement of 523 is:  $476+1=477$ .

# Thank you!

# **UNIT-3: BOOLEAN ALGEBRA AND LOGIC GATES(4 Hrs)**

# Topics covered:

- Basic definition, Basic properties and theorem of Boolean algebra, DeMorgan's theorem,
- Logic gates and truth tables, Universality of NAND and NOR gates, Trio-stage logic.

- ✓ Logic gates are the fundamental building blocks of digital systems.
- ✓ The name logic gate is derived from the ability of such devices to make decisions, in the sense that it produces one output level when some combinations of input levels are present

## ➤ Inputs & Outputs for Logic Circuits

- ✓ Input & Output of logic gates can occur only in two levels.

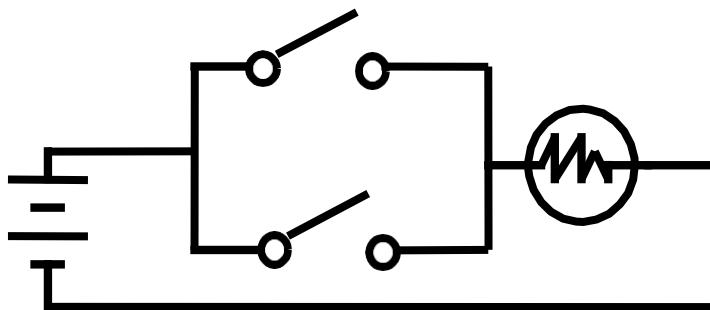
HIGH	LOW
True	False
ON	OFF
1	0

# Basics.....

## ➤ Truth Table

- ✓ A table which lists all the possible combinations of input variables and the corresponding outputs is called a “Truth Table”.
- ✓ It shows how the logic circuits output responds to various combinations of logic levels at the inputs

Switches in parallel => OR



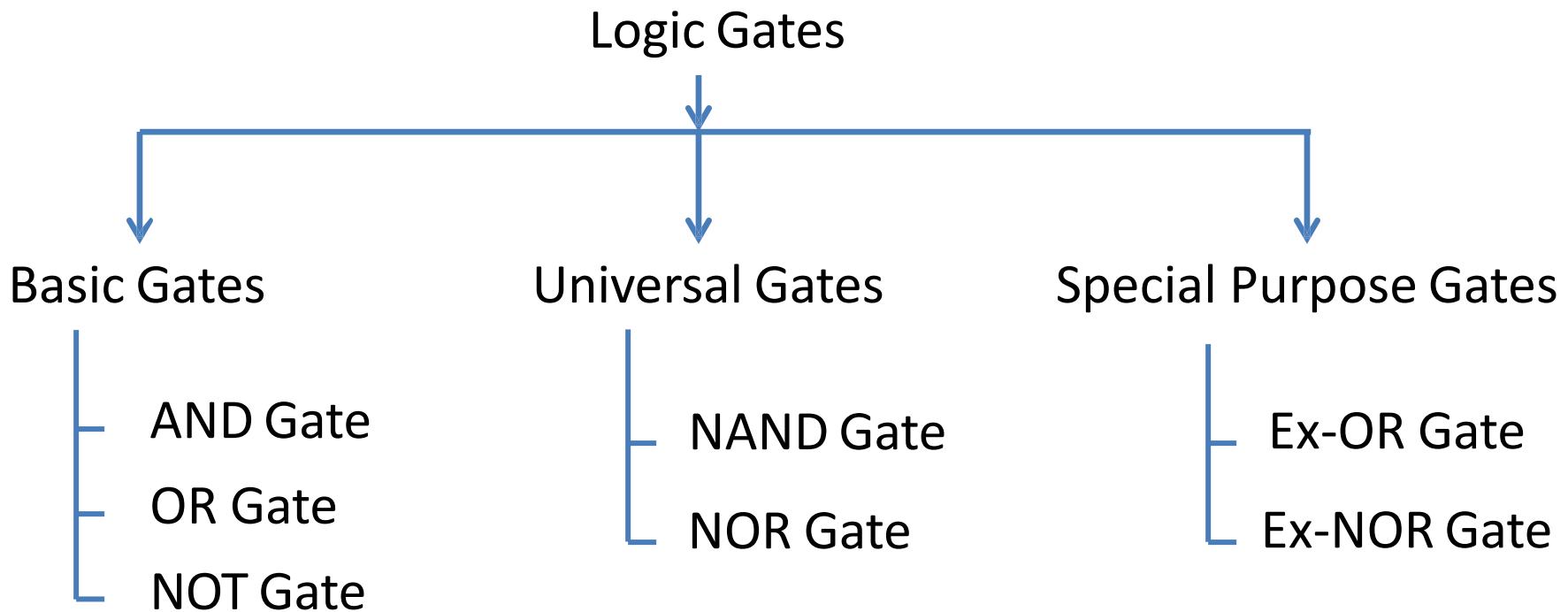
Switch 1	Switch 2	Output
OFF	OFF	OFF
OFF	ON	GLOW
ON	OFF	GLOW
ON	ON	GLOW

## ➤ Logic

- ✓ A logic in which the voltage levels represent logic 1 and logic 0.
- ✓ Level logic may be Positive or Negative.
- ✓ A “**Positive Logic**” is the one which the higher of the two voltage levels represents the logic 1 and the lower of the two voltage level represents the logic 0.
- ✓ A “**Negative Logic**” is the one which the lower of the two voltage levels represents the logic 1 and the higher of the two voltage level represents the logic 0.

# Logic Gates

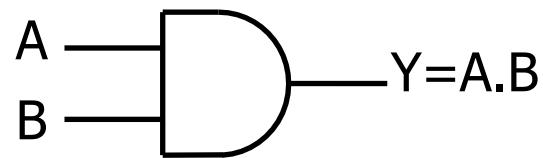
---



## AND Gate

---

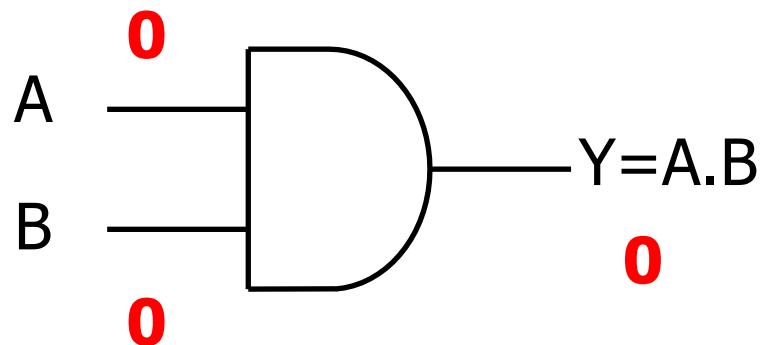
- ✓ An AND gate has two or more inputs but only one output.
- ✓ The output assumes the logic 1 state, when both inputs are at logic 1 state.
- ✓ The output assumes the logic 0 state even if one of its inputs is at logic 0 state.



Logic Symbol

# AND Gate

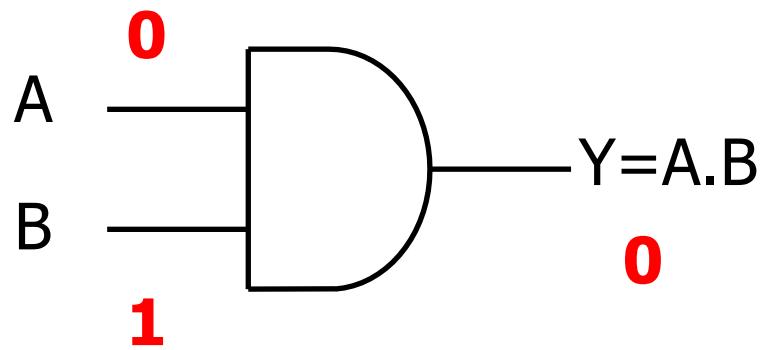
---



Inputs		Output
A	B	$Y=A \cdot B$
0	0	0

# AND Gate

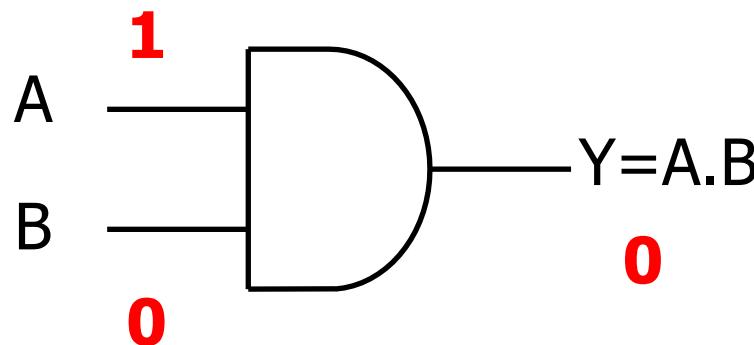
---



Inputs		Output
A	B	$Y=A \cdot B$
0	0	0
0	1	0

# AND Gate

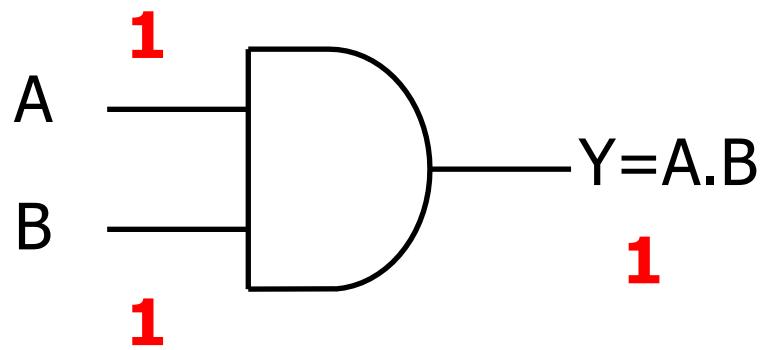
---



Inputs		Output
A	B	$Y=A \cdot B$
0	0	0
0	1	0
1	0	0

# AND Gate

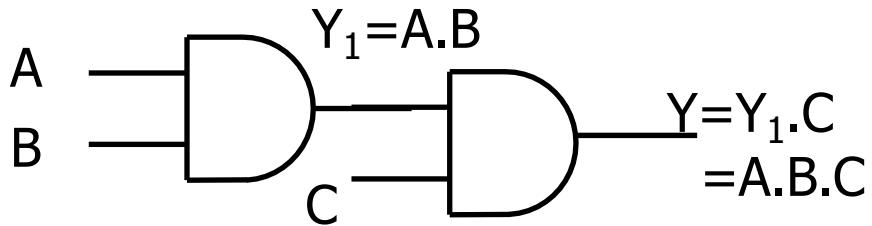
---



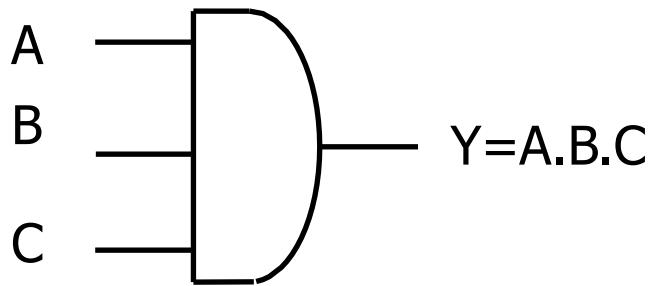
Inputs		Output
A	B	$Y=A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

# 3 - Input AND Gate

3 – Input AND Gate using 2 – Input AND Gate



Symbol : 3 – Input AND Gate



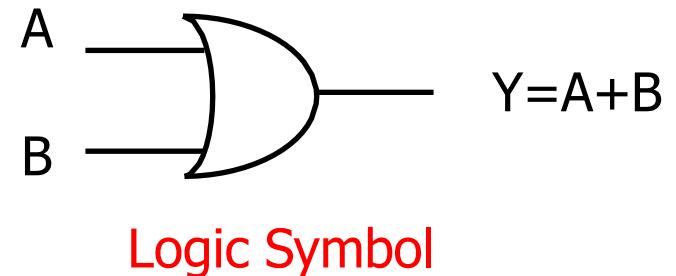
Truth Table : 3 – Input AND Gate

Input			Output $Y=A \cdot B \cdot C$
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

# OR Gate

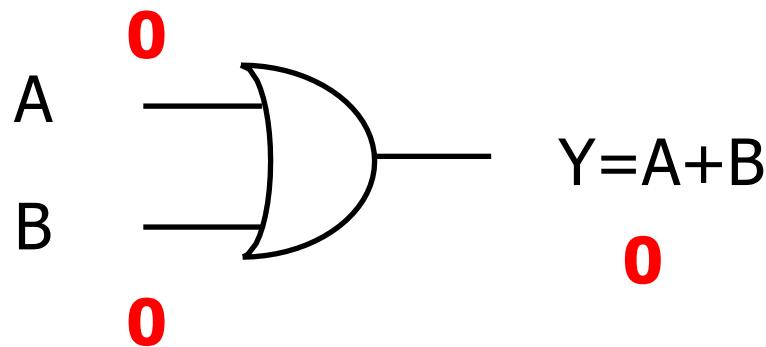
---

- ✓ An OR gate has two or more inputs but only one output.
- ✓ The output assumes the logic 1 state, when even if one of its inputs is in logic 1 state.
- ✓ The output assumes the logic 0 state only when both the inputs are in logic 0 state.



# OR Gate

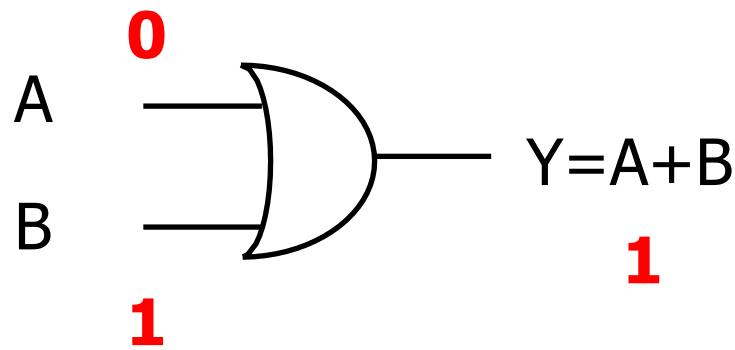
---



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

# OR Gate

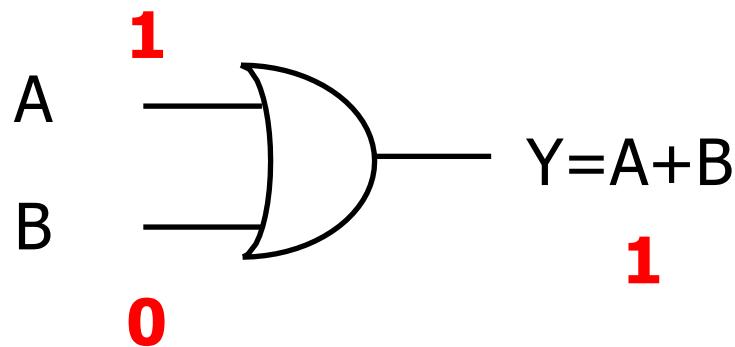
---



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

# OR Gate

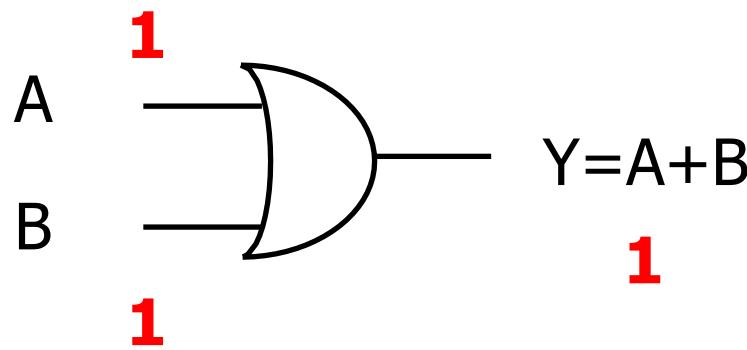
---



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

# OR Gate

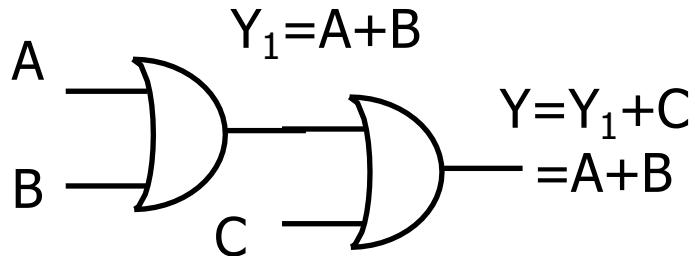
---



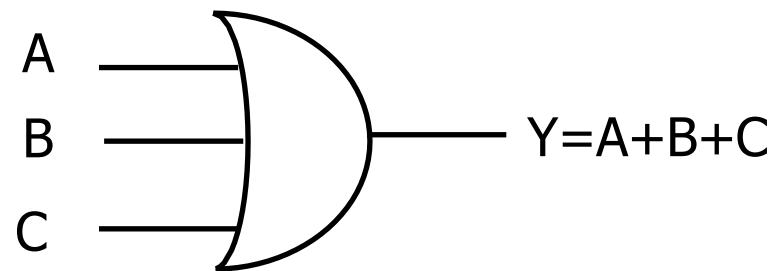
Inputs		Output
A	B	$Y = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

# Three Input OR Gate

3 – Input OR Gate using 2 – Input OR Gate



Symbol : 3 – Input OR Gate



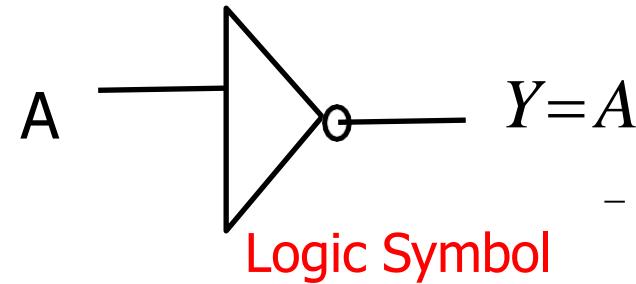
Truth Table : 3 – Input OR Gate

Input			Output $Y=A+B+C$
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

## NOT Gate (Inverter)

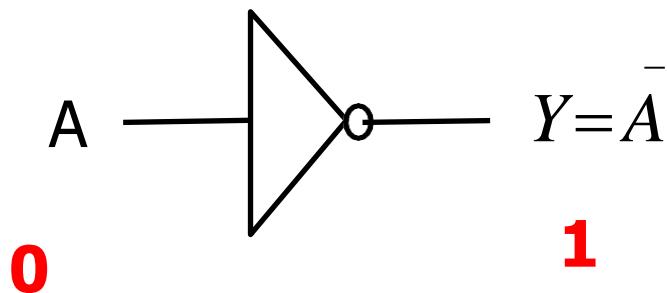
---

- ✓ A NOT gate, also called inverter, has only one input and of course only one output.
- ✓ It is a device whose output is always the complement of its input.
- ✓ That is, the output of a NOT gate assumes the logic 1 state when its input is in logic 0 state and vice versa.



# NOT Gate

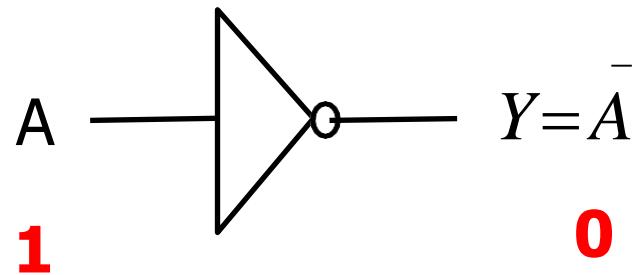
---



Input	Output
A	$\bar{Y} = \bar{A}$
0	1

# NOT Gate

---



Input	Output
A	$\bar{Y} = \bar{A}$
0	1
1	0

## Universal Gates (NAND and NOR Gate)

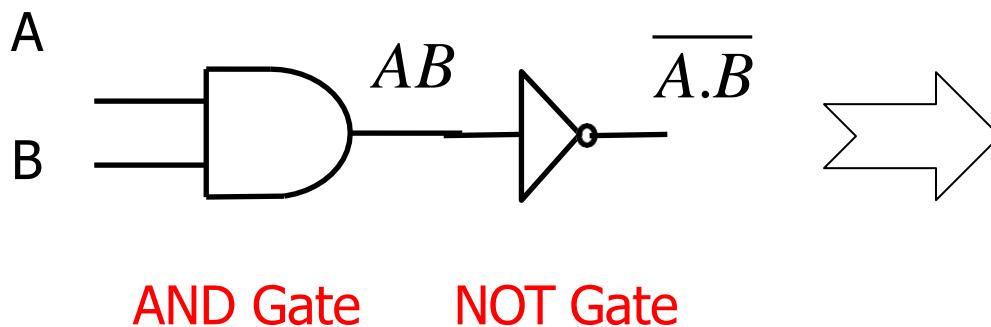
---

- ✓ NAND and NOR gates are Universal Gates.
- ✓ Both NAND and NOR gates can perform all the three basic logic functions (AND, OR and NOT).
- ✓ Therefore, AOI logic can be converted to NAND logic or NOR logic

# NAND Gate

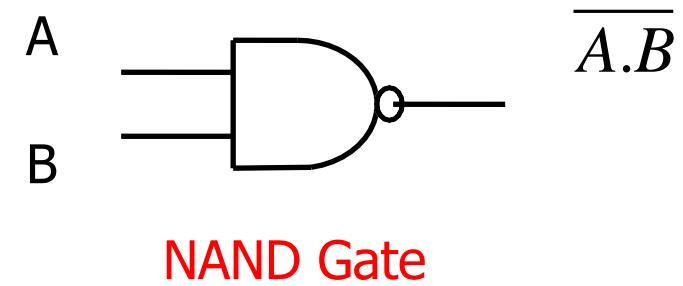
---

- ✓ NAND means NOT AND i.e. AND output is inverted.
- ✓ So NAND gate is a combination of an AND gate and a NOT gate.



AND Gate

NOT Gate

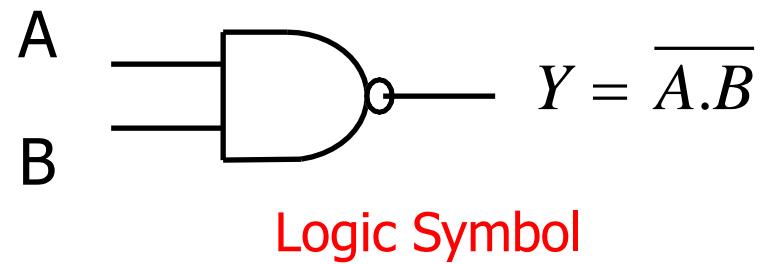


NAND Gate

## NAND Gate

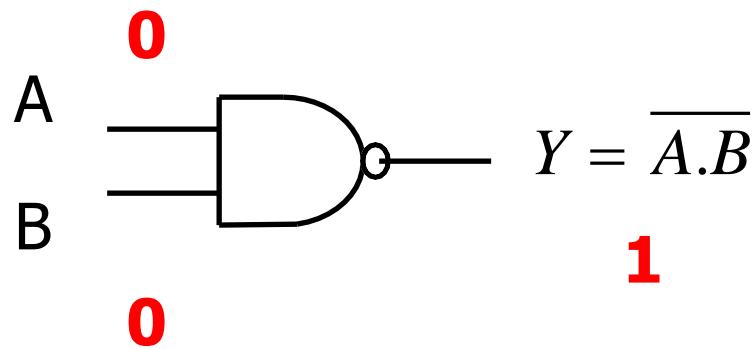
---

- ✓ The output is logic 0 level, only when all the inputs are logic 1 level.
- ✓ For any other combination of inputs, the output is a logic 1 level.



# NAND Gate

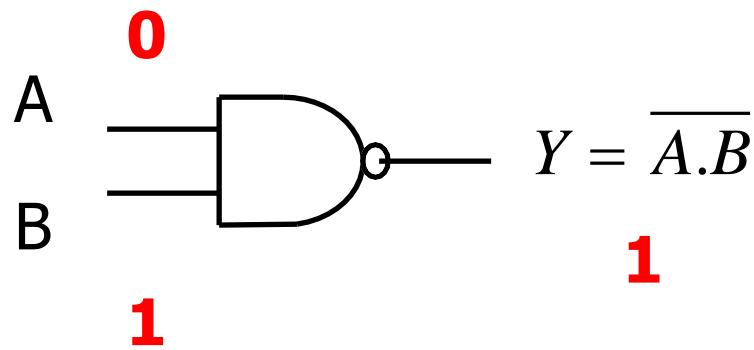
---



Inputs		Output
A	B	$Y = \overline{A \cdot B}$
0	0	1

# NAND Gate

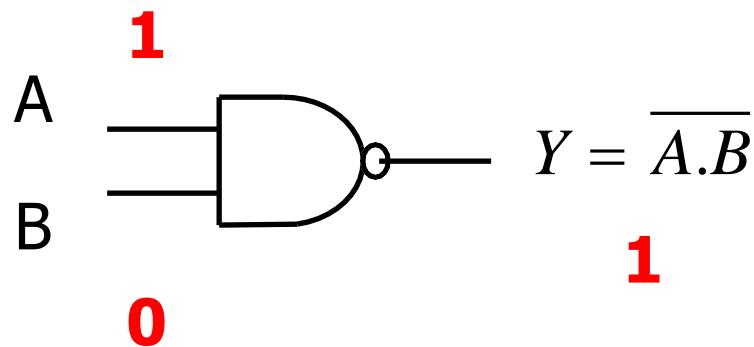
---



Inputs		Output
A	B	$Y = \overline{A.B}$
0	0	1
0	1	1

# NAND Gate

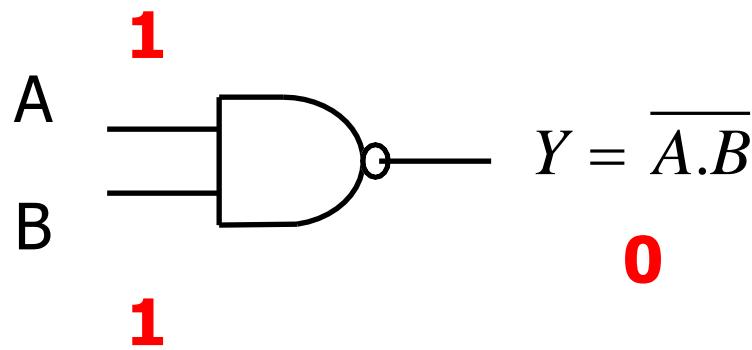
---



Inputs		Output
A	B	$Y = \overline{A.B}$
0	0	1
0	1	1
1	0	1

# NAND Gate

---

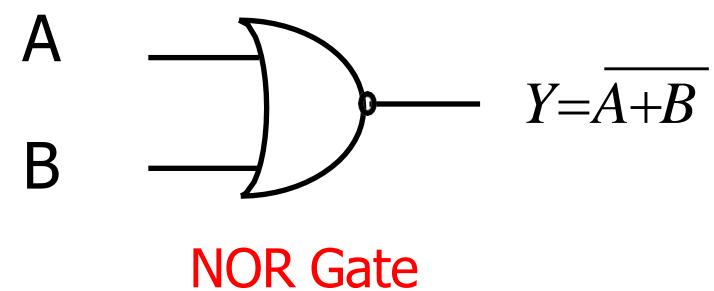
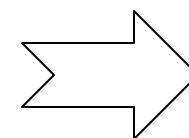
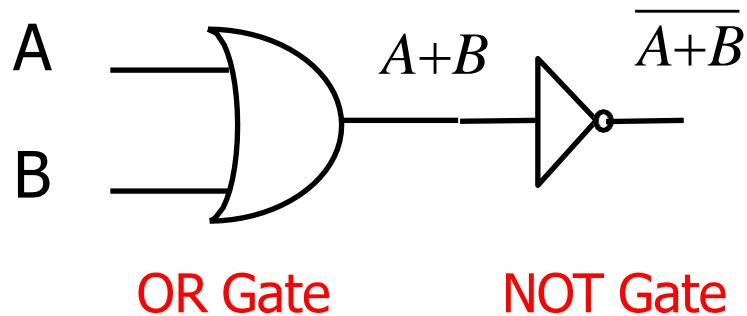


Inputs		Output
A	B	$Y = \overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

# NOR Gate

---

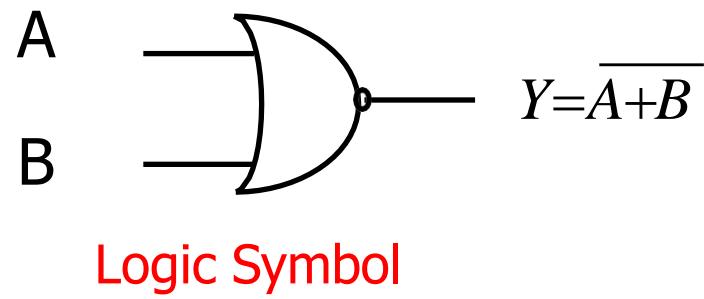
- ✓ NOR means NOT OR i.e. OR output is inverted.
- ✓ So NOR gate is a combination of an OR gate and a NOT gate.



## NOR Gate

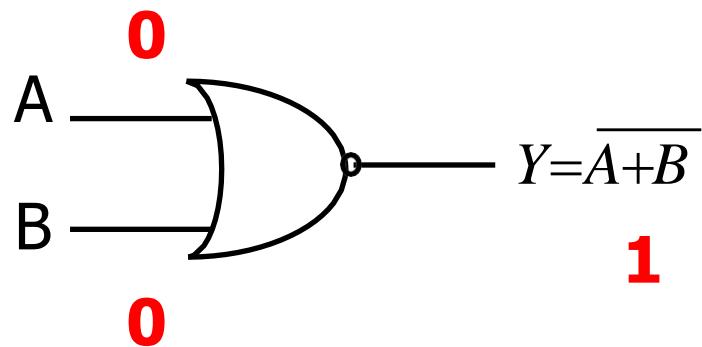
---

- ✓ The output is logic 1 level, only when all the inputs are logic 0 level.
- ✓ For any other combination of inputs, the output is a logic 0 level.



# NOR Gate

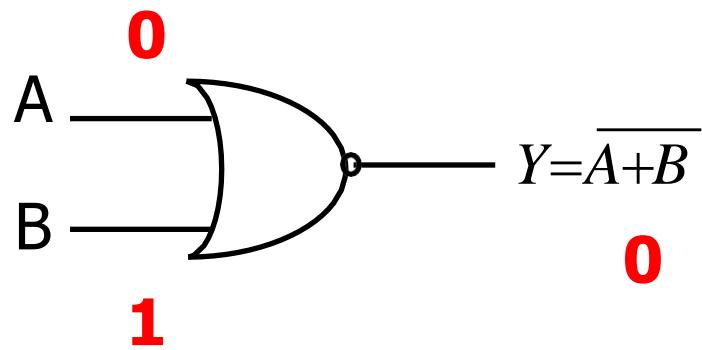
---



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

# NOR Gate

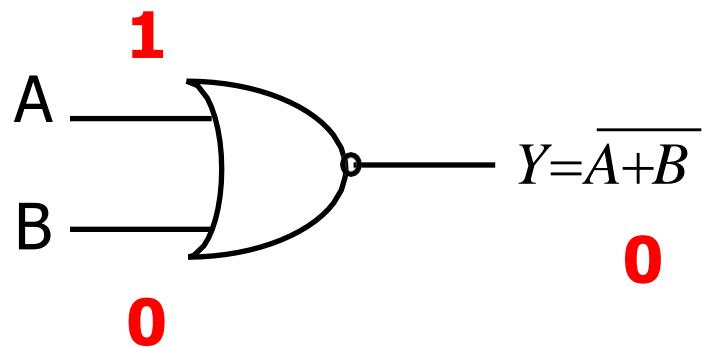
---



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

# NOR Gate

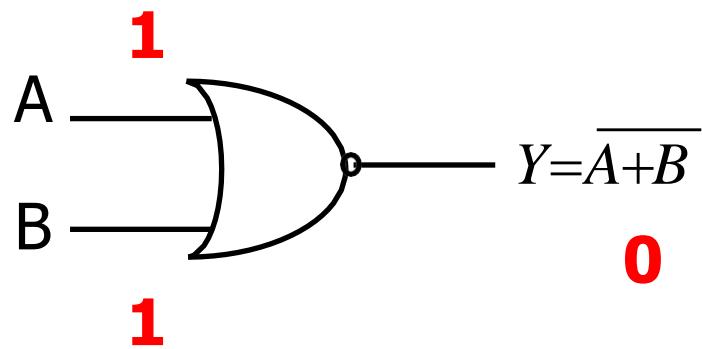
---



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

# NOR Gate

---

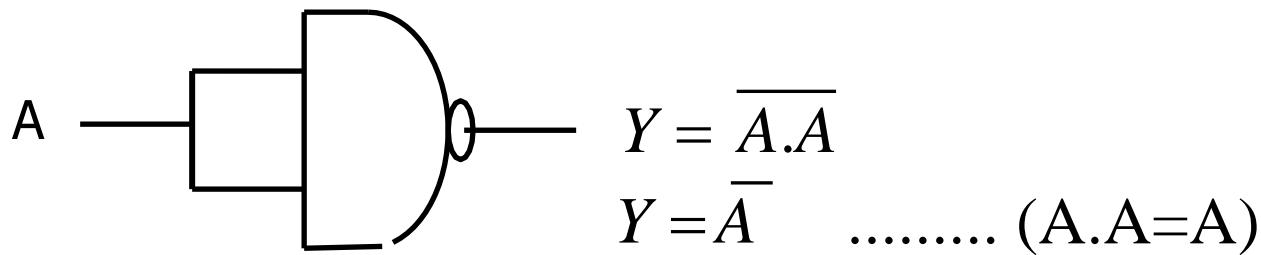
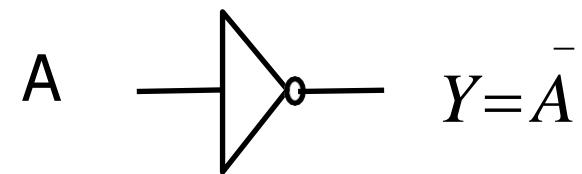


Inputs		Output
A	B	$Y = \overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

# Universal Gate

---

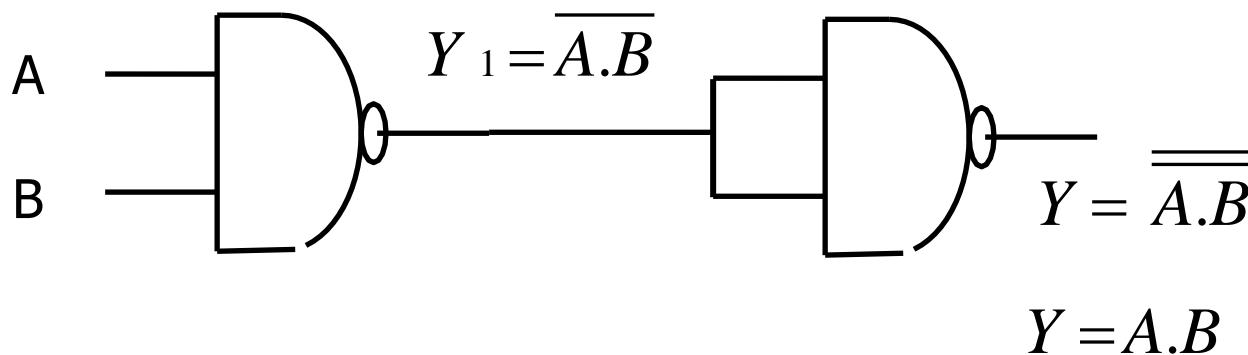
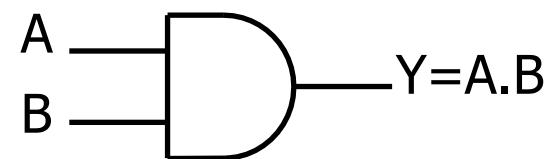
## NOT Gate using NAND Gate



# Universal Gate

---

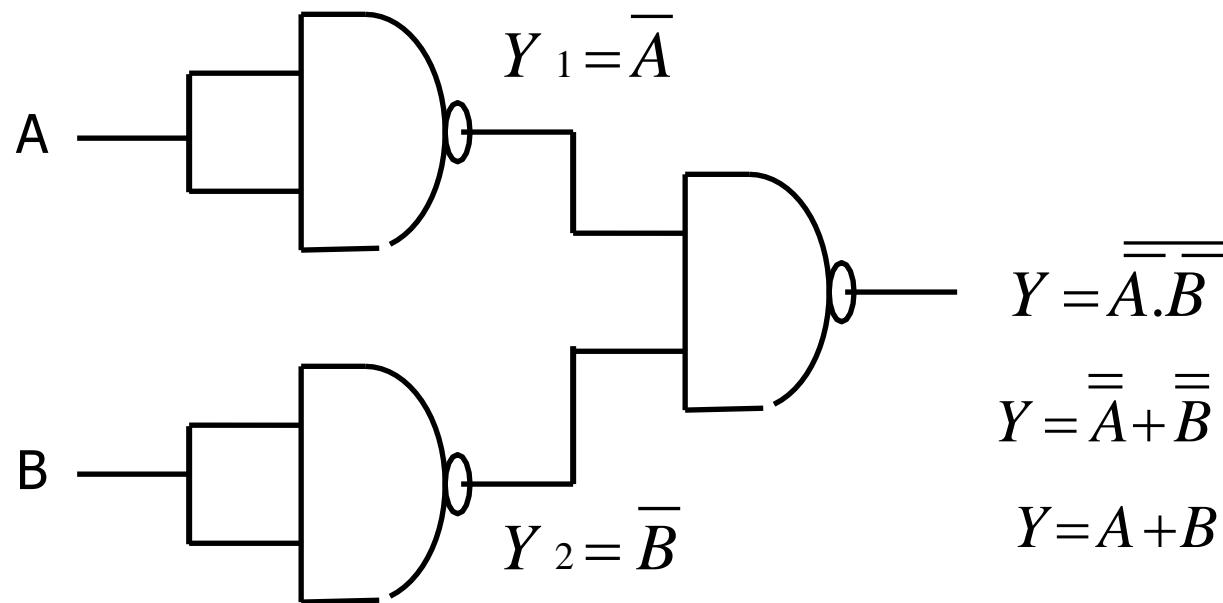
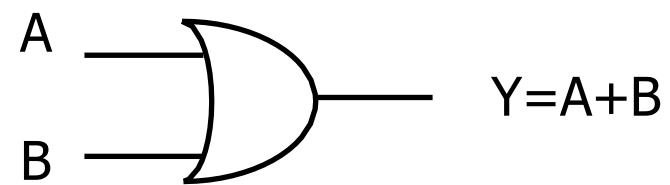
## AND Gate using NAND Gate



# Universal Gate

---

## OR Gate using NAND Gate



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

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

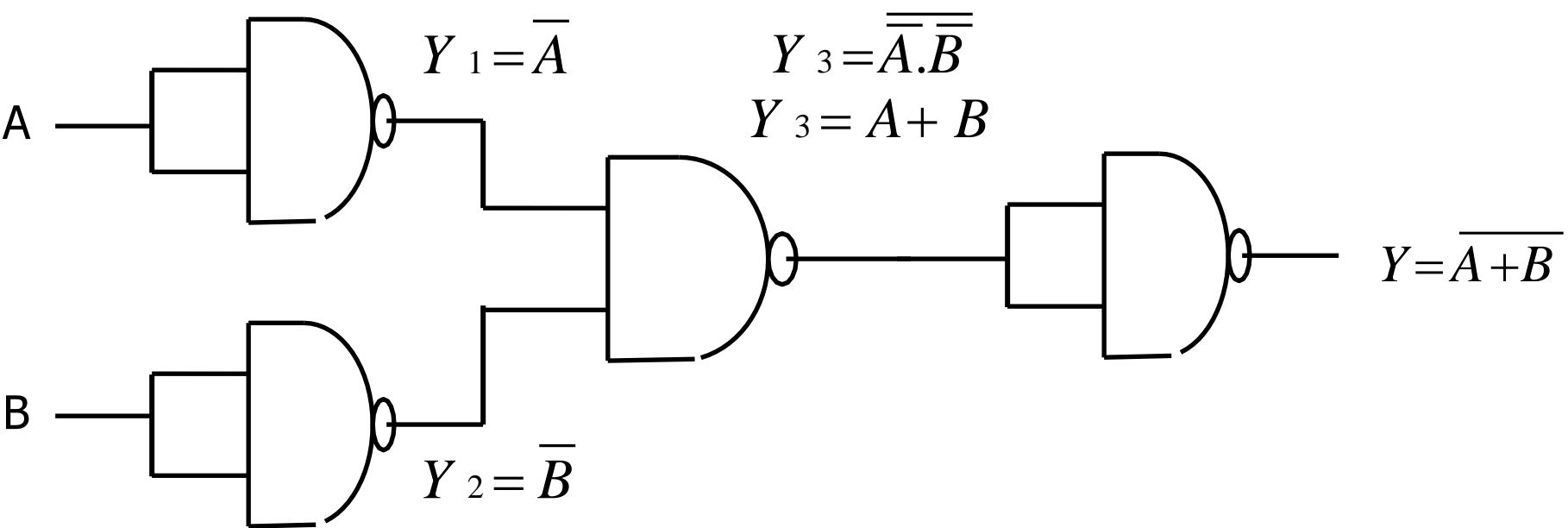
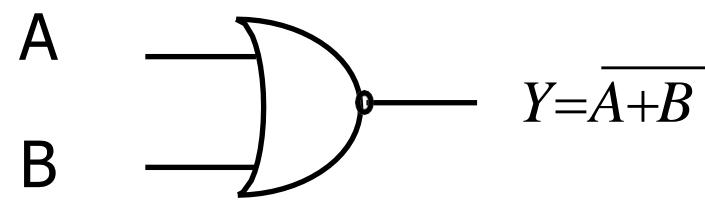
$$Y = A + B$$

( Demorgan's Theorem)

# Universal Gate

---

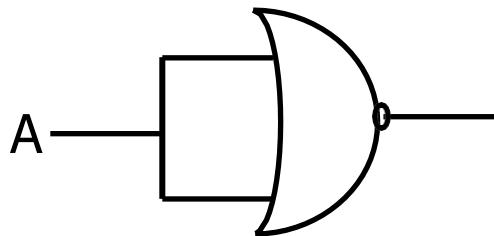
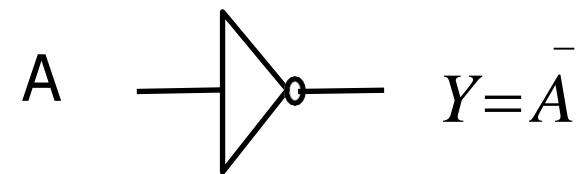
## NOR Gate using NAND Gate



# Universal Gate

---

## NOT Gate using NOR Gate

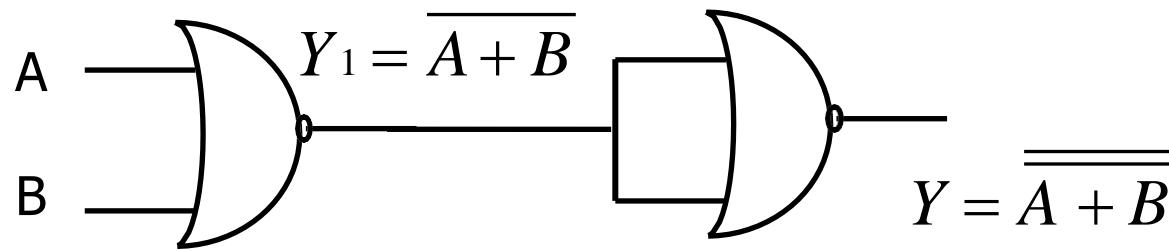
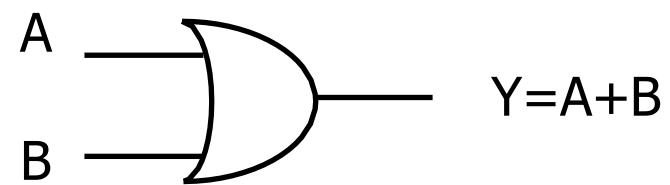


$$Y = \overline{A + A}$$
$$Y = \overline{\overline{A}} \quad \dots\dots\dots (A+A=A)$$

# Universal Gate

---

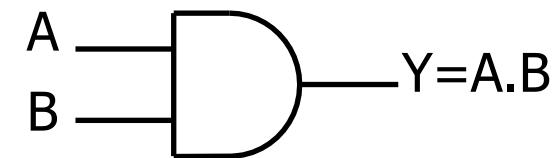
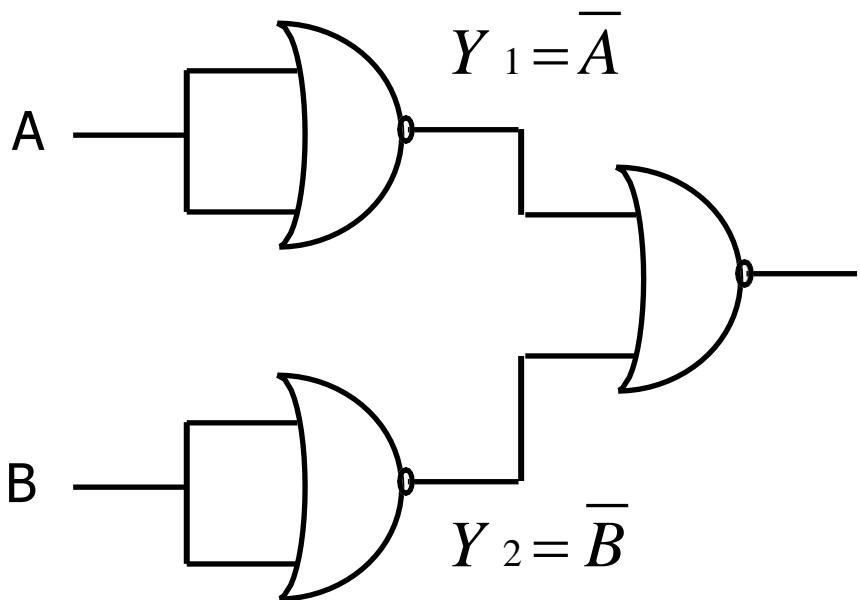
## OR Gate using NOR Gate



# Universal Gate

---

## AND Gate using NOR Gate



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

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

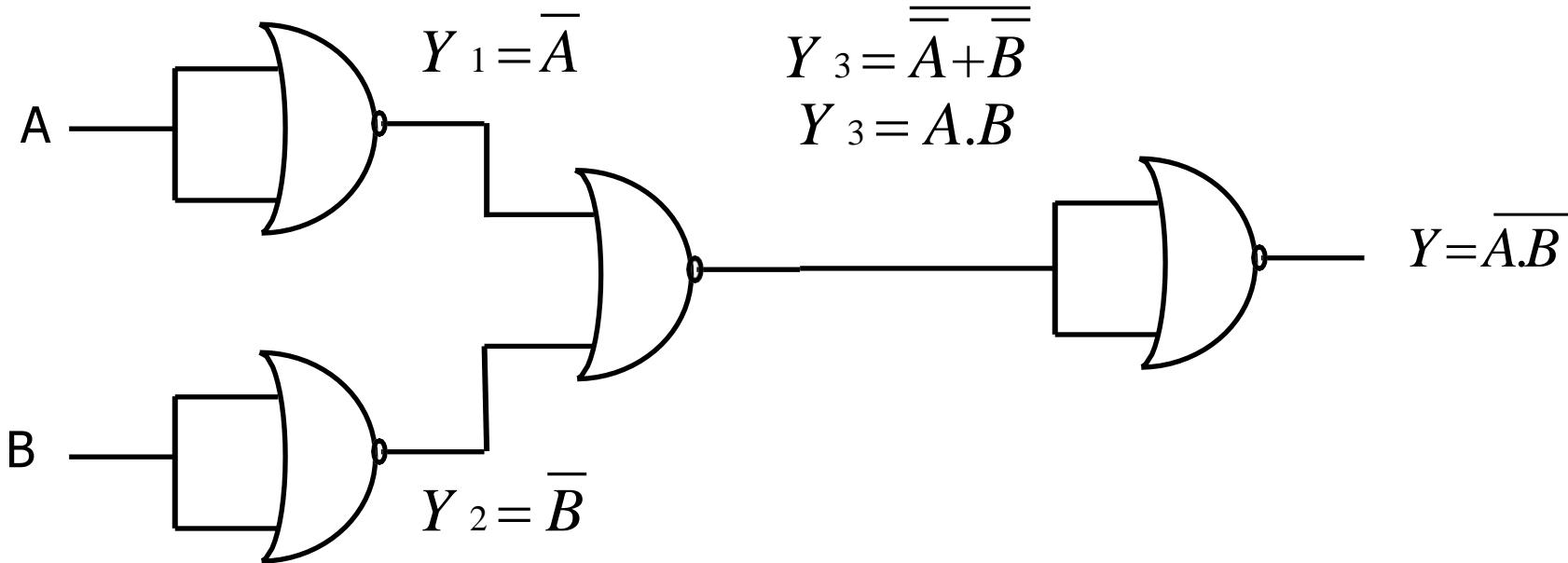
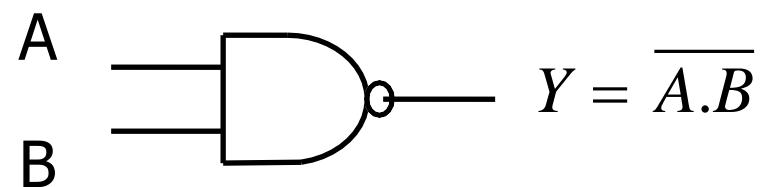
$$Y = A \cdot B$$

( Demorgan's Theorem)

# Universal Gate

---

## NAND Gate using NOR Gate



## Special Purpose Gate – Ex-OR Gate

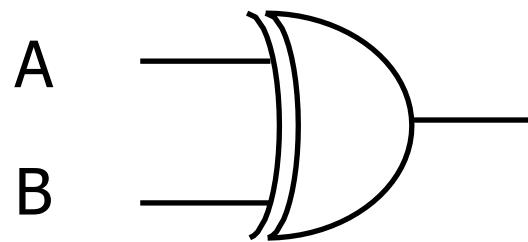
---

- ✓ An Ex-OR gate is two input, one output logic circuit.
- ✓ The output assumes the logic 1 state, when one and only one of its two inputs assumes a logic 1 state.
- ✓ Under the conditions when both the inputs assume the logic 0 state or logic 1 state, the output assumes logic 0.

## Ex-OR Gate

---

✓ If input variables are represented by A and B and the output variable by Y the representation for the output of this gate is as



$$Y = A \oplus B$$

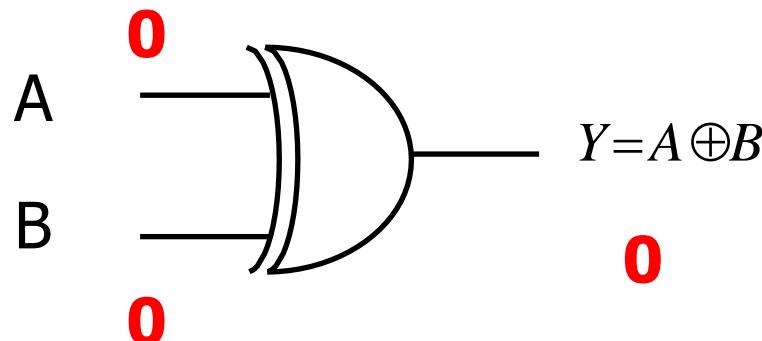
$$Y = \bar{A} \bar{B} + A \bar{B}$$

Logic Symbol

Logic Expression

# Ex-OR Gate

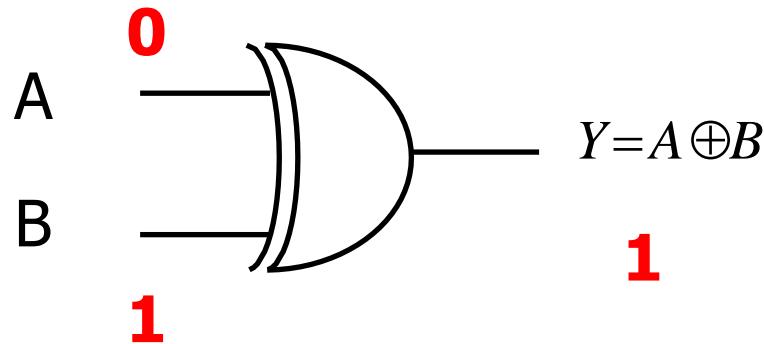
---



Inputs		Output
A	B	$Y = A \oplus B$
0	0	0

# Ex-OR Gate

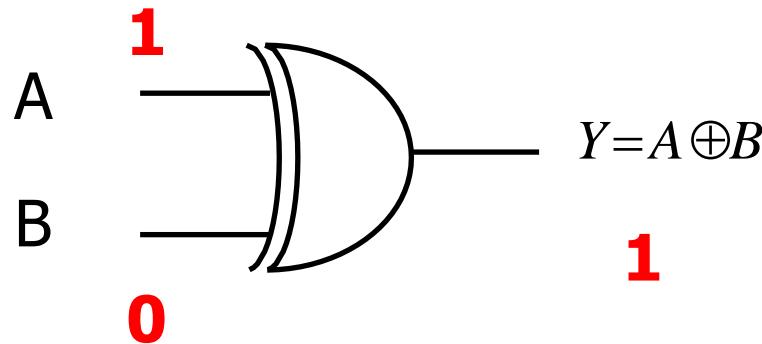
---



Inputs		Output
A	B	$Y = A \oplus B$
0	0	0
0	1	1

# Ex-OR Gate

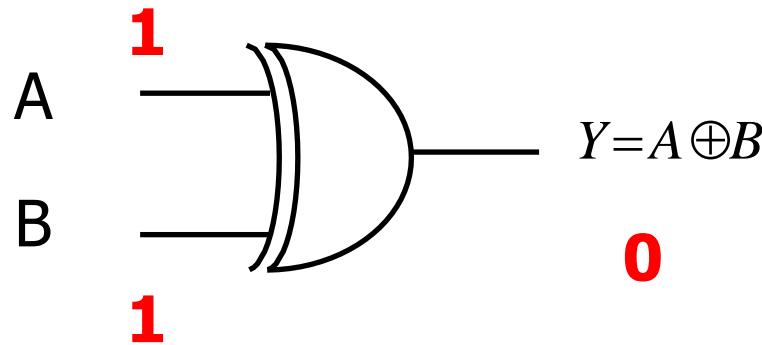
---



Inputs		Output
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1

# Ex-OR Gate

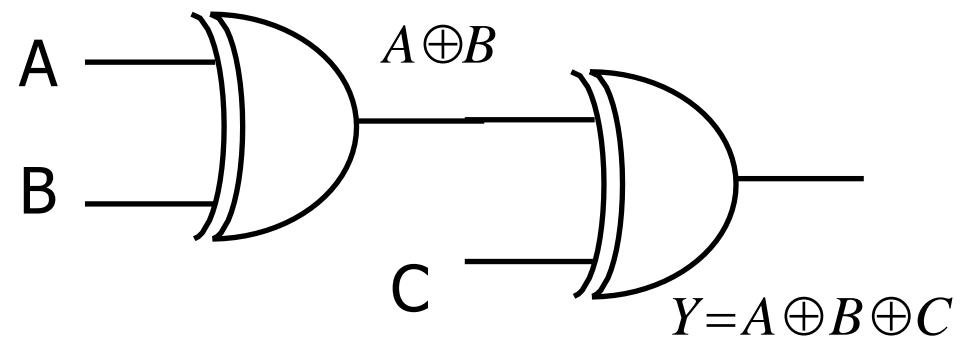
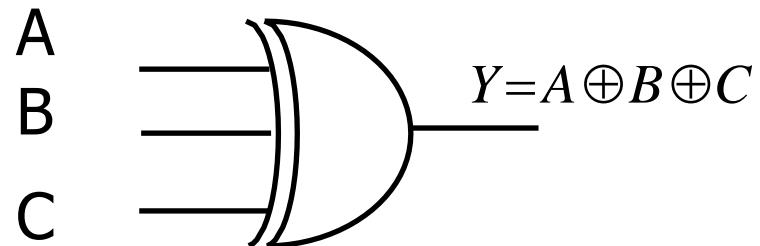
---



Inputs		Output
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

## 3 – Input Ex-OR Gate

---



INPUT			OUTPUT
A	B	C	$Y = A \oplus B \oplus C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

## Special Purpose Gate – Ex-NOR Gate

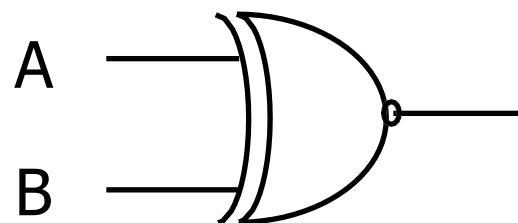
---

- ✓ An Ex-NOR gate is two input, one output logic circuit.
- ✓ The output assumes a logic 0 state, when one of the input assumes a logic 0 state and other a logic 1 state.
- ✓ The output assumes a logic 1 state only when both the inputs assume a logic 0 state or when both the inputs assume a logic state.

## Ex-NOR Gate

---

✓ If input variables are represented by A and B and the output variable by Y the representation for the output of this gate is as



$$Y = A \odot B$$

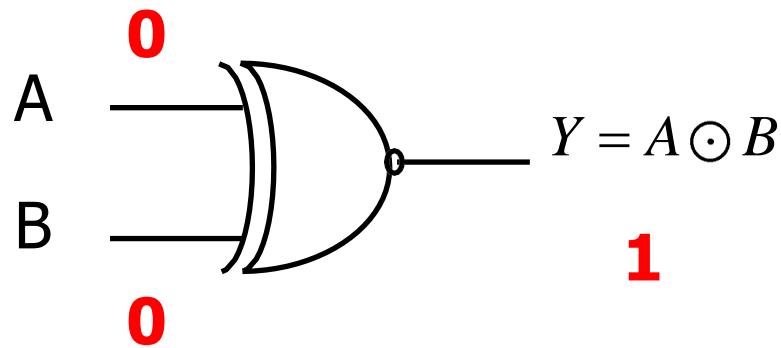
$$Y = AB + \bar{A} \bar{B}$$

Logic Symbol

Logic Expression

# Ex-NOR Gate

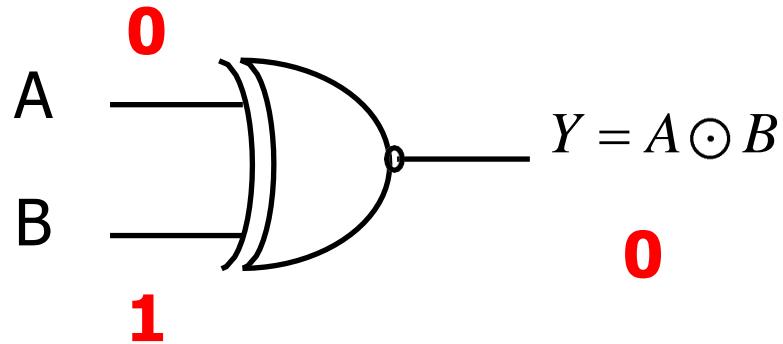
---



Inputs		Output
A	B	$Y = A \odot B$
0	0	1

# Ex-NOR Gate

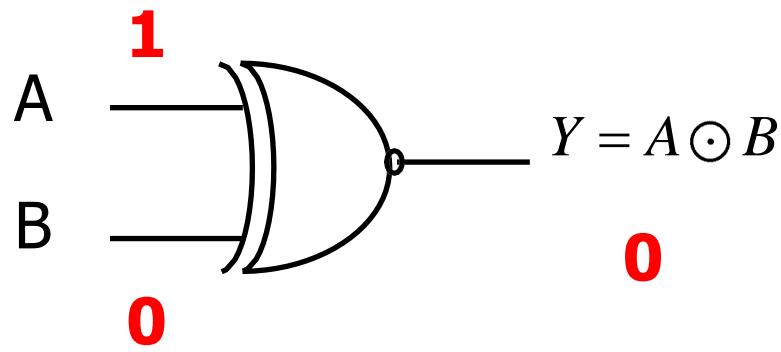
---



Inputs		Output
A	B	$Y = A \odot B$
0	0	1
0	1	0

# Ex-NOR Gate

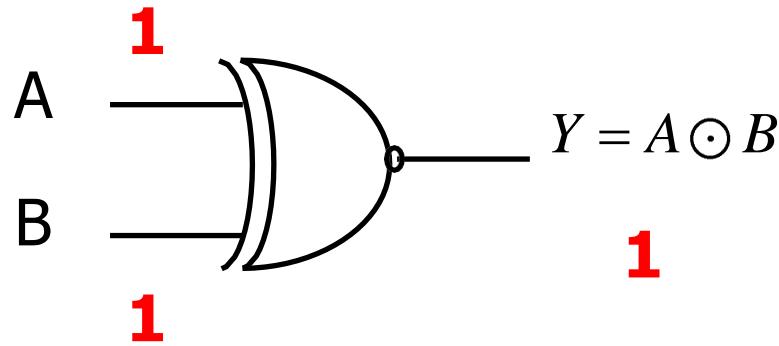
---



Inputs		Output
A	B	$Y = A \odot B$
0	0	1
0	1	0
1	0	0

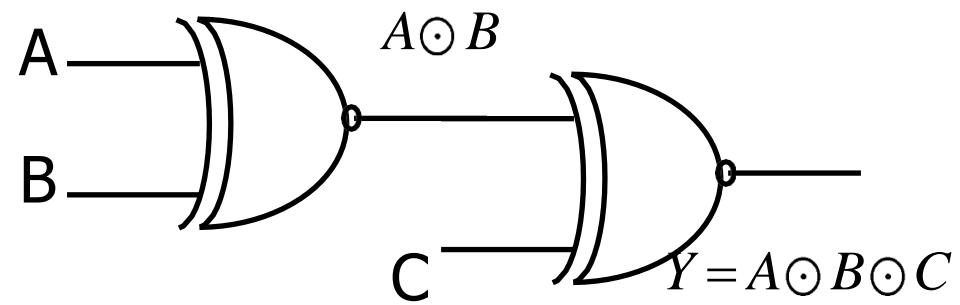
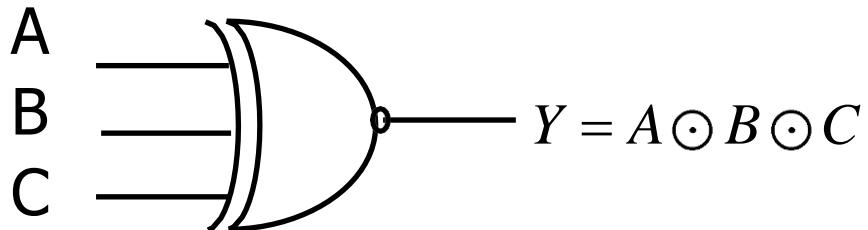
# Ex-NOR Gate

---



Inputs		Output
A	B	$Y = A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

## 3 – Input Ex-NOR Gate



INPUT			OUTPUT
A	B	C	$Y = A \odot B \odot C$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

# Boolean Algebra

---

- ✓ Boolean Algebra is used to analyze and simplify the digital (Logic) circuit.
- ✓ Since it uses only the binary numbers i.e. 0 and 1 it is also called as “Binary Algebra” or “Logical Algebra”.

# Boolean Algebra

---

- ✓ The rules of Boolean Algebra are different from those of the conventional algebra.
- ✓ It is invented by George Boole in the year 1854.

## ➤ Axioms

- ✓ Axioms or postulates of Boolean algebra are set of logical expressions that we accept without proof and upon which we can build a set of useful theorems.
- ✓ Actually, axioms are nothing more than the definitions of the three basic logic operations that we have already discussed AND, OR and INVERT.

## ➤ Axioms

### AND Operation

Axiom 1:       $0 \cdot 0 = 0$

Axiom 2:       $0 \cdot 1 = 0$

Axiom 3:       $1 \cdot 0 = 0$

Axiom 4:       $1 \cdot 1 = 1$

## ➤ Axioms

### OR Operation

Axiom 5:  $0 + 0 = 0$

Axiom 6:  $0 + 1 = 1$

Axiom 7:  $1 + 0 = 1$

Axiom 8:  $1 + 1 = 1$

## ➤ Axioms

### NOT Operation

Axiom 9:  $\bar{1} = 0$

Axiom 10:  $\bar{0} = 1$

## ➤ Inversion Law (or Complementation Law)

- ✓ The term complement means to invert i.e. to change 0's to 1's and 1's to 0's.

Law 1:  $\bar{1} = 0$

Law 2:  $\bar{0} = 1$

Law 3: If  $A=0$ , then  $\bar{A} = 1$

Law 4: If  $A=1$ , then  $\bar{A} = 0$

Law 5:  $\bar{\bar{A}} = A$  (Double Inversion Law)

## ➤ AND Laws

Law 1:	$A \cdot 0 = 0$	Null Law
Law 2:	$A \cdot 1 = A$	Identity Law
Law 3:	$A \cdot A = A$	
Law 4:	$A \cdot \bar{A} = 0$	

## ➤ OR Laws

Law 1:  $A + 0 = A$  Null Law

Law 2:  $A + 1 = 1$  Identity Law

Law 3:  $A + \bar{A} = A$

Law 4:  $A + \bar{A} = 1$

## ➤ Commutative Laws

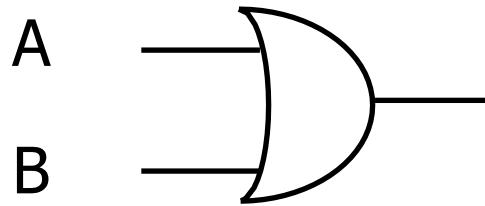
Law 1:  $A+B = B+A$

- ✓ This Law states that, A OR B is the same as B OR A i.e. the order in which the variables are ORed is immaterial.
- ✓ This means that it makes no difference which input of an OR gate is connected to A and which to B.

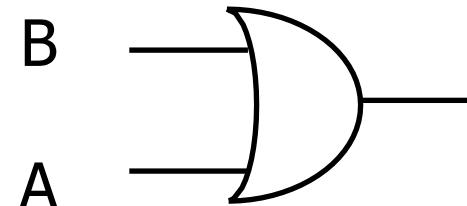
# Boolean Algebra

Proof:

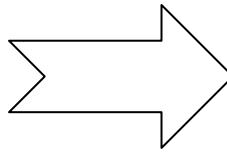
$$A+B$$



$$B+A$$



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



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

## ➤ Commutative Laws

- ✓ This law can be extended to any number of variables. For example,

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

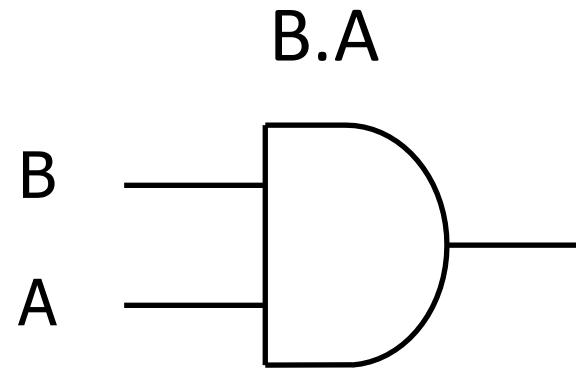
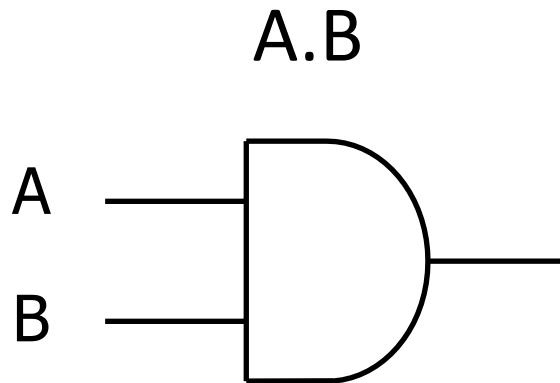
## ➤ Commutative Laws

Law 2:       $A \cdot B = B \cdot A$

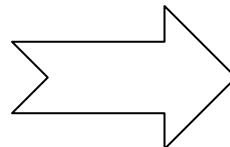
- ✓ This Law states that, A AND B is the same as B AND A i.e. the order in which the variables are ANDed is immaterial.
- ✓ This means that it makes no difference which input of an AND gate is connected to A and which to B.

# Boolean Algebra

Proof:



Inputs		Output
A	B	$Y = A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1



Inputs		Output
B	A	$Y = B \cdot A$
0	0	0
0	1	0
1	0	0
1	1	1

## ➤ Commutative Laws

✓ This law can be extended to any number of variables. For example,

$$A \cdot B \cdot C = B \cdot C \cdot A = C \cdot A \cdot B = B \cdot A \cdot C$$

## ➤ Associative Laws

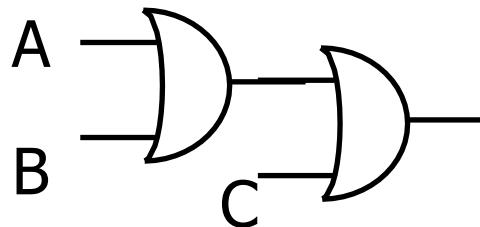
Law 1:  $(A+B)+C = A+(B+C)$

- ✓ A OR B ORed with C is the same as A ORed with B OR C.
- ✓ This law states that the way the variables are grouped and ORed is immaterial.

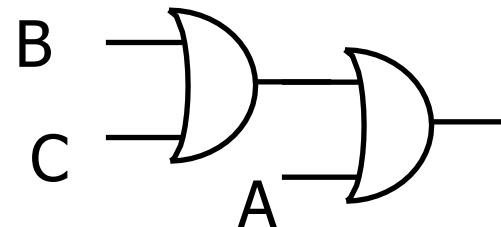
# Boolean Algebra

Proof:

$$(A+B)+C$$



$$A+(B+C)$$



A	B	C	A+B	(A+B)+C
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

A	B	C	B+C	A+(B+C)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

## ➤ Associative Laws

✓ This law can be extended to any number of variables. For example,

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

## ➤ Associative Laws

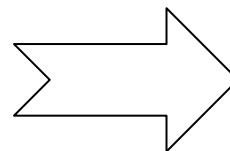
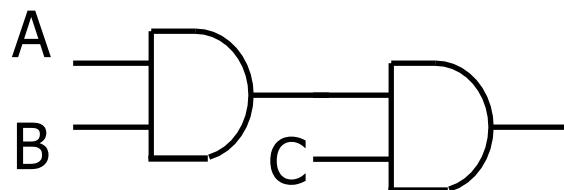
Law 2:  $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

- ✓ A AND B ANDed with C is the same as A ANDed with B AND C.
- ✓ This law states that the way the variables are grouped and ANDed is immaterial.

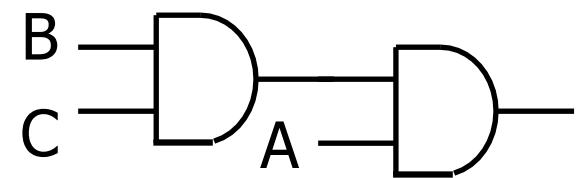
# Boolean Algebra

Proof:

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



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



A	B	C	A.B	(A.B).C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

A	B	C	B.C	A.(B.C)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

## ➤ Associative Laws

✓ This law can be extended to any number of variables. For example,

$$A.(B.C.D) = (A.B.C).D = (A.B).(C.D)$$

## ➤ Distributive Laws

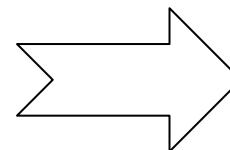
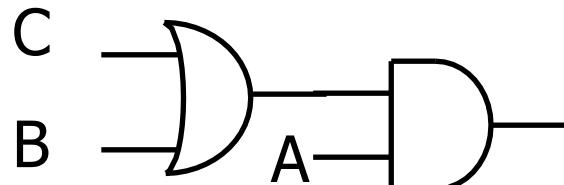
Law 1:  $A(B+C) = AB+AC$

✓ This law states that ORing of several variables and ANDing the result with a single variable is equivalent to ANDing that single variable with each of the several variables and then ORing the products.

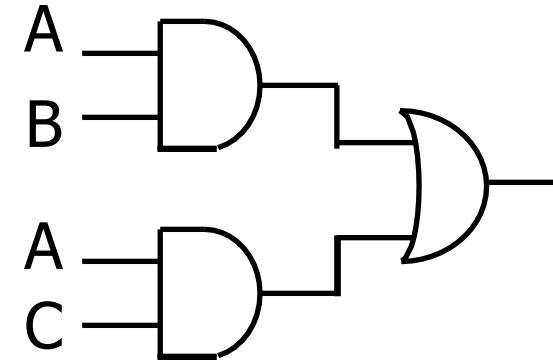
# Boolean Algebra

Proof:

$$A.(B+C)$$



$$AB+AC$$



A	B	C	$B+C$	$A(B+C)$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

A	B	C	$AB$	$AC$	$AB+AC$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

## ➤ Distributive Laws

✓ This law can be extended to any number of variables. For example,

$$ABC(D+E) = ABCD + ABCE$$

$$AB(CD+EF) = ABCD + ABEF$$

## ➤ Distributive Laws

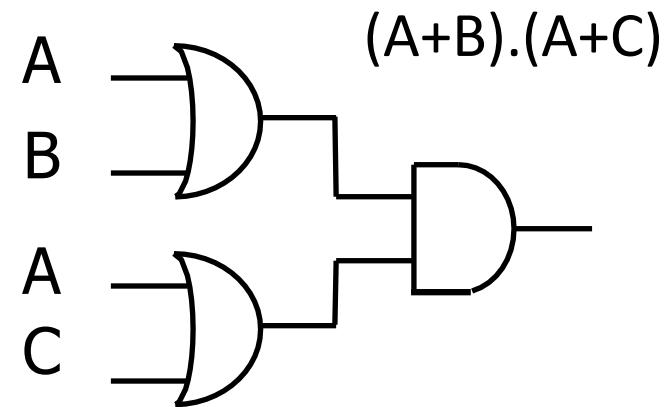
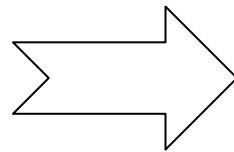
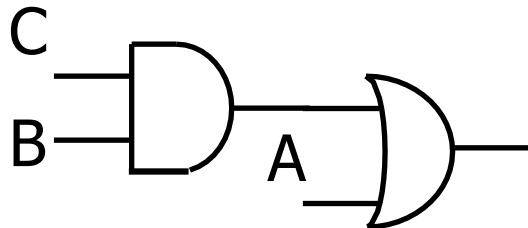
Law 2:  $A+BC = (A+B).(A+C)$

✓ This law states that ANDing of several variables and ORing the result with a single variable is equivalent to ORing that single variable with each of the several variables and then ANDing the products.

# Boolean Algebra

Proof:

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



A	B	C	BC	A+BC
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A	B	C	A+B	A+C	(A+B) (A+C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

## ➤ Redundant Literal Rule

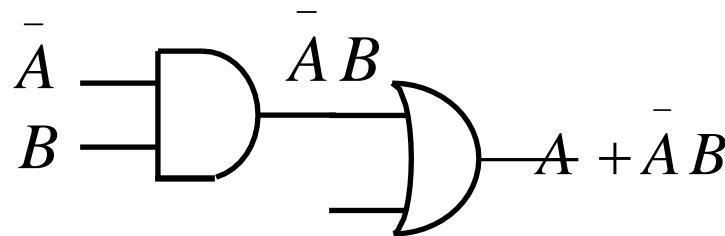
Law 1:  $A + \bar{A}B = A + B$

✓ This law states that ORing of variable with the AND of the complement of that variable with another variable, is equal to the ORing of the two variables

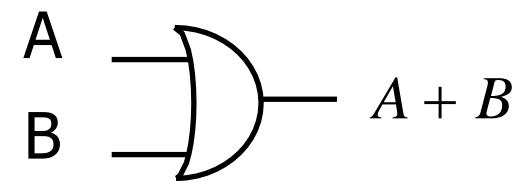
# Boolean Algebra

Proof:

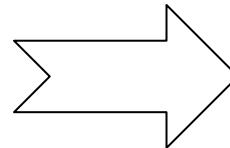
$$A + \bar{A}B$$



$$A + B$$



A	B	$\bar{A}B$	$A + \bar{A}B$
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	1



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

## ➤ Redundant Literal Rule

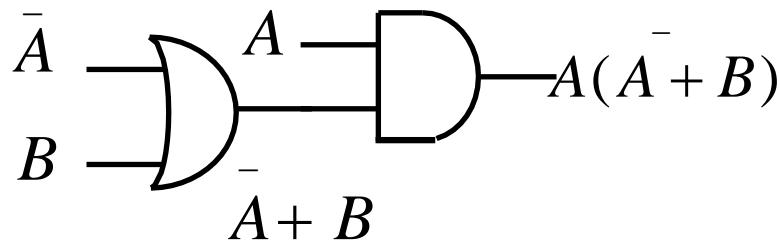
Law 2:  $A(\bar{A} + B) = A \cdot B$

- ✓ This law states that ANDing of variable with the OR of the complement of that variable with another variable, is equal to the ANDing of the two variables

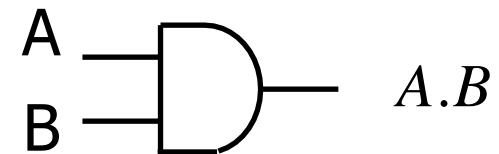
# Boolean Algebra

Proof:

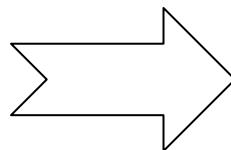
$$A(\bar{A} + B)$$



$$A \cdot B$$



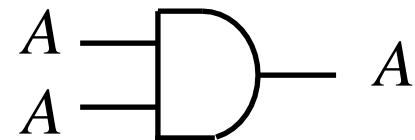
A	B	$\bar{A} + B$	$A(\bar{A} + B)$
0	0	1	0
0	1	1	0
1	0	0	0
1	1	1	1



A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

## ➤ Idempotence Laws

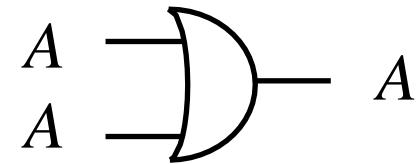
Law 1:  $A \cdot A = A$



- ✓ Idempotence means the same value
- ✓ If  $A=0$ , then  $A \cdot A = 0 \cdot 0 = 0 = A$
- ✓ If  $A=1$ , then  $A \cdot A = 1 \cdot 1 = 1 = A$
- ✓ This law states that ANDing of a variable with itself is equal to that variable only.

## ➤ Idempotence Laws

Law 2:  $A + A = A$



- ✓ Idempotence means the same value
- ✓ If  $A=0$ , then  $A+A = 0+0 = 0 = A$
- ✓ If  $A=1$ , then  $A+A = 1+1 = 1 = A$
- ✓ This law states that ORing of a variable with itself is equal to that variable only.

## ➤ Absorption Laws

Law 1:  $A + A \cdot B = A$

- ✓ This law states that ORing of a variable with AND of that variable and another variable is equal to that variable itself.
- ✓ Therefore,

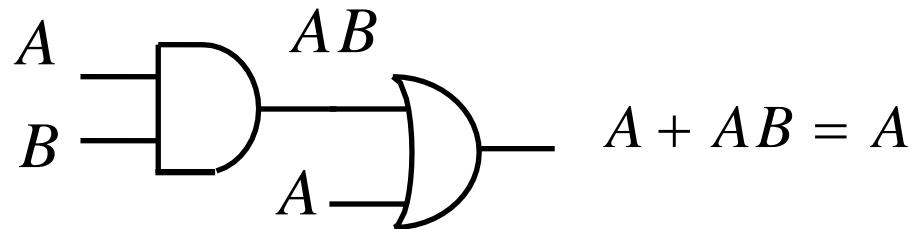
$$A + A \cdot \text{Any Term} = A$$

# Boolean Algebra

Proof:

$$A + A \cdot B$$

$$A$$



A	B	$A \cdot B$	$A + A \cdot B$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

## ➤ Absorption Laws

Law 2:  $A(A + B) = A$

- ✓ This law states that ANDing of a variable with OR of that variable and another variable is equal to that variable itself.
- ✓ Therefore,

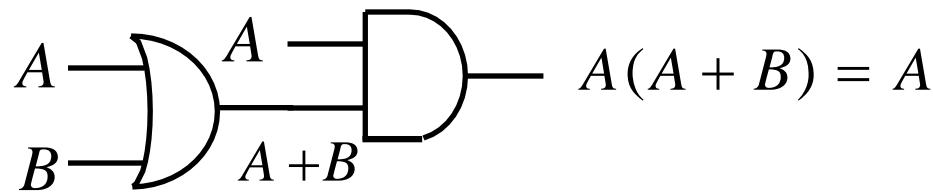
$$A \cdot (A + \text{Any Term}) = A$$

# Boolean Algebra

---

Proof:

$$A(A + B) = A$$



A	B	$A + B$	$A(A + B)$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

## ➤ De-Morgan's Theorem

First Theorem:

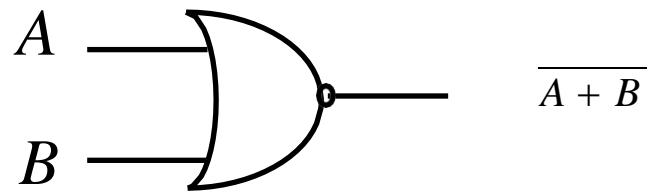
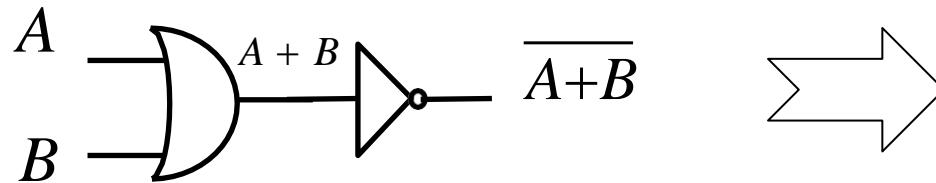
$$\overline{A + B} = \overline{\overline{A}} \cdot \overline{\overline{B}}$$

- ✓ This theorem states that the complement of a sum of variables is equal to the product of their individual complements.
- ✓ What it means is that the complement of two or more variables ORed together, is the same as the AND of the complements of each of the individual variables

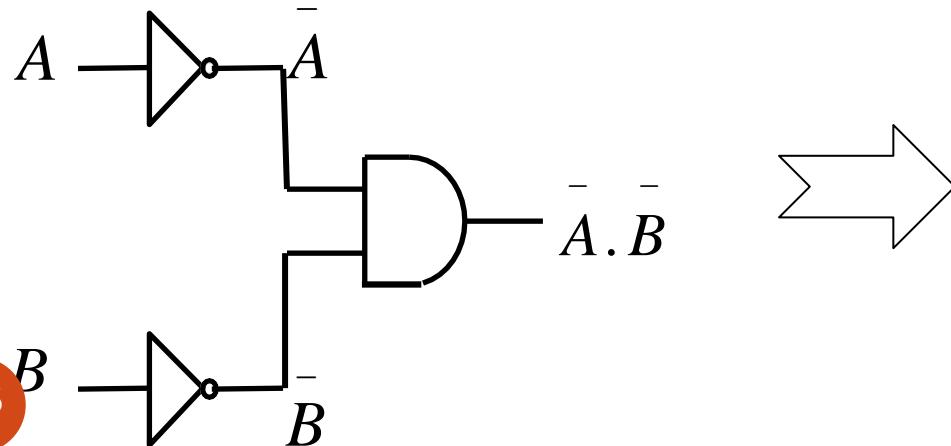
# Boolean Algebra

## Proof: Logic Diagram

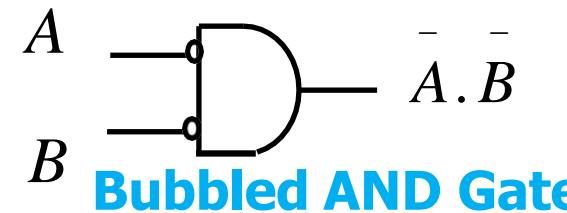
L.H.S.



R.H.S.



**NOR Gate**



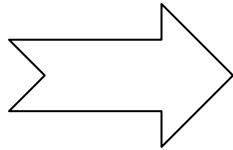
**Bubbled AND Gate**

# Boolean Algebra

## Proof: Logic Table

$\overline{A + B}$

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



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

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

## ➤ De-Morgan's Theorem

✓ This law can be extended to any number of variables. For example,

$$\overline{A+B+C+D+\dots} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{D} \cdot \dots$$

$$\overline{AB+CD+EFG+\dots} = (\overline{A}\overline{B}) \cdot (\overline{C}\overline{D}) \cdot (\overline{E}\overline{F}\overline{G}) \cdot \dots$$

## ➤ De-Morgan's Theorem

Second Theorem:

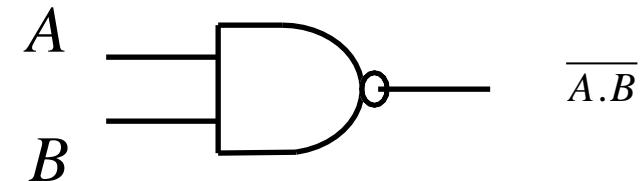
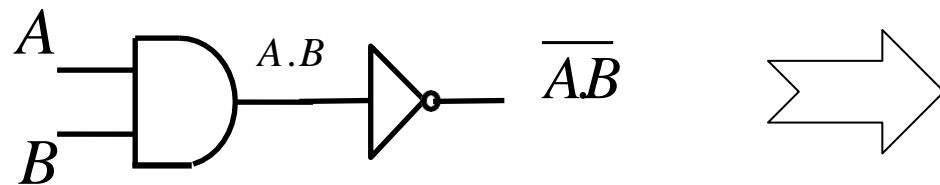
$$\overline{A \cdot B} = \overline{\overline{A}} + \overline{\overline{B}}$$

- ✓ This theorem states that the complement of a product of variables is equal to the sum of their individual complements.
- ✓ What it means is that the complement of two or more variables ANDed together, is the same as the OR of the complements of each of the individual variables

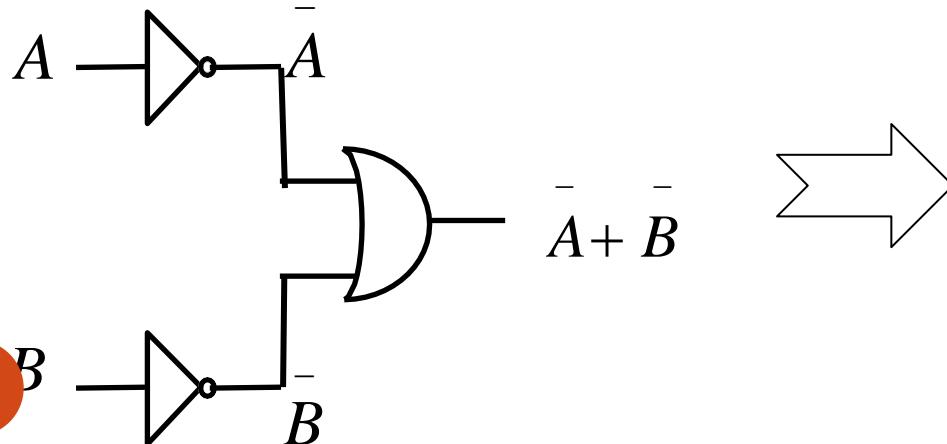
# Boolean Algebra

## Proof: Logic Diagram

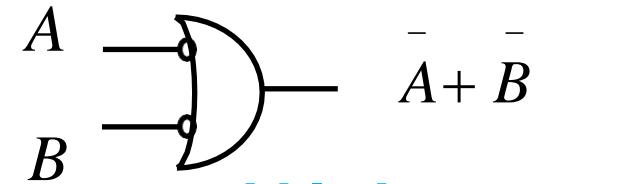
L.H.S.



R.H.S.



**NAND Gate**

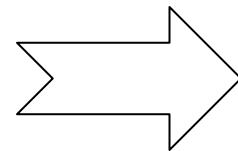


**Bubbled OR Gate**

# Boolean Algebra

Proof:

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



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

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

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

## ➤ De-Morgan's Theorem

- ✓ This law can be extended to any number of variables. For example,

$$\overline{A \cdot B \cdot C \cdot D \dots} = \overline{A} + \overline{B} + \overline{C} + \overline{D} \dots$$

$$\overline{(AB)(CD)(EFG) \dots} = \overline{AB} + \overline{CD} + \overline{EFG} + \dots$$

✓ Duality represents relation between expressions in positive logic system and expression in negative logic system.

- ✓ The distinction between positive and negative logic system is important.
- ✓ An OR gate in positive logic system becomes an AND gate in negative logic system and vice versa.
- ✓ Positive & negative logics thus give rise to a basic duality in all Boolean identities.

- ✓ When changing from one logic system to another 0 becomes 1 and 1 becomes 0.
- ✓ Furthermore, an AND gate becomes an OR gate and an OR gate becomes AND gate.

## Duality

---

- ✓ Given Boolean identity, we can produce a dual identity by changing all ‘+’ signs to “ signs, all “ signs to ‘+’ signs and complementing all 0's and 1's.
- ✓ The variables are not complemented in this process.

# Examples of Dual Identities

---

Sr. No.	Given Expression	Dual
1	$\overline{0} = 1$	$\overline{1} = 0$
2	$0.\overline{1} = 0$	$1 + \overline{0} = 1$
3	$0.\overline{0} = 0$	$1 + 1 = 1$
4	$1.\overline{1} = 1$	$0 + 0 = 0$
5	$A.\overline{0} = 0$	$A + \overline{1} = 1$
6	$A.\overline{1} = A$	$A + \overline{0} = A$

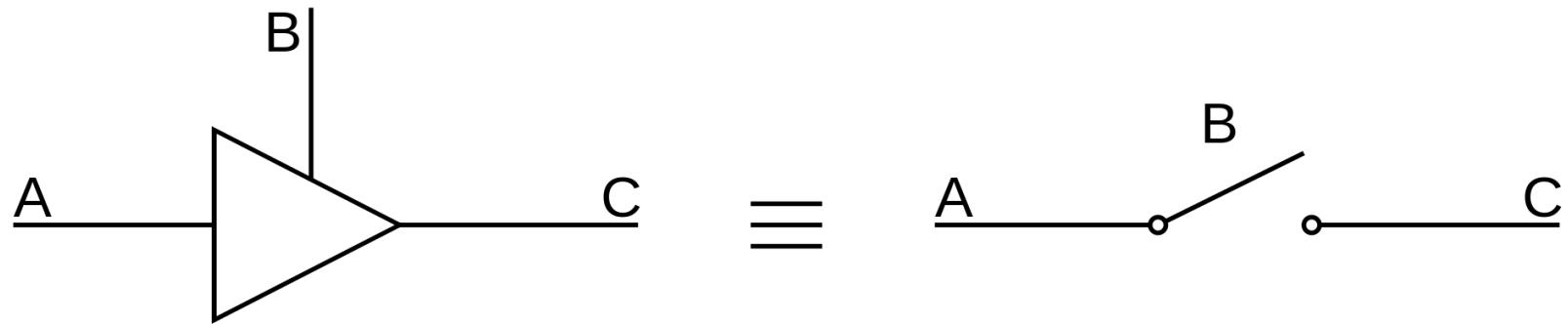
# Examples of Dual Identities

---

Sr. No.	Given Expression	Dual
7	$A \cdot B = B \cdot A$	$A + B = B + A$
8	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$
9	$A \cdot (B + C) = A \cdot B + A \cdot C$	$A + BC = (A + B)(A + C)$
10	$A \cdot (A + B) = A$	$A + AB = A$
11	$A \cdot (A \cdot B) = A \cdot B$	$A + A + B = A + B$
12	$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$
13	$(\overline{A} + B)(\overline{A} + C) = (\overline{A} + B)(\overline{A} + C)$	$\overline{AB} + \overline{AC} = \overline{AB} + \overline{AC}$

# TRI-STATE LOGIC

- In digital electronics **three-state**, **tri-state**, or **3-state** logic allows an output port to assume a high impedance state, effectively removing the output from the circuit, in addition to the 0 and 1 logic levels.
- Three-state outputs are implemented in many registers, bus drivers and flip-flops in the 7400 and 4000 series as well as in other types, but also internally in many integrated circuits.



A tristate buffer can be thought of as a switch. If  $B$  is on, the switch is closed. If  $B$  is off, the switch is open

## Example 1

---

Reduce the following Boolean Expression using Boolean Laws:

$$A\bar{B} + \bar{A}B + A.B + \bar{A}\bar{B}$$

## Example 1

---

Reduce the following Boolean Expression using Boolean Laws:

$$A\bar{B} + \bar{A}B + A.B + \bar{A}\bar{B}$$

$$= A\bar{B} + \bar{A}B + A.B + \bar{A}.B$$

$$= A\bar{B} + A.B + \bar{A}.B + \bar{A}.B$$

$$= A.(\bar{B} + B) + \bar{A}(B + B) \quad (B + B = 1)$$

$$= A + A \quad (A + \bar{A} = 1)$$

$$= 1$$

$$A\bar{B} + \bar{A}B + A.B + \bar{A}\bar{B} = 1$$

## Example 2

---

Reduce the following Boolean Expression using Boolean Laws:

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

## Example 2

---

Reduce the following Boolean Expression using Boolean Laws:

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

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

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

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

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

$$= C(B + A)(\bar{B} + B) \quad (\quad \text{Distributive Law})$$

$$= C(B + A) \quad (\quad \bar{B} + B = 1)$$

$$= AC + BC$$

## Example 3

---

Realize  $Y=AB+AC$  using one OR gate and one AND gate

## Example 3

---

Realize  $Y=AB+AC$  using one OR gate and one AND gate

$$Y = AB + AC$$

A.B is one product term  
Hence requires 1 AND gate

A.C is one product term  
Hence requires 1 AND gate

A.C & A.B is one sum term  
Hence requires 1 OR gate

Hence to implement  $Y=AB+AC$  equation we require 2 AND gates and 1 OR gate

But we have to use only 1 AND gate and 1 OR gate

Hence simplification is necessary

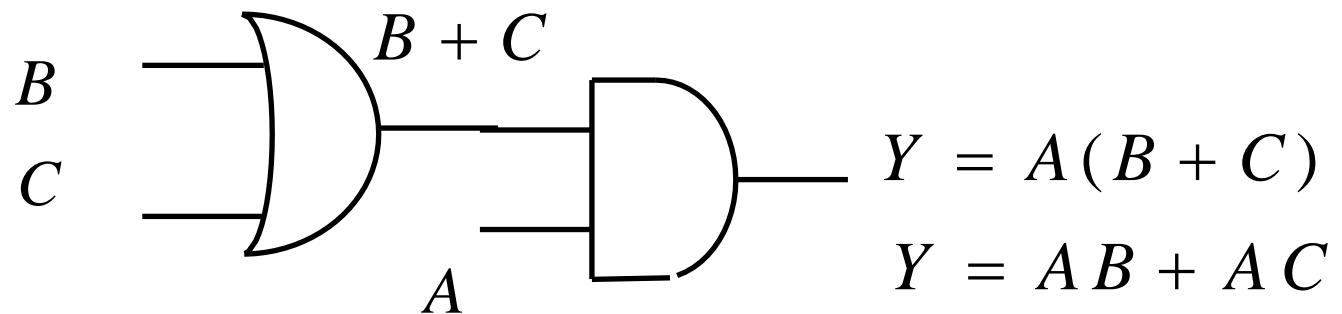
## Example 3

Continue.....

---

$$Y = AB + AC$$

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



## Example 4

---

Prove that:

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

## Example 4

---

Prove that:

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

$$\begin{aligned}\text{L.H.S} &= A + \overline{A}B \\&= A(1) + \overline{A}B \\&= A(1+B) + \overline{A}B \\&= A + AB + \overline{A}B \\&= A + B(A + \overline{A}) \\&= A + B \quad (A + \overline{A} = 1)\end{aligned}$$

$$\text{L.H.S} = \text{R.H.S}$$

## Example 5

---

Prove that:

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

## Example 5

---

Prove that:

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

$$\begin{aligned} \text{L.H.S} &= (A + B)(A + \bar{B}) \\ &= AA + A\bar{B} + AB + B\bar{B} \\ &= A + A\bar{B} + AB + 0 \quad (\quad AA=A, B\bar{B}=0) \\ &= A + A(\bar{B} + B) \\ &= A + A \quad (\quad \bar{B} + B = 1) \\ &= A \quad (\quad A + A = A) \end{aligned}$$

$$\text{L.H.S} = \text{R.H.S}$$

## Example 6

---

With the help of Boolean Laws, Prove that:

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

## Example 6

---

With the help of Boolean Laws, Prove that:

$$(A + \bar{B} + AB)(A + B).\bar{A}\bar{B} = 0$$

$$\begin{aligned} \text{L.H.S.} &= (A + \bar{B} + AB)(A + B).\bar{A}\bar{B} \\ &\quad (A + \bar{B} + AB)(A\bar{A}\bar{B} + A\bar{B}\bar{B}) \\ &= (A + \bar{B} + AB).(0) \quad (\quad A.\bar{A}=0, B.\bar{B}=0) \\ &= 0 \end{aligned}$$

$$\text{L.H.S} = \text{R.H.S}$$

## Example 7

---

With the help of Boolean Laws, Prove that:

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

## Example 7

---

With the help of Boolean Laws, Prove that:

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

$$\begin{aligned} \text{L.H.S.} &= AB + \overline{A}B + \overline{A}\overline{B} \\ &= \overline{A}B + \overline{A}\overline{B} + AB \\ &= \overline{A}(B + \overline{B}) + AB \\ &= \overline{A} + AB \\ &= (A + \overline{A})(\overline{A} + B) \\ &= 1 \cdot (\overline{A} + B) \\ &= \overline{A} + B \end{aligned}$$

$$\text{L.H.S} = \text{R.H.S}$$

$$(B + \overline{B} = 0)$$

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

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

## Example 8

---

Simplify;

$$F = XY + XYZ + XYZ + X\bar{Z}Y$$

## Example 8

---

Simplify;  $F = XY + XYZ + XYZ + X\bar{Z}Y$

$$\begin{aligned}F &= XY + XYZ + XYZ + X\bar{Z}Y \\&= XY + XYZ + X\bar{Z}Y \quad (\quad XYZ+XYZ=XYZ) \\&= XY(1 + Z + \bar{Z}) \\&= XY \quad (\quad 1 + Z + \bar{Z} = 1) \\&= XY\end{aligned}$$

## Example 9

---

Prove that;

$$AB + ABC + A\bar{B} = A$$

## Example 9

---

Prove that;

$$AB + ABC + A\bar{B} = A$$

$$L.H.S. = AB + ABC + A\bar{B}$$

$$= AB(1 + C) + A\bar{B}$$

$$= AB + A\bar{B} \quad (1 + C = 1)$$

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

$$= A \quad (B + \bar{B} = 1)$$

$$L.H.S = R.H.S$$

## IC Chip Manufacturing Process

- ✓ Small Scale Integration (SSI)
- ✓ Medium Scale Integration (MSI)
- ✓ Large Scale Integration (LSI)
- ✓ Very Large Scale Integration (VLSI)
- ✓ Ultra Large Scale Integration (ULSI)
- ✓ Giant Scale Integration (GSI)

# Logic Families

---

- Gate/transistor ratio is roughly
  - SSI      < 12 gates/chip
  - MSI      < 100 gates/chip
  - LSI      ...1K gates/chip
  - VLSI     ...10K gates/chip
  - ULSI     ...100K gates/chip
  - GSI      ...1Meg gates/chip

# Moore's Law

---

- ✓ A prediction made by Moore (a co-founder of Intel) in 1965: "... a number of transistors to double every 2 years."

# Standard Representation

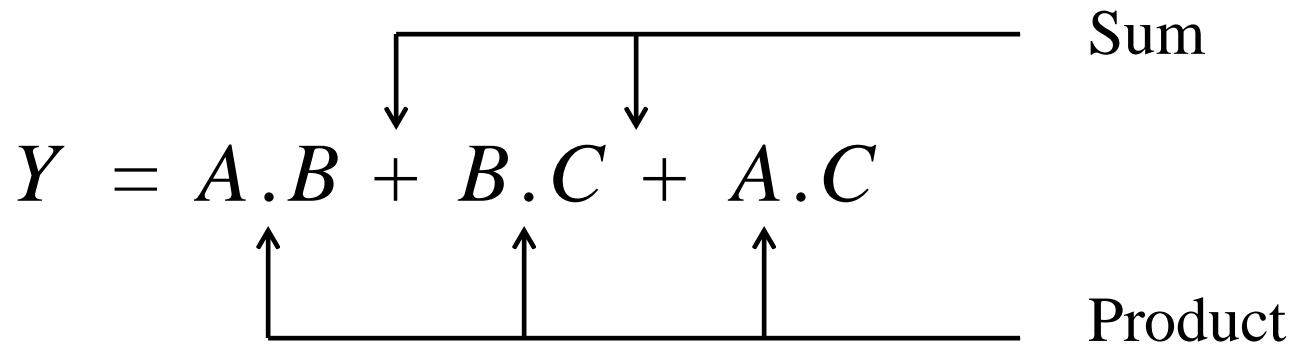
---

- Any logical expression can be expressed in the following two forms:
  - ✓ Sum of Product (SOP) Form
  - ✓ Product of Sum (POS) Form

# SOP Form

---

For Example, logical expression given  
is;



# POS Form

---

For Example, logical expression given  
is;

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

Product

Sum

## Standard or Canonical SOP & POS Forms

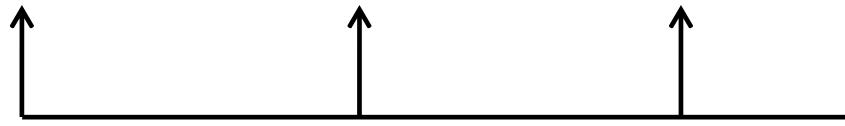
---

- ✓ We can say that a logic expression is said to be in the standard (or canonical) SOP or POS form if each product term (for SOP) and sum term (for POS) consists of all the literals in their complemented or uncomplemented form.

# Standard SOP

---

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

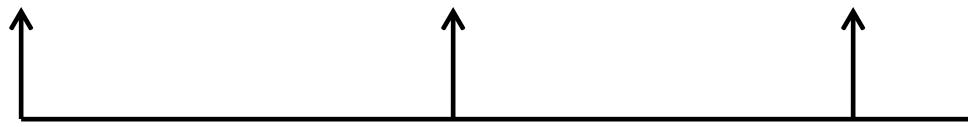


Each product term  
consists all the  
literals

## Standard POS

---

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



Each sum term  
consists all the  
 literals

# Examples

---

Sr. No.	Expression	Type
1	$Y = AB + ABC + \overline{A}BC$	Non Standard SOP
2	$Y = AB + A\overline{B} + \overline{A}\overline{B}$	Standard SOP
3	$Y = (\overline{A} + B).(A + \overline{B}).(\overline{A} + \overline{B})$	Standard POS
4	$Y = (\overline{A} + B).(A + \overline{B} + C)$	Non Standard POS

# Conversion of SOP form to Standard SOP

---

*Procedure:*

1. Write down all the terms.
2. If one or more variables are missing in any product term, expand the term by multiplying it with the sum of each one of the missing variable and its complement .
3. Drop out the redundant terms.

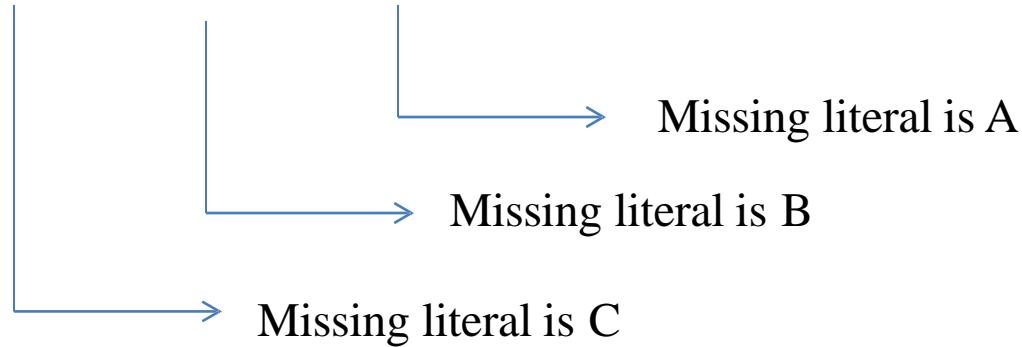
# Example 1

---

Convert given expression into its standard SOP form

$$Y = AB + A\bar{C} + BC$$

$$Y = AB + A\bar{C} + BC$$



$$Y = AB \cdot (C + \bar{C}) + A\bar{C} \cdot (B + \bar{B}) + BC \cdot (A + \bar{A})$$

A single blue arrow originates from the term  $AB \cdot (C + \bar{C})$  in the expanded expression and points to the text "Term formed by ORing of missing literal & its complement".

Term formed by ORing of  
missing  
literal & its complement

## Example 1

Continue

....

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

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

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

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



**Standard SOP form**

**Each product term consists all the literals**

# Conversion of POS form to Standard POS

---

## *Procedure:*

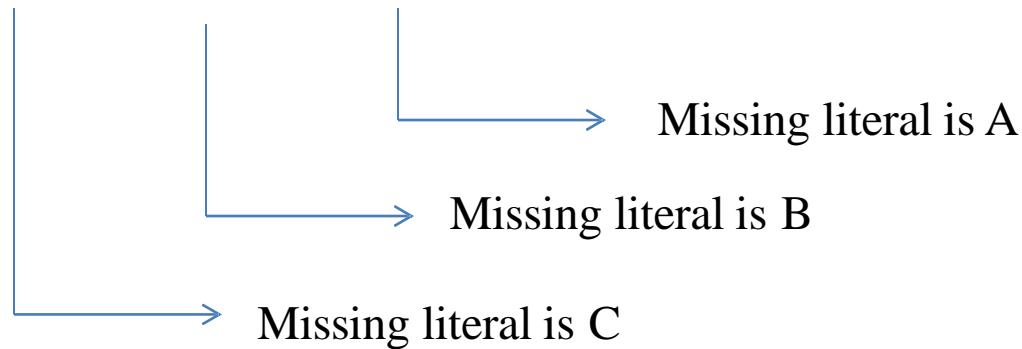
1. Write down all the terms.
2. If one or more variables are missing in any sum term, expand the term by adding the products of each one of the missing variable and its complement.
3. Drop out the redundant terms.

## Example 2

---

Convert given expression into its standard SOP form  $Y = (A + B).(A + C).(B + \bar{C})$

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



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

Three blue lines originate from the terms  $A$ ,  $B$ , and  $C$  in the expanded expression  $Y$  and converge at a single point below them. This point is labeled "Term formed by ANDing of missing literal & its complement".

## Example 2

Continue

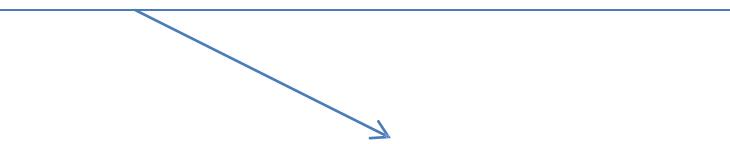
---

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

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

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

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



**Standard POS form**  
**Each sum term consists all the literals**

# Concept of Minterm and Maxterm

---

- ✓ **Minterm:** Each individual term in the standard SOP form is called as “Minterm”.
- ✓ **Maxterm:** Each individual term in the standard POS form is called as “Maxterm”.

- 
- ✓ The concept of minterm and max term allows us to introduce a very convenient shorthand notation to express logic functions

# Minterms & Maxterms for 3 variable/literal logic function

Variables			Minterms	Maxterms
A	B	C	$m_i$	$M_i$
0	0	0	$\overline{A}\overline{B}\overline{C} = m_0$	$A + B + C = M_0$
0	0	1	$\overline{A}\overline{B}C = m_1$	$A + B + \overline{C} = M_1$
0	1	0	$\overline{A}\overline{B}C = m_2$	$A + \overline{B} + C = M_2$
0	1	1	$\overline{A}BC = m_3$	$A + \overline{B} + \overline{C} = M_3$
1	0	0	$A\overline{B}\overline{C} = m_4$	$\overline{A} + B + C = M_4$
1	0	1	$A\overline{B}C = m_5$	$\overline{A} + B + \overline{C} = M_5$
1	1	0	$AB\overline{C} = m_6$	$\overline{A} + \overline{B} + C = M_6$
1	1	1	$ABC = m_7$	$\overline{A} + \overline{B} + \overline{C} = M_7$

# Representation of Logical expression using minterm

---

$$Y = \underline{ABC} + \underline{\overline{A}BC} + \underline{A\overline{B}\overline{C}} + \underline{A\overline{B}C}$$

$m_7$        $m_3$        $m_4$        $m_5$

Logical Expression      ←  
Corresponding minterms      ←

$$Y = m_7 + m_3 + m_4 + m_5$$

$$Y = \Sigma m(3, 4, 5, 7)$$

OR

$$Y = f(A, B, C) = \Sigma m(3, 4, 5, 7)$$

where  $\Sigma$  denotes sum of products

# Representation of Logical expression using maxterm

---

$$Y = (\underline{A + \overline{B} + C}) \cdot (\underline{A + B + C}) \cdot (\underline{\overline{A} + \overline{B} + C}) \leftarrow \begin{array}{l} \text{Logical Expression} \\ \text{Corresponding} \\ \text{maxterms} \end{array}$$

$M_2$        $M_0$        $M_6$

$$Y = M_2 \cdot M_0 \cdot M_6$$

$$Y = \prod M (0, 2, 6) \quad \text{OR}$$

$$Y = f(A, B, C) = \prod M (0, 2, 6)$$

where  $\prod$  denotes product of sum

# Conversion from SOP to POS & Vice versa

---

- ✓ The relationship between the expressions using minterms and maxterms is complementary.
- ✓ We can exploit this complementary relationship to write the expressions in terms of maxterms if the expression in terms of minterms is known and vice versa

## Conversion from SOP to POS & Vice versa

---

- ✓ For example, if a SOP expression for 4 variable is given by,

$$Y = \Sigma m(0,1,3,5,6,7,11,12,15)$$

- ✓ Then we can get the equivalent POS expression using complementary relationship as follows:

$$Y = \Pi M (2, 4, 8, 9, 10, 13, 14)$$

## Examples

---

1. Convert the given expression into standard form

$$Y = A + BC + ABC$$

2. Convert the given expression into standard form

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

## Karnaugh Map (K-map)

---

- ✓ In the algebraic method of simplification, we need to write lengthy equations, find the common terms, manipulate the expressions etc., so it is time consuming work.
- ✓ Thus “K-map” is another simplification technique to reduce the Boolean equation.

# Karnaugh Map (K-map)

---

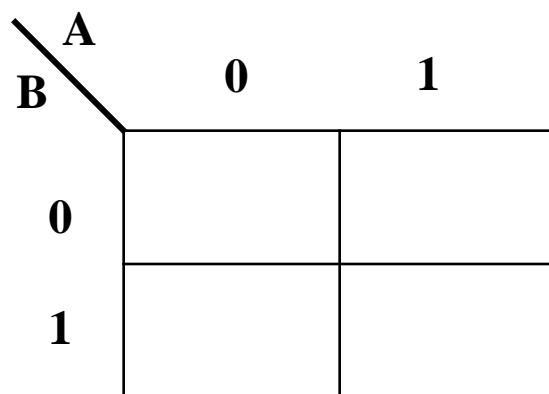
- ✓ It overcomes all the disadvantages of algebraic simplification techniques.
- ✓ The information contained in a truth table or available in the SOP or POS form is represented on K-map.

# Karnaugh Map (K-map)

---

## ➤ K-map Structure - 2 Variable

- ✓ A & B are variables or inputs
- ✓ 0 & 1 are values of A & B
- ✓ 2 variable k-map consists of 4 boxes i.e.  $2^2=4$



# Karnaugh Map (K-map)

## ➤ K-map Structure - 2 Variable

- ✓ Inside 4 boxes we have enter values of Y i.e. output

		$\bar{A}$	$A$
	$\bar{B}$	0	1
$\bar{B}$	0	$\bar{A}\bar{B}$	$\bar{A}B$
$B$	1	$A\bar{B}$	$AB$

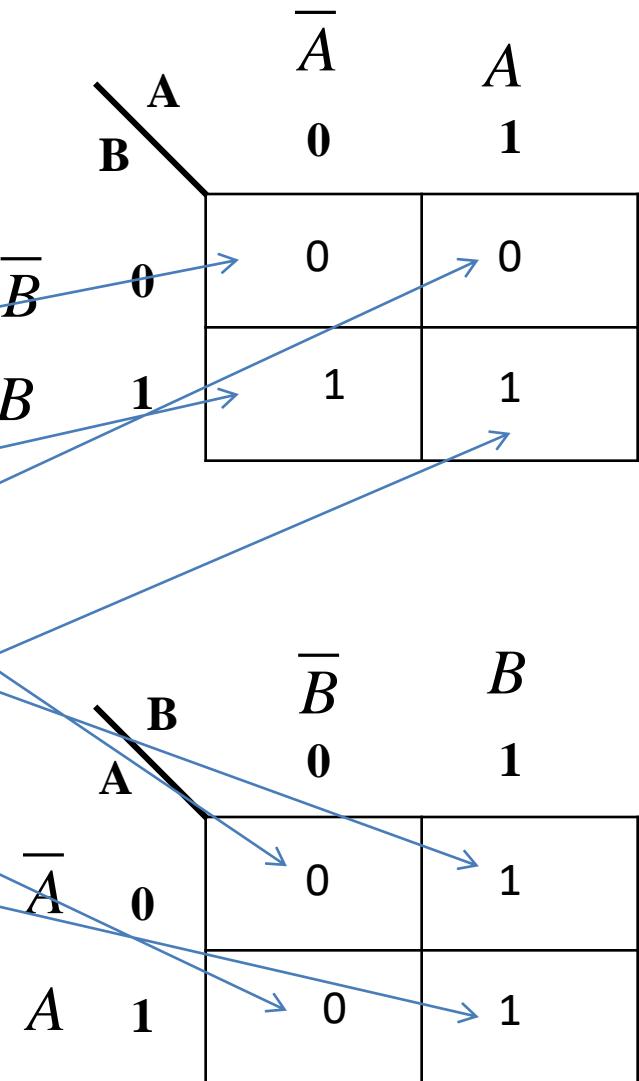
		$\bar{A}$	$A$
	$\bar{B}$	0	1
$\bar{B}$	0	$m_0$	$m_1$
$B$	1	$m_2$	$m_3$

K-map & its associated minterms

# Karnaugh Map (K-map)

✓ Relationship between Truth Table & K-map

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

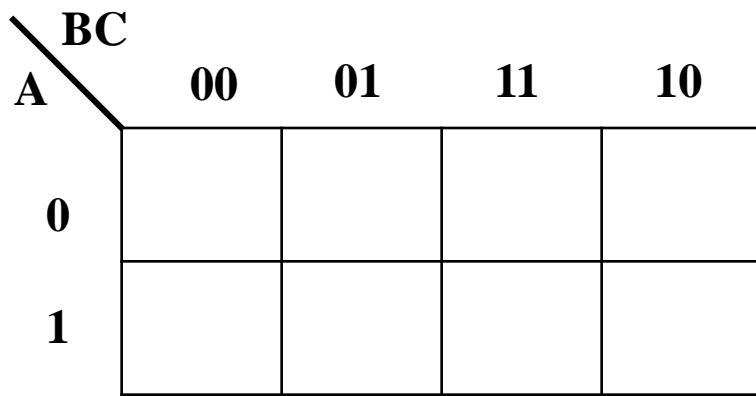


# Karnaugh Map (K-map)

---

## ➤ K-map Structure - 3 Variable

- ✓ A, B & C are variables or inputs
- ✓ 3 variable k-map consists of 8 boxes i.e.  $2^3=8$



# Karnaugh Map (K-map)

---

- ✓ 3 Variable K-map & its associated product terms

		BC	00	01	11	10
		A	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$\bar{A}BC$	$\bar{A}B\bar{C}$
		0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$\bar{A}BC$	$\bar{A}B\bar{C}$
		1	$A\bar{B}\bar{C}$	$A\bar{B}C$	$ABC$	$A\bar{B}\bar{C}$

# Karnaugh Map (K-map)

✓ 3 Variable K-map & its associated minterms

AB		00	01	11	10	
C		0	$m_0$	$m_2$	$m_6$	$m_4$
		1	$m_1$	$m_3$	$m_7$	$m_5$

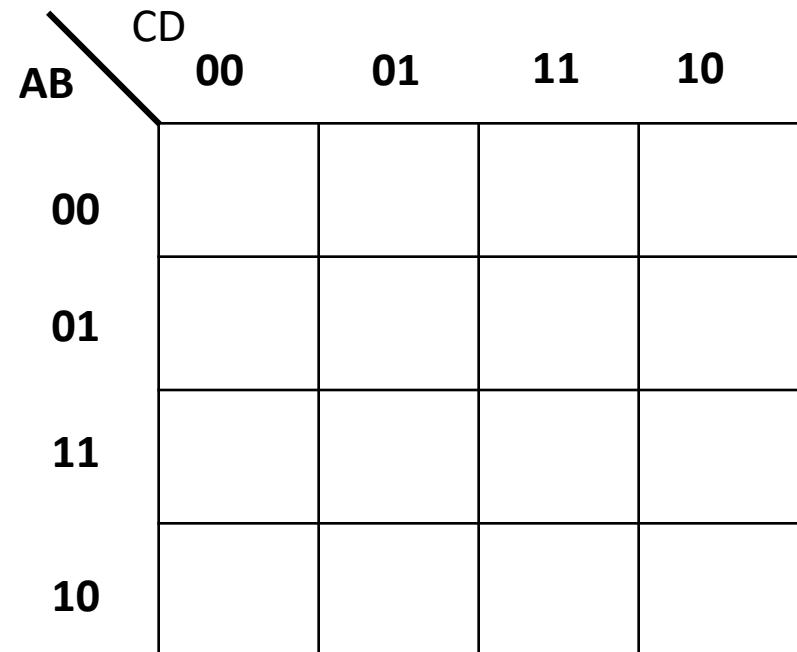
BC		00	01	11	10	
A		0	$m_0$	$m_1$	$m_3$	$m_2$
		1	$m_4$	$m_5$	$m_7$	$m_6$

BC		0	1	
A		00	$m_0$	$m_4$
		01	$m_1$	$m_5$
		11	$m_3$	$m_7$
		10	$m_2$	$m_6$

# Karnaugh Map (K-map)

## ➤ K-map Structure - 4 Variable

- ✓ A, B, C & D are variables or inputs
- ✓ 4 variable k-map consists of 16 boxes i.e.  $2^4=16$



# Karnaugh Map (K-map)

✓ 4 Variable K-map and its associated product terms

		AB	CD			
		00	01	11	10	
		00	$\overline{AB}\overline{CD}$	$A\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$
		01	$\overline{AB}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}D$	$A\overline{B}\overline{C}D$
		11	$\overline{A}\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$AB\overline{C}\overline{D}$	$A\overline{B}\overline{C}D$
		10	$\overline{A}\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$AB\overline{C}\overline{D}$	$A\overline{B}\overline{C}D$

		CD	AB			
		00	01	11	10	
		00	$\overline{AB}\overline{CD}$	$\overline{AB}\overline{CD}$	$A\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$
		01	$\overline{AB}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}D$	$A\overline{B}\overline{C}D$
		11	$\overline{A}\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$AB\overline{C}\overline{D}$	$A\overline{B}\overline{C}D$
		10	$\overline{A}\overline{B}\overline{C}\overline{D}$	$A\overline{B}\overline{C}\overline{D}$	$AB\overline{C}\overline{D}$	$A\overline{B}\overline{C}D$

# Karnaugh Map (K-map)

✓ 4 Variable K-map and its associated minterms

		AB	00	01	11	10
		CD	00	01	11	10
00	00	$m_0$	$m_4$	$m_{12}$	$m_8$	
	01	$m_1$	$m_5$	$m_{13}$	$m_9$	
	11	$m_3$	$m_7$	$m_{15}$	$m_{11}$	
	10	$m_2$	$m_6$	$m_{11}$	$m_{10}$	

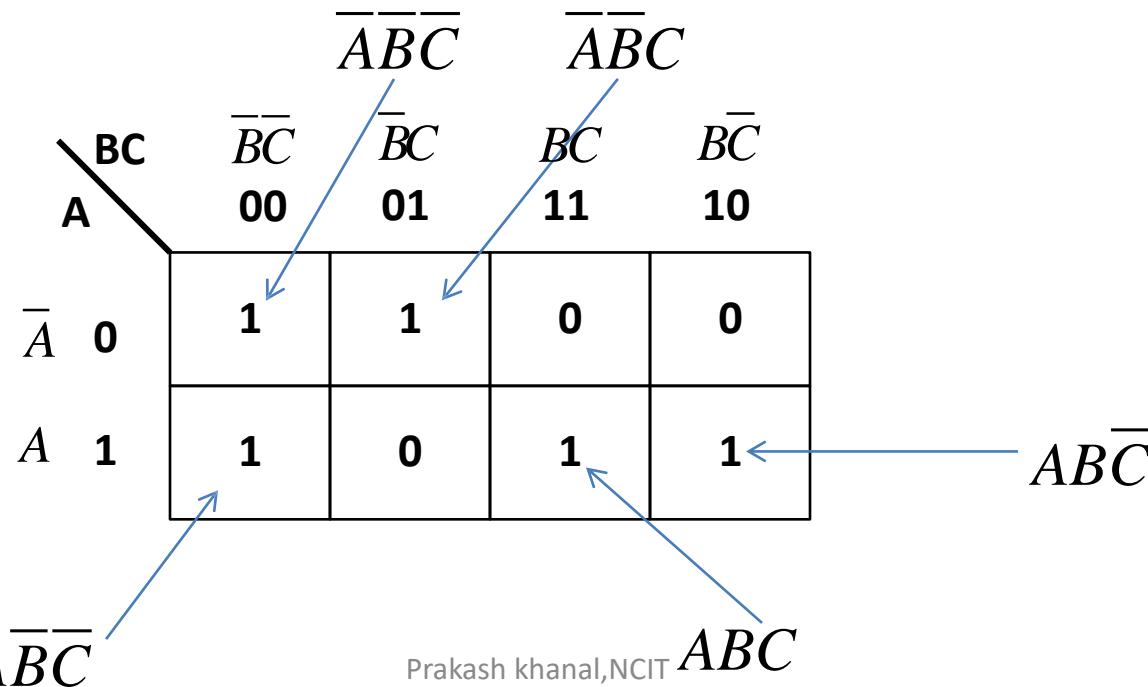
		CD	00	01	11	10
		AB	00	01	11	10
00	00	$m_0$	$m_1$	$m_3$	$m_2$	
	01	$m_4$	$m_5$	$m_7$	$m_6$	
	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$	
	10	$m_8$	$m_9$	$m_{11}$	$m_{10}$	

# Representation of Standard SOP form expression on K-map

For example, SOP equation is given as

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

- ✓ The given expression is in the standard SOP form.
- ✓ Each term represents a minterm.
- ✓ We have to enter '1' in the boxes corresponding to each minterm as below



## Simplification of K-map

---

- ✓ Once we plot the logic function or truth table on K-map, we have to use the grouping technique for simplifying the logic function.
- ✓ Grouping means the combining the terms in adjacent cells.
- ✓ The grouping of either 1's or 0's results in the simplification of Boolean expression.

## Simplification of K-map

---

- ✓ If we group the adjacent 1's then the result of simplification is SOP form
- ✓ If we group the adjacent 0's then the result of simplification is POS form

## Grouping:

---

- ✓ While grouping, we should group most number of 1's.
- ✓ The grouping follows the binary rule i.e we can group 1,2,4,8,16,32,.....number of 1's.
- ✓ We cannot group 3,5,7,.....number of 1's
- ✓ *Pair*: A group of two adjacent 1's is called as Pair
- ✓ *Quad*: A group of four adjacent 1's is called as Quad
- ✓ *Octet*: A group of eight adjacent 1's is called as Octet

# Grouping of Two Adjacent 1's : Pair

✓ A pair eliminates 1 variable

		$\bar{A}BC$		$\bar{A}\bar{B}\bar{C}$	
		$\bar{B}C$	$\bar{B}C$	$BC$	$B\bar{C}$
		00	01	11	10
A	0	0	0	1	1
	1	0	0	0	0

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

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

$$Y = \bar{A}B$$

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

# Grouping of Two Adjacent 1's : Pair

$\bar{A}$	$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	$BC$
$A$	0	0	0	0
$\bar{A}$	1	1	0	1
$A$				

$\bar{A}$	$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	$BC$
$A$	0	0	1	1
$\bar{A}$	1	0	0	1
$A$				

$\bar{A}$	$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	$BC$
$A$	0	0	1	0
$\bar{A}$	1	0	1	0
$A$				

$\bar{A}$	$\bar{B}$	$B$
$A$	0	1
$\bar{A}$	0	1
$A$	1	0

# Possible Grouping of Four Adjacent 1's : Quad

✓ A Quad eliminates 2 variable

		CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
		AB	00	01	11	10
AB	00	0	0	0	0	0
	01	0	0	0	0	0
	11	0	0	0	0	0
	10	1	1	1	1	1

		CD	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
		AB	00	01	11	10
AB	00	0	1	0	0	0
	01	0	1	0	0	0
	11	0	1	0	0	0
	10	0	1	0	0	0

# Possible Grouping of Four Adjacent 1's : Quad

✓ A Quad eliminates 2 variable

$\bar{AB}$	$CD$	$\bar{CD}$	$\bar{C}D$	$C\bar{D}$	$CD$	$\bar{C}\bar{D}$
$AB$	$00$	$01$	$11$	$10$		
$\bar{A}\bar{B}$	0	0	0	0		
$\bar{A}\bar{B}$	1	1	0	0		
$A\bar{B}$	1	1	0	0		
$A\bar{B}$	0	0	0	0		

$\bar{AB}$	$CD$	$\bar{CD}$	$\bar{C}D$	$C\bar{D}$	$CD$	$\bar{C}\bar{D}$
$AB$	$00$	$01$	$11$	$10$		
$\bar{A}\bar{B}$	0	1	1	0		
$\bar{A}\bar{B}$	0	0	0	0		
$A\bar{B}$	0	0	0	0		
$A\bar{B}$	0	1	1	0		

# Possible Grouping of Four Adjacent 1's : Quad

✓ A Quad eliminates 2 variable

$\bar{A}\bar{B}$	$CD$	$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
$\bar{A}B$	00	00	01	11	10
$A\bar{B}$	00	1	0	0	1
$\bar{A}B$	01	0	0	0	0
$A\bar{B}$	11	0	0	0	0
$A\bar{B}$	10	1	0	0	1

$\bar{A}\bar{B}$	$CD$	$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
$\bar{A}B$	00	00	01	11	10
$A\bar{B}$	00	0	0	0	0
$\bar{A}B$	01	1	0	0	1
$A\bar{B}$	11	1	0	0	1
$A\bar{B}$	10	0	0	0	0

# Possible Grouping of Four Adjacent 1's : Quad

✓ A Quad eliminates 2 variable

$\bar{A}\bar{B}$	$CD$	$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
$A\bar{B}$	$AB$	$00$	$01$	$11$	$10$
$\bar{A}B$	$\bar{A}\bar{B}$	00	01	11	10
00	0	0	0	0	0
01	0	1	1	1	1
11	0	1	1	1	1
10	0	0	0	0	0

$\bar{A}\bar{B}$	$CD$	$\bar{C}\bar{D}$	$\bar{C}D$	$CD$	$C\bar{D}$
$A\bar{B}$	$AB$	$00$	$01$	$11$	$10$
$\bar{A}B$	$\bar{A}\bar{B}$	00	01	11	10
00	0	0	0	0	0
01	0	1	1	1	0
11	0	1	1	1	0
10	0	1	1	1	0

# Possible Grouping of Eight Adjacent 1's : Octet

✓ A Octet eliminates 3 variable

$\bar{AB}$	$\bar{CD}$	$\bar{CD}$	$\bar{CD}$	$CD$	$CD$
$\bar{AB}$	00	01	11	11	10
$AB$	0	0	0	0	0
$\bar{AB}$	0	0	0	0	0
$AB$	1	1	1	1	1
$\bar{AB}$	1	1	1	1	1

$\bar{AB}$	$\bar{CD}$	$\bar{CD}$	$\bar{CD}$	$CD$	$CD$
$\bar{AB}$	00	01	11	11	10
$AB$	0	1	1	0	0
$\bar{AB}$	0	1	1	0	0
$AB$	0	1	1	0	0
$\bar{AB}$	0	1	1	0	0

# Possible Grouping of Eight Adjacent 1's : Octet

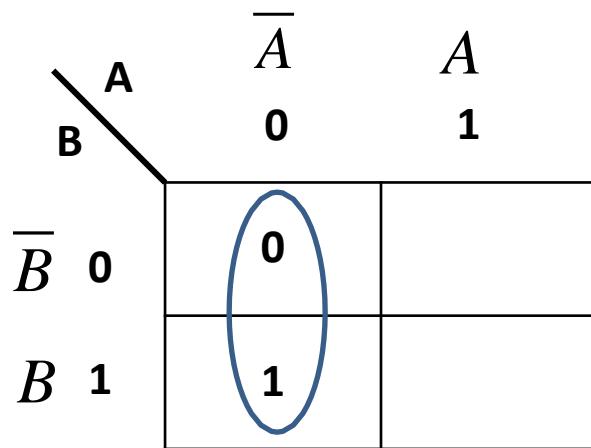
✓ A Octet eliminates 3 variable

$\bar{AB}$	$CD$	$\bar{CD}$	$\bar{CD}$	$CD$	$CD$	$\bar{CD}$
$\bar{AB}$	00	00	01	11	11	10
$\bar{AB}$	00	1	1	1	1	1
$\bar{AB}$	01	0	0	0	0	0
$\bar{AB}$	11	0	0	0	0	0
$\bar{AB}$	10	1	1	1	1	1

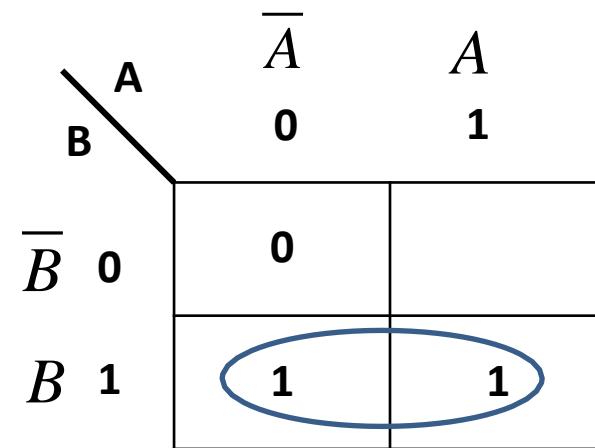
$\bar{AB}$	$CD$	$\bar{CD}$	$\bar{CD}$	$CD$	$CD$	$\bar{CD}$
$\bar{AB}$	00	00	01	11	11	10
$\bar{AB}$	00	1	0	0	1	
$\bar{AB}$	01	1	0	0	1	
$\bar{AB}$	11	1	0	0	1	
$\bar{AB}$	10	1	0	0	1	

# Rules for K-map simplification

- Groups may not include any cell containing a zero.***



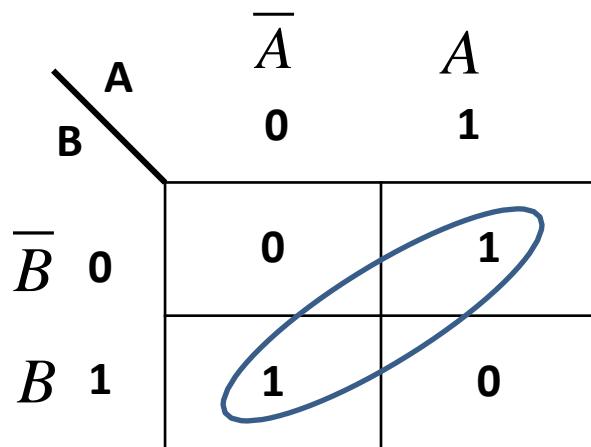
Not Accepted



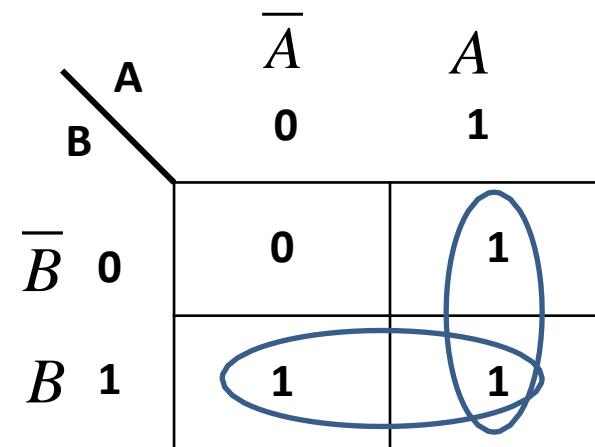
Accepted

# Rules for K-map simplification

**2. Groups may be horizontal or vertical, but may not be diagonal**



Not Accepted



Accepted

# Rules for K-map simplification

---

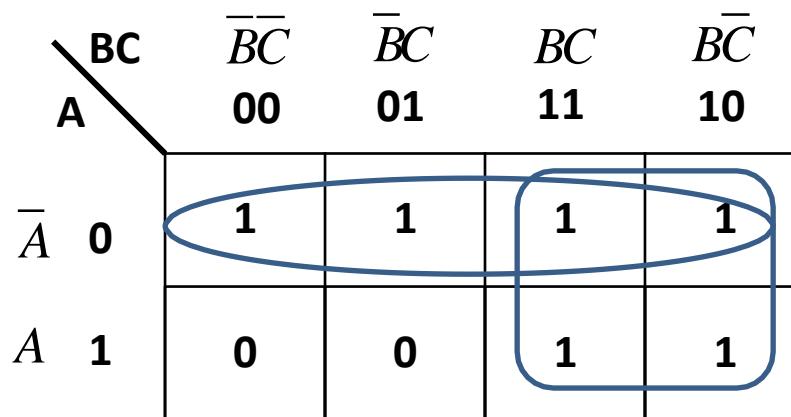
**5. Each cell containing a one must be in at least one group**

		$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	$BC$
		00	01	11	10
$A$	$\bar{A}$	0	0	0	1
	$A$	0	0	1	0

# Rules for K-map simplification

---

## 6. Groups may be overlap



## Rules for K-map simplification

**7. Groups may wrap around the table. The leftmost cell in a row may be grouped with rightmost cell and the top cell in a column may be grouped with bottom cell**

		CD	$\bar{CD}$	$\bar{CD}$	CD	CD
		00	01	11	11	10
AB	$\bar{AB}$	1	1	1	1	
	$\bar{AB}$	0	0	0	0	
$A\bar{B}$	$\bar{AB}$	0	0	0	0	
	$\bar{AB}$	1	1	1	1	

		BC	$\bar{BC}$	$\bar{BC}$	BC	$\bar{BC}$
		00	01	11	11	10
A	$\bar{A}$	1				
	A	1				
$\bar{A}$	$\bar{A}$		0	0		1
	A		1	0	0	1

# Rules for K-map simplification

---

**9. A pair eliminates one variable.**

**10. A Quad eliminates two variables.**

**11. A octet eliminates three variables**

# Example 1

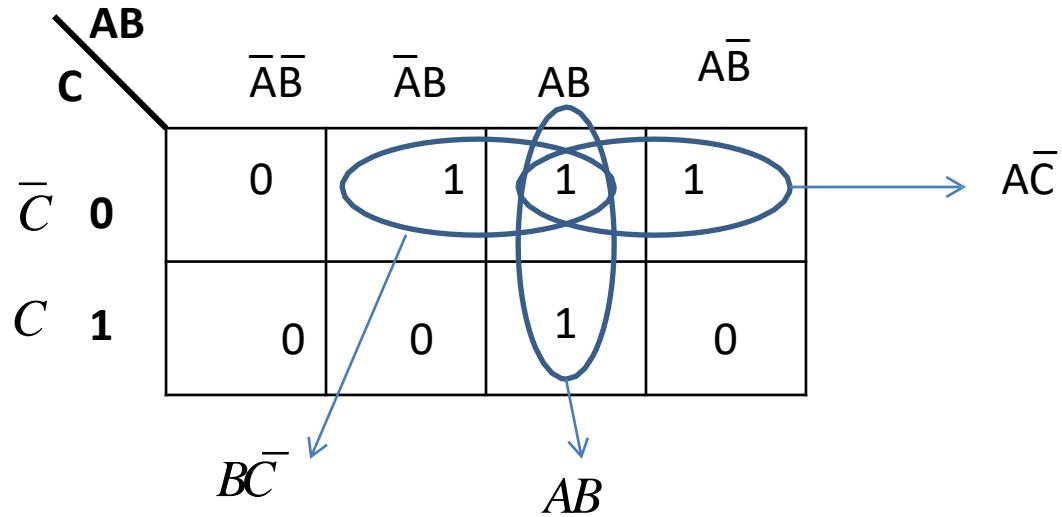
---

*For the given K-map write simplified Boolean expression*

		$\bar{A}\bar{B}$	$\bar{A}B$	$AB$	$A\bar{B}$
		00	01	11	10
$C$	0	0	1	1	1
	1	0	0	1	0

## Example 1

continue.....



**Simplified Boolean expression**

$$Y = BC\bar{C} + AB + \bar{A}C\bar{C}$$

## Example 2

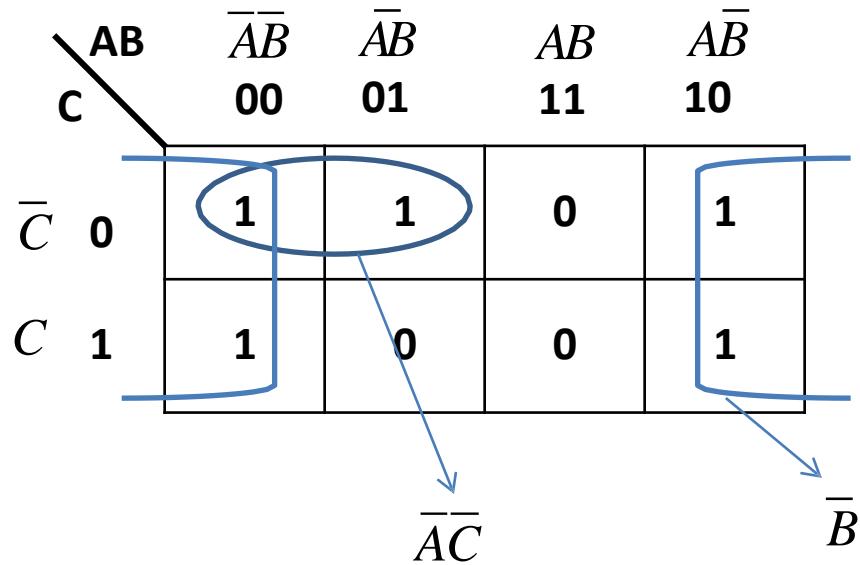
---

*For the given K-map write simplified Boolean expression*

		$\bar{A}\bar{B}$	$\bar{A}B$	$AB$	$A\bar{B}$
		00	01	11	10
$C$	0	1	1	0	1
	1	1	0	0	1

## Example 2

continue.....



**Simplified Boolean expression**

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

## Example 3

---

A logical expression in the standard SOP form  
is as follows;

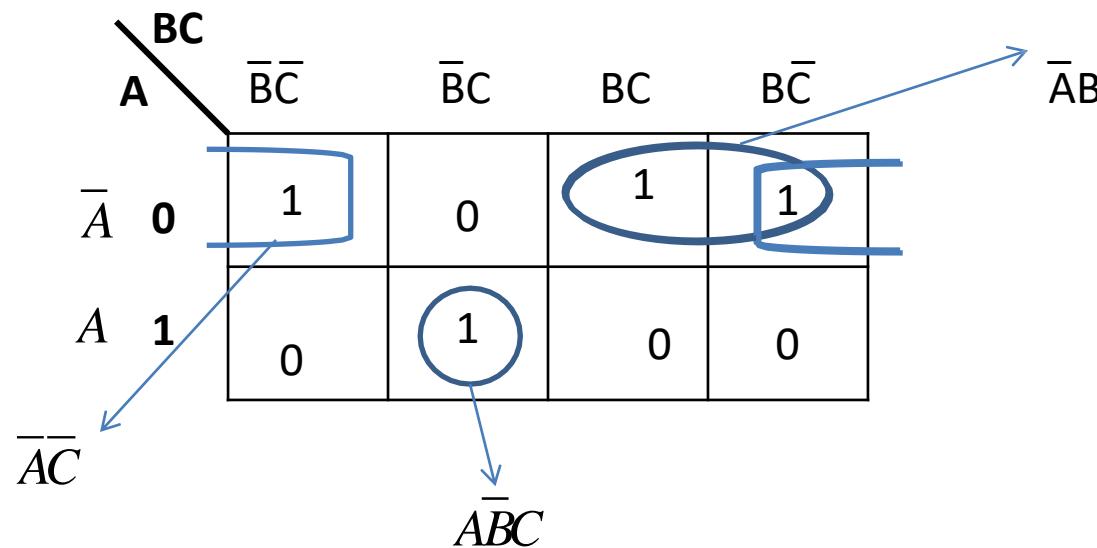
$$Y = \overline{A} \overline{B} \overline{C} + \overline{A} B \overline{C} + \overline{A} B C + A \overline{B} C$$

Minimize it with using the K-map technique

## Example 3

continue.....

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



Simplified Boolean expression

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

## Example 4

---

A logical expression representing a logic circuit is;

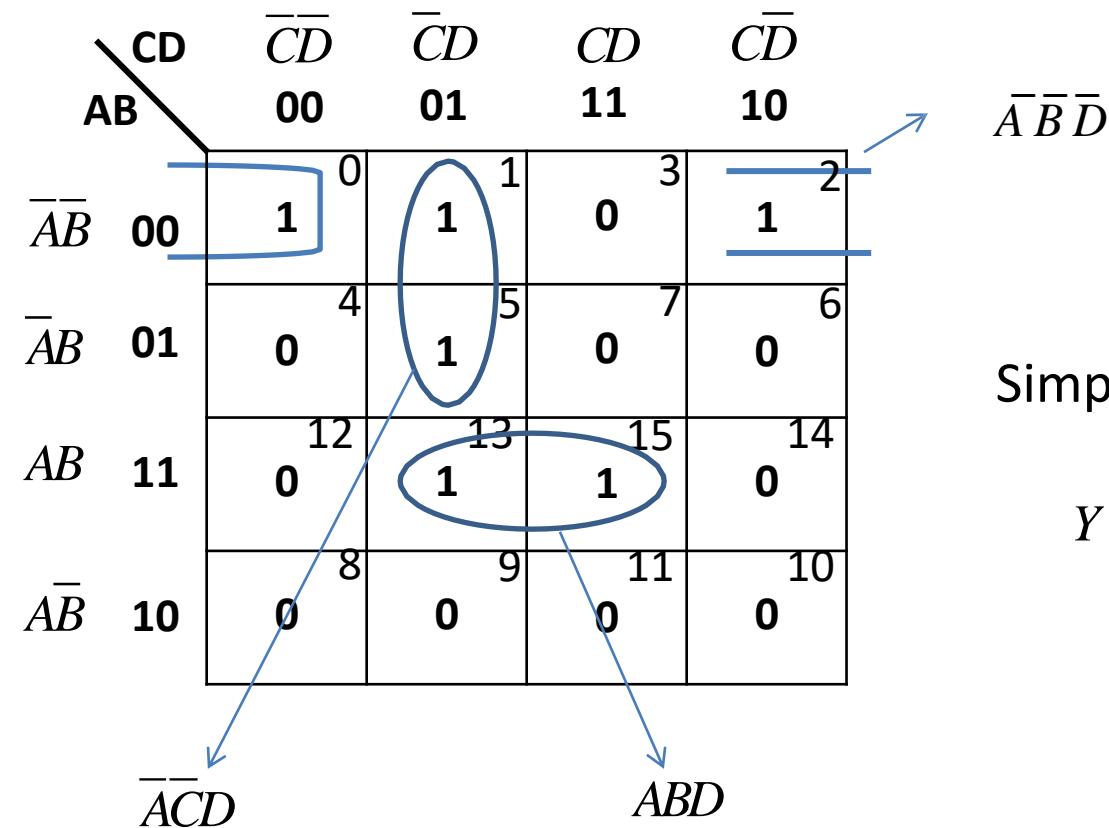
$$Y = \Sigma m (0, 1, 2, 5, 13, 15)$$

Draw the K-map and find the minimized logical expression

## Example 4

continue.....

$$Y = \Sigma m(0, 1, 2, 5, 13, 15)$$



Simplified Boolean expression

$$Y = \bar{A}\bar{B}\bar{D} + \bar{A}\bar{C}D + ABD$$

## Example 5

---

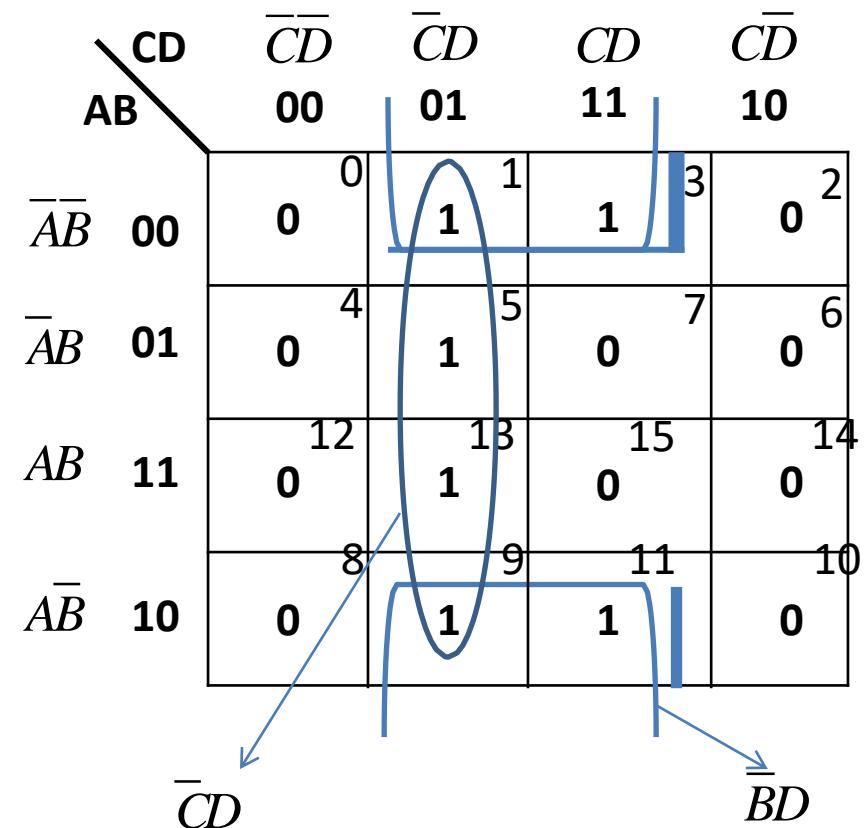
Minimize the following Boolean expression  
using K-map ;

$$f(A, B, C, D) = \Sigma m(1, 3, 5, 9, 11, 13)$$

## Example 5

continue.....

$$f(A, B, C, D) = \sum m(1, 3, 5, 9, 11, 13)$$



Simplified Boolean expression

$$f = \bar{B}D + \bar{C}D$$

$$f = D(\bar{B} + \bar{C})$$

## Example 6

---

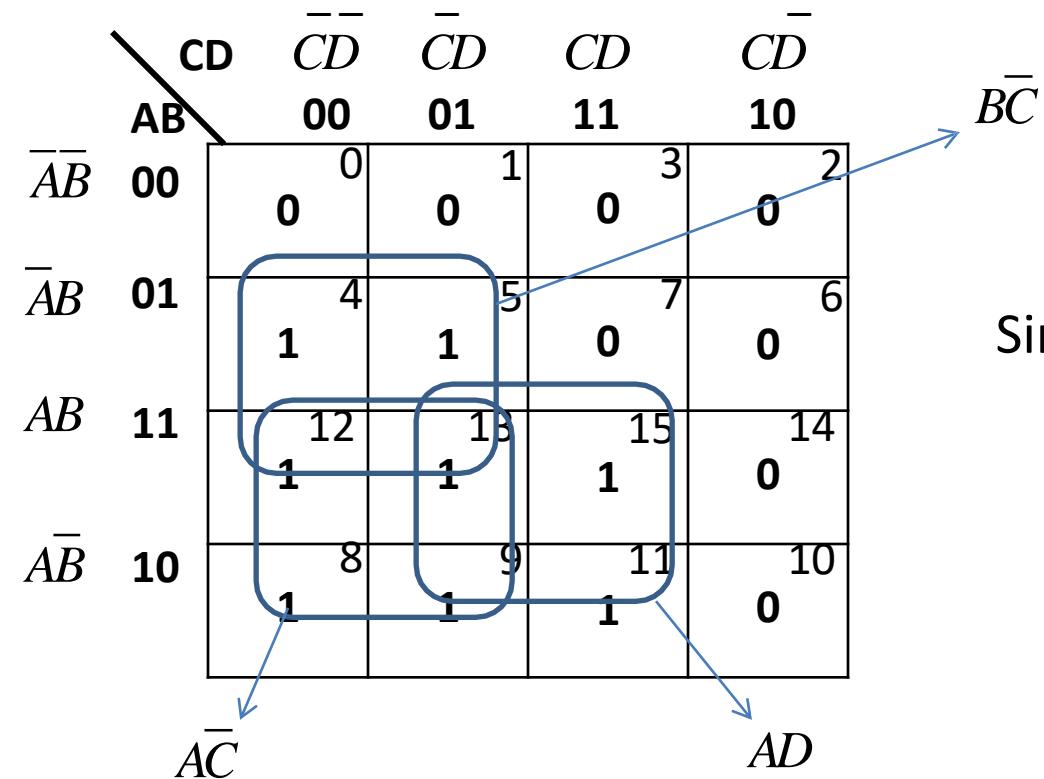
Minimize the following Boolean expression  
using K-map ;

$$f(A, B, C, D) = \Sigma m(4, 5, 8, 9, 11, 12, 13, 15)$$

## Example 6

continue.....

$$f(A, B, C, D) = \sum m(4, 5, 8, 9, 11, 12, 13, 15)$$



Simplified Boolean expression

$$f = B\bar{C} + A\bar{C} + AD$$

## Example 7

---

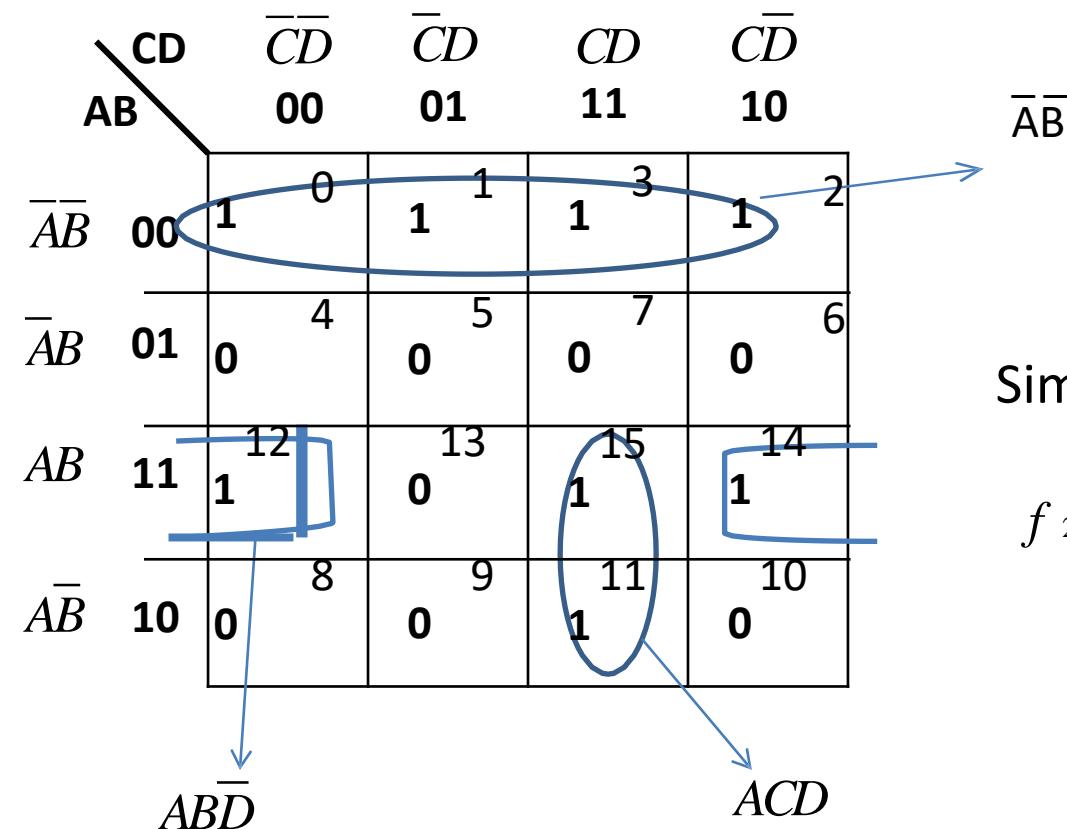
Minimize the following Boolean expression  
using K-map ;

$$f_2(A, B, C, D) = \Sigma m(0, 1, 2, 3, 11, 12, 14, 15)$$

## Example 7

continue....

$$f_2(A, B, C, D) = \sum m(0, 1, 2, 3, 11, 12, 14, 15)$$



Simplified Boolean expression

$$f_2 = \overline{AB} + AB\overline{D} + ACD$$

## Example 8

---

Solve the following expression with K-maps;

$$1 \quad f_1(A, B, C) = \Sigma m(0, 1, 3, 4, 5)$$

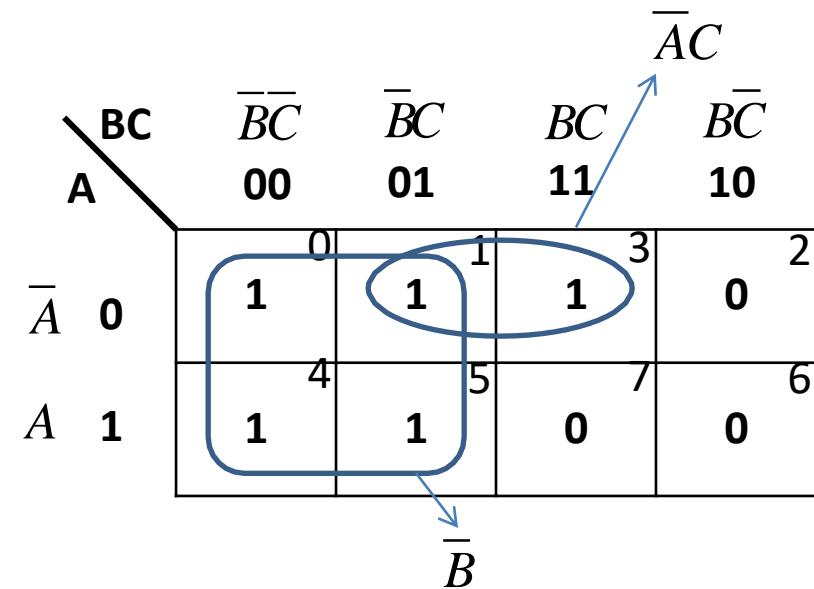
$$2 \quad f_2(A, B, C) = \Sigma m(0, 1, 2, 3, 6, 7)$$

## Example 8

continue.....

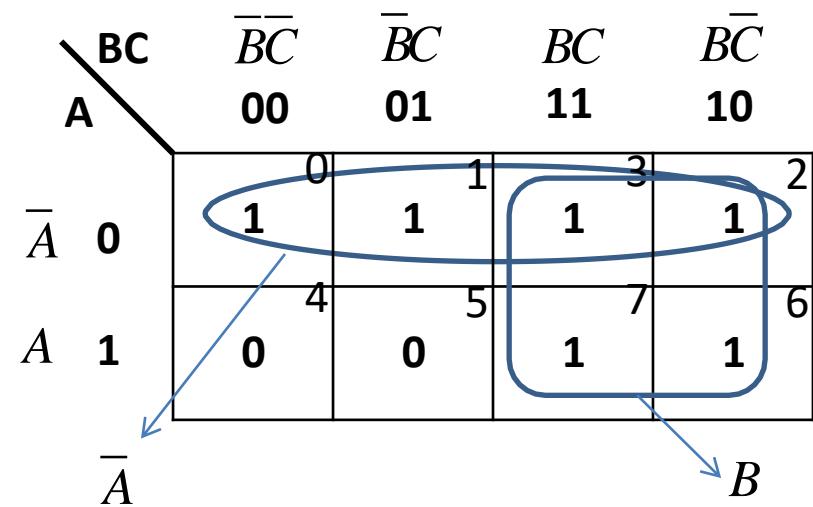
$$f_1(A, B, C) = \Sigma m(0, 1, 3, 4, 5)$$

$$f_2(A, B, C) = \Sigma m(0, 1, 2, 3, 6, 7)$$



Simplified Boolean expression

$$f_1 = \bar{A}C + \bar{B}$$



Simplified Boolean expression

$$f_2 = \bar{A} + B$$

## Example 9

---

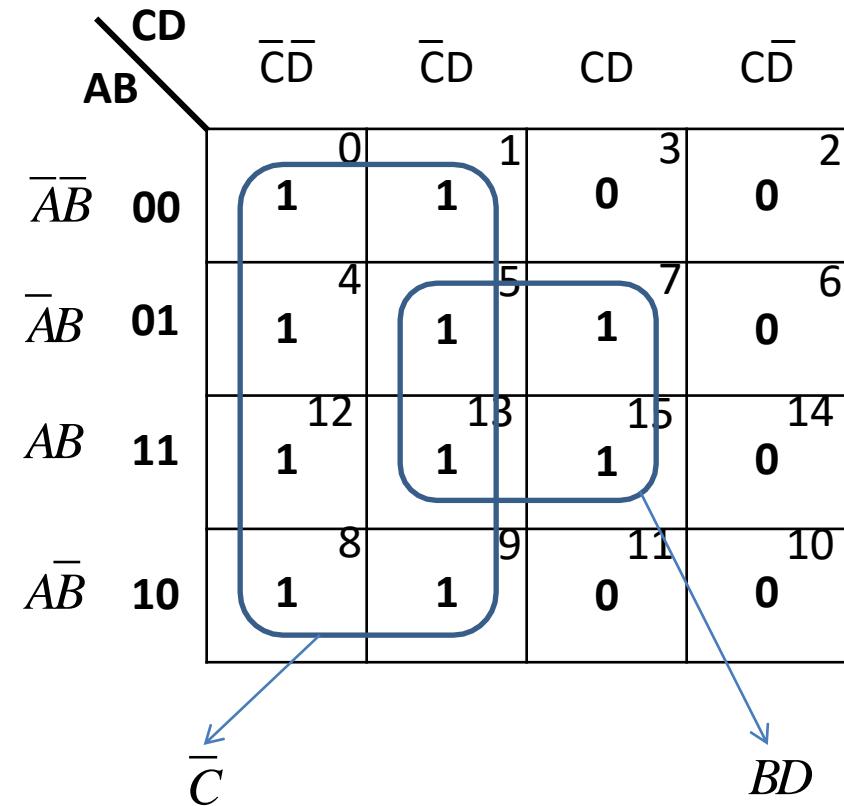
Simplify ;

$$f(A, B, C, D) = \Sigma m(0, 1, 4, 5, 7, 8, 9, 12, 13, 15)$$

## Example 9

continue....

$$f(A, B, C, D) = \sum m(0, 1, 4, 5, 7, 8, 9, 12, 13, 15)$$



Simplified Boolean expression

$$f = \overline{C} + BD$$

## Example 10

---

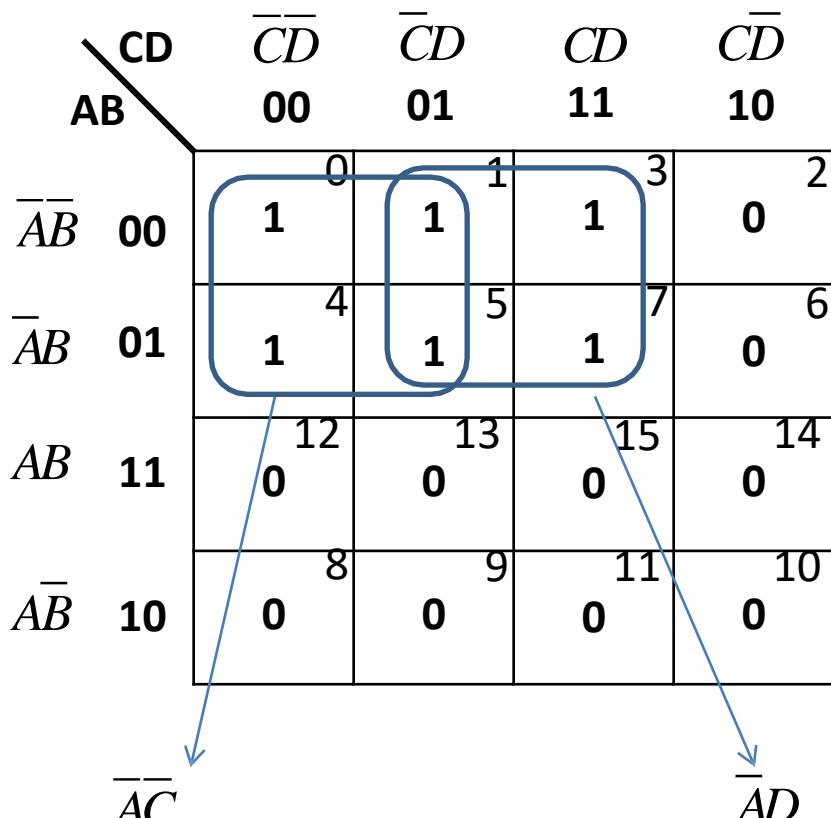
Solve the following expression with K-maps;

- 1       $f_1(A, B, C, D) = \Sigma m(0, 1, 3, 4, 5, 7)$
- 2       $f_2(A, B, C) = \Sigma m(0, 1, 3, 4, 5, 7)$

## Example 10

continue.....

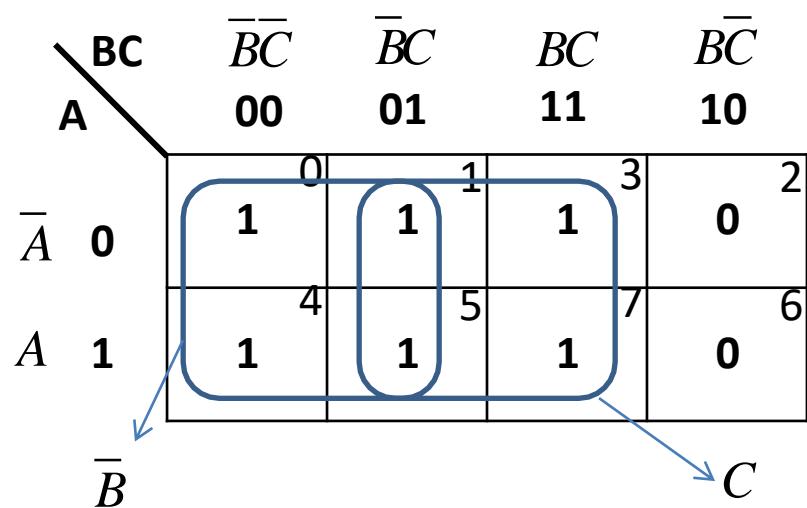
$$f_1(A, B, C, D) = \Sigma m(0, 1, 3, 4, 5, 7)$$



Simplified Boolean expression

$$f_1 = \overline{A}\overline{C} + \overline{A}D$$

$$f_2(A, B, C) = \Sigma m(0, 1, 3, 4, 5, 7)$$



Simplified Boolean expression

$$f_2 = \overline{B} + C$$

11/8/2020

Prakash khanal,NCIT

## K-map for Product of Sum Form (POS Expressions)

---

- ✓ Karnaugh map can also be used for Boolean expression in the Product of sum form (POS).
- ✓ The procedure for simplification of expression by grouping of cells is also similar

# K-map for Product of Sum Form (POS Expressions)

---

- ✓ The letters with bars (NOT) represent 1 and unbarred letters represent 0 of Binary.
- ✓ A zero is put in the cell for which there is a term in the Boolean expression
- ✓ Grouping is done for adjacent cells containing zeros.

## Example 11

---

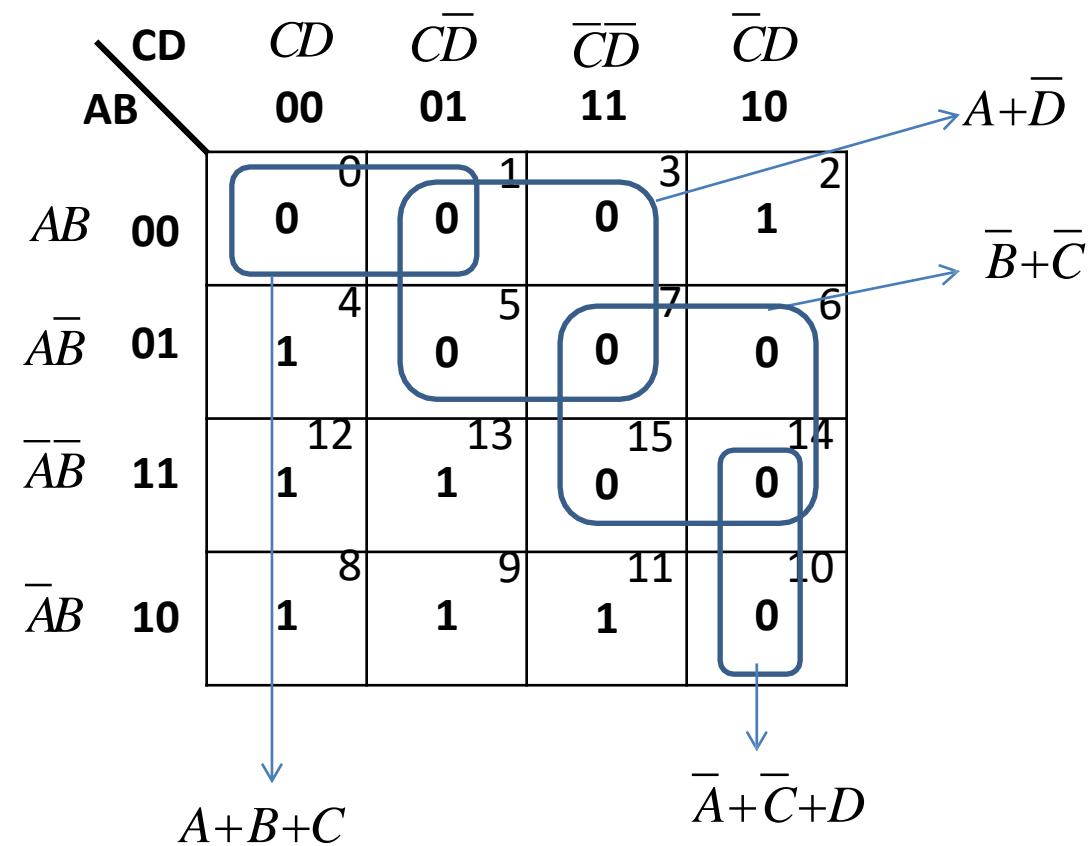
Simplify ;

$$f(A, B, C, D) = \prod M(0, 1, 3, 5, 6, 7, 10, 14, 15)$$

## Example 11

continue.....

$$f(A, B, C, D) = \prod M(0, 1, 3, 5, 6, 7, 10, 14, 15)$$



Simplified Boolean expression

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

## Example 12

---

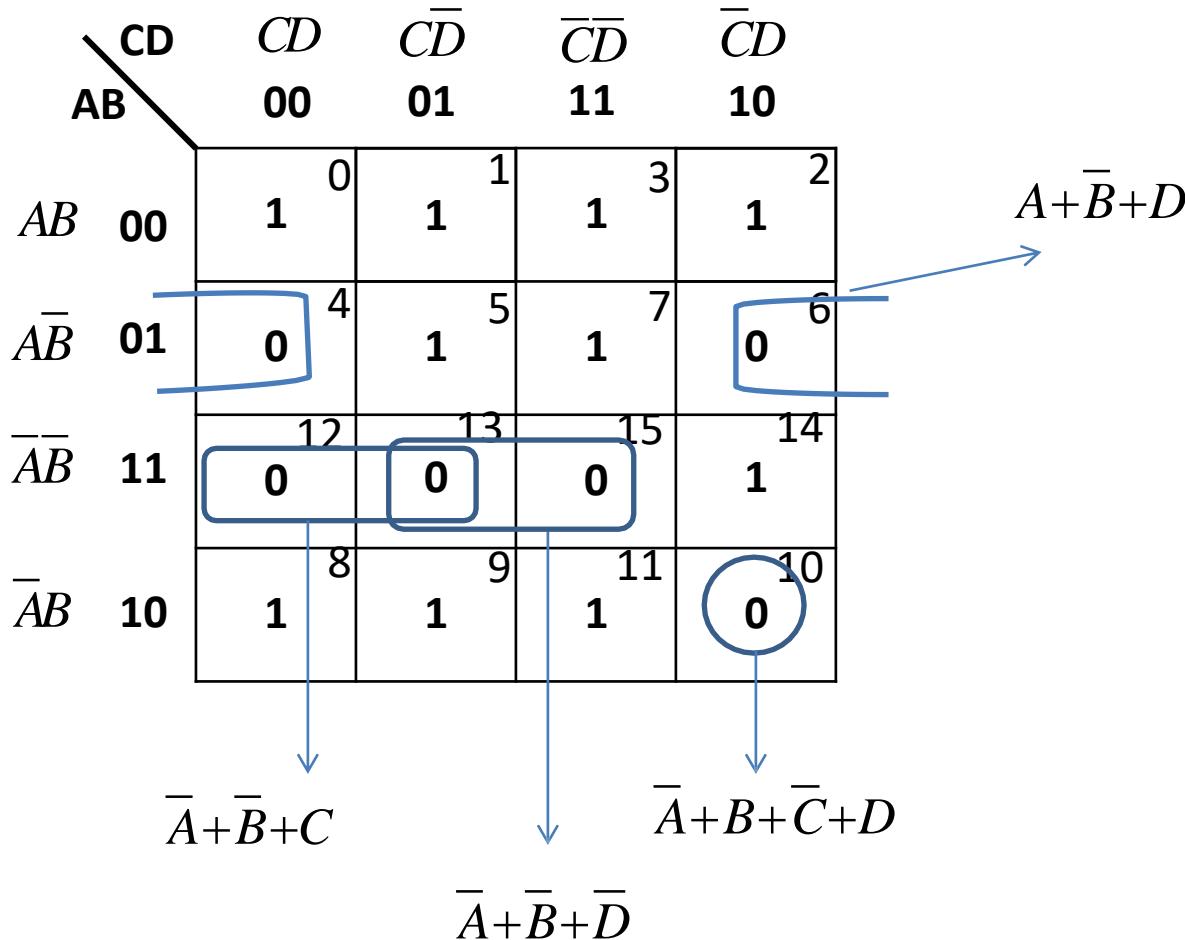
Simplify ;

$$f(A, B, C, D) = \prod M(4, 6, 10, 12, 13, 15)$$

## Example 12

continue....

$$f(A, B, C, D) = \prod M(4, 6, 10, 12, 13, 15)$$



Simplified Boolean expression

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

## K-map and don't care conditions

---

- ✓ For SOP form we enter 1's corresponding to the combinations of input variables which produce a high output and we enter 0's in the remaining cells of the K-map.
- ✓ For POS form we enter 0's corresponding to the combinations of input variables which produce a high output and we enter 1's in the remaining cells of the K-map.

## K-map and don't care conditions

---

- ✓ But it is not always true that the cells not containing 1's (in SOP) will contain 0's, because some combinations of input variable do not occur.
- ✓ Also for some functions the outputs corresponding to certain combinations of input variables do not matter.

## K-map and don't care conditions

---

- ✓ In such situations we have a freedom to assume a 0 or 1 as output for each of these combinations.
- ✓ These conditions are known as the “Don’t Care Conditions” and in the K-map it is represented as ‘X’, in the corresponding cell.
- ✓ The don’t care conditions may be assumed to be 0 or 1 as per the need for simplification

## K-map and don't care conditions - Example

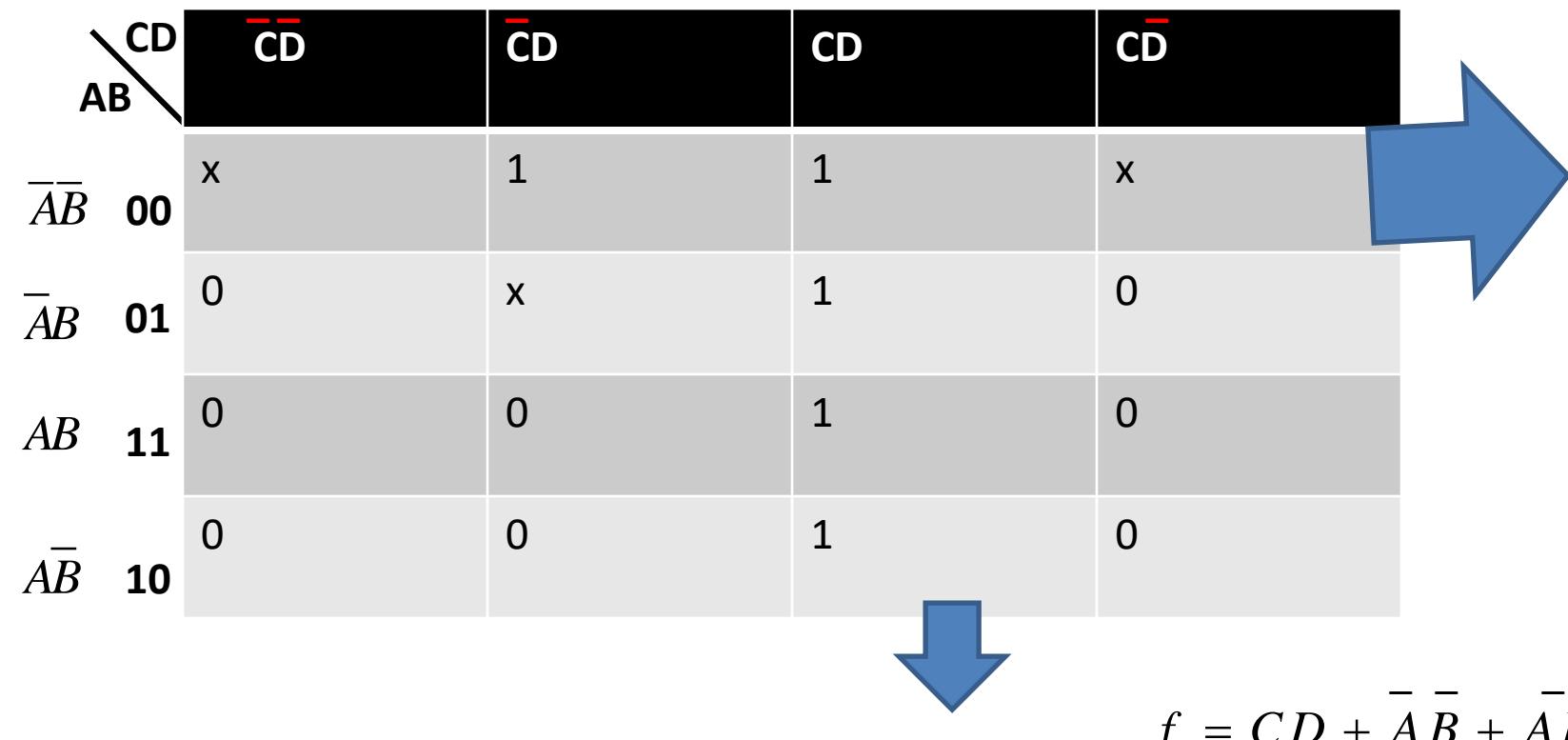
---

Simplify ;

$$f(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + d(0, 2, 5)$$

# K-map and don't care conditions - Example

$$f(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + d(0, 2, 5)$$



# EXAMPLE:

- Simplify the following Boolean function using don't care conditions:

$$F(W,X,Y,Z) = W(\bar{X}\bar{Y} + \bar{X}\bar{Y} + XYZ) + X\bar{Z}(Y+W)$$

$$D(W,X,Y,Z) = \bar{W}\bar{X}(\bar{Y}Z + Y\bar{Z})$$

And implement using an universal gate.

# Five variable k-map:

		CDE	000	001	011	010	110	111	101	100
AB			00	01	11	10	110	111	101	100
00		0	1	3	2	6	7	5	4	
01		8	9	11	10	14	15	13	12	
11		24	25	27	26	30	31	29	28	
10		16	17	19	18	22	23	21	20	

Five-variable k-map

- Simplify the following Boolean function:

$$f(A, B, C, D, E) = \Sigma m(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$$

		CDE							
		000	001	011	010	110	111	101	100
AB	00	1	0	0	1	1	0	0	1
	01	0	1	1	0	0	1	1	0
AB	11	0	1	1	0	0	1	1	0
	10	0	1	0	0	0	0	1	0

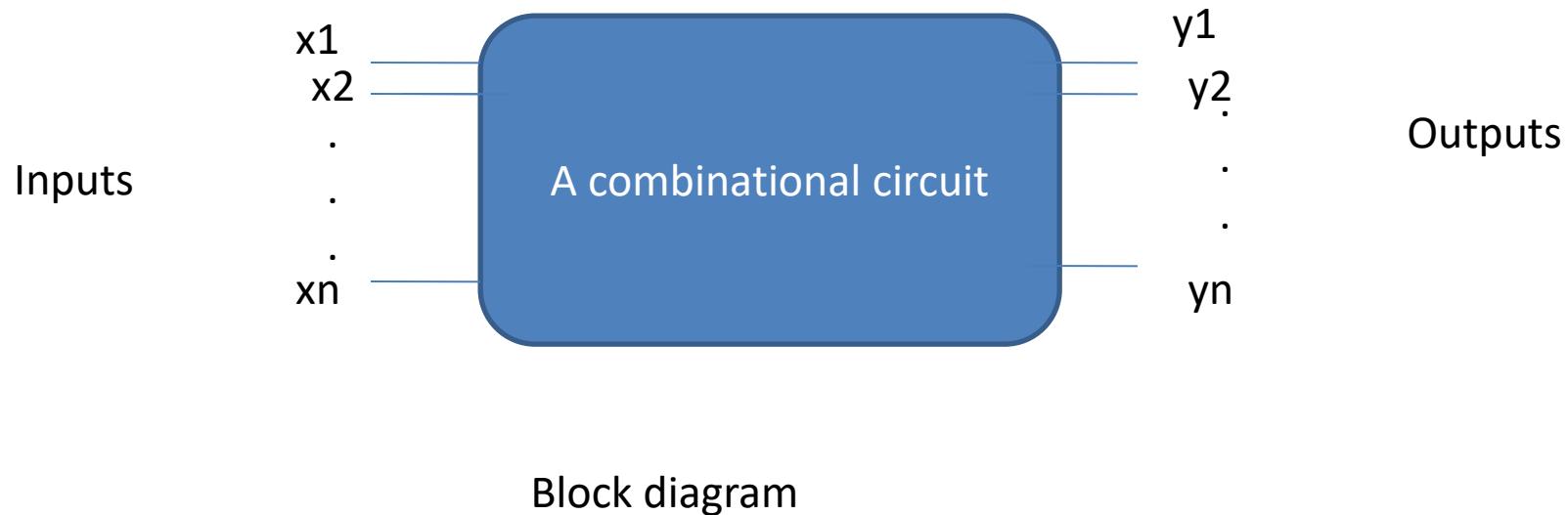
$$F = ABE + ADE + BE$$

# Introduction:

- Logic circuit for digital system may be:
  - Combinational Logic
  - Sequential Logic

# Combinational logic

- Output at any times depends only on present combination of input.
- Combinational circuit does not need any memory.
- Combinational circuits are faster in operation than sequential circuits.



## Design procedure:

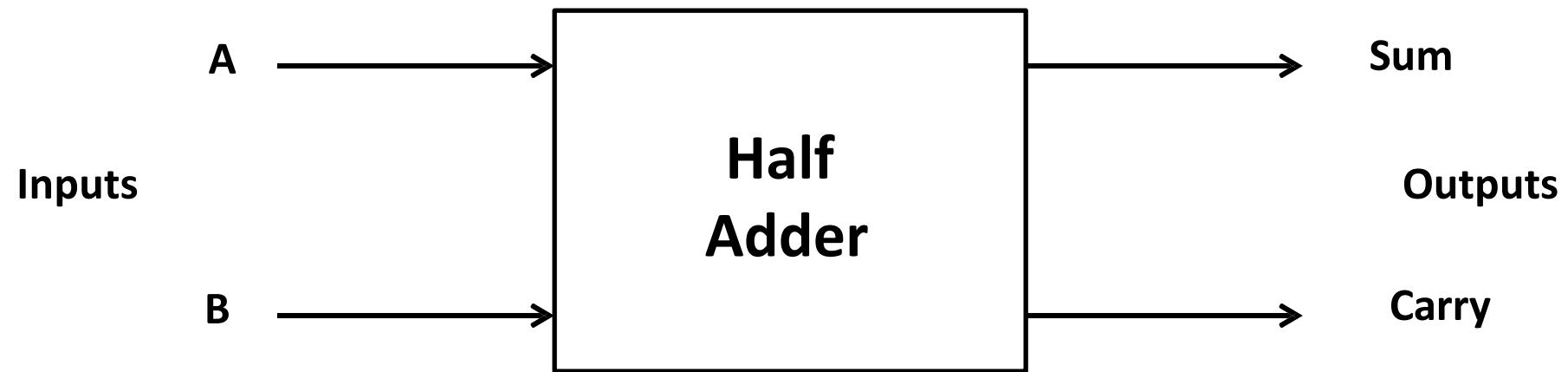
The design steps for combinational logic are:

- Question is given to you.
- Find the number of i/p and o/p.
- Draw the truth table.
- Simplify the o/p using K-map. Simplification is done for o/p only.
- Draw the logic diagram.

# Half Adder

---

- ✓ Half adder is a combinational logic circuit with two inputs and two outputs.
- ✓ It is a basic building block for addition of two single bit numbers.



# Half Adder

---

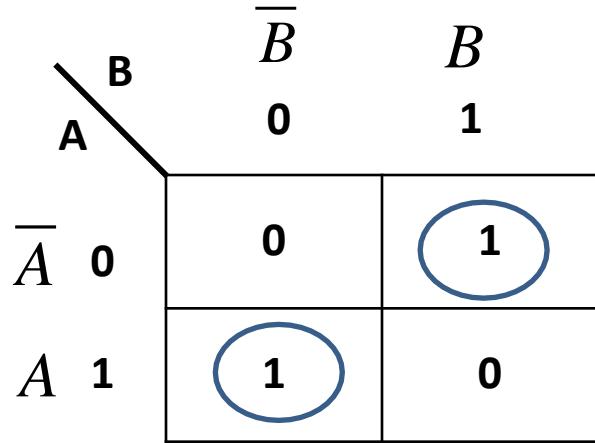
## Truth Table for Half Adder

Input		Output	
A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

# Half Adder

---

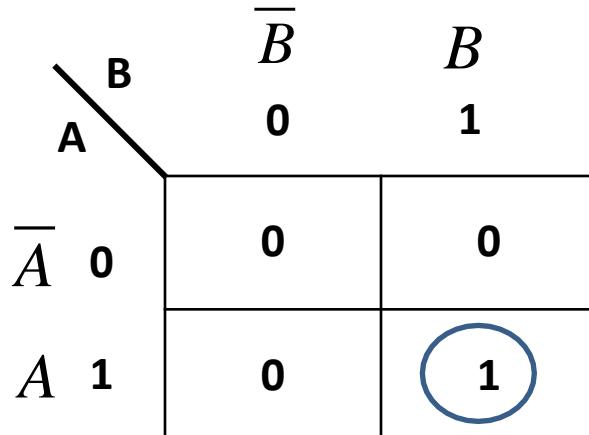
K-map for Sum Output:



$$S = \bar{A}B + A\bar{B}$$

$$S = A \oplus B$$

K-map for Carry Output:

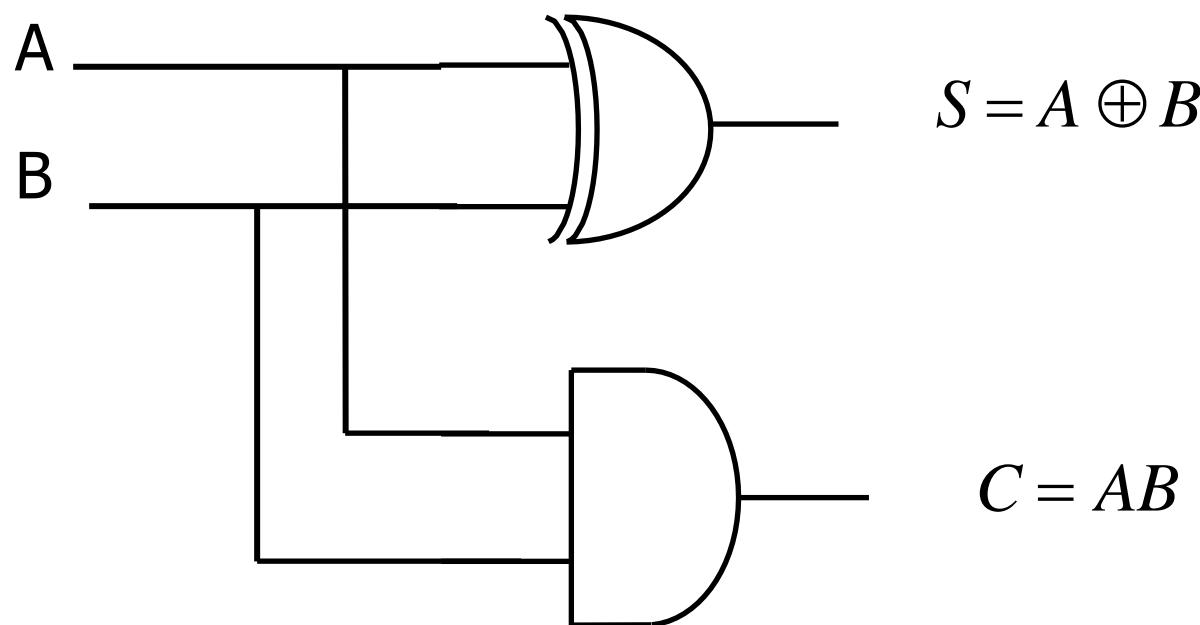


$$C = AB$$

# Half Adder

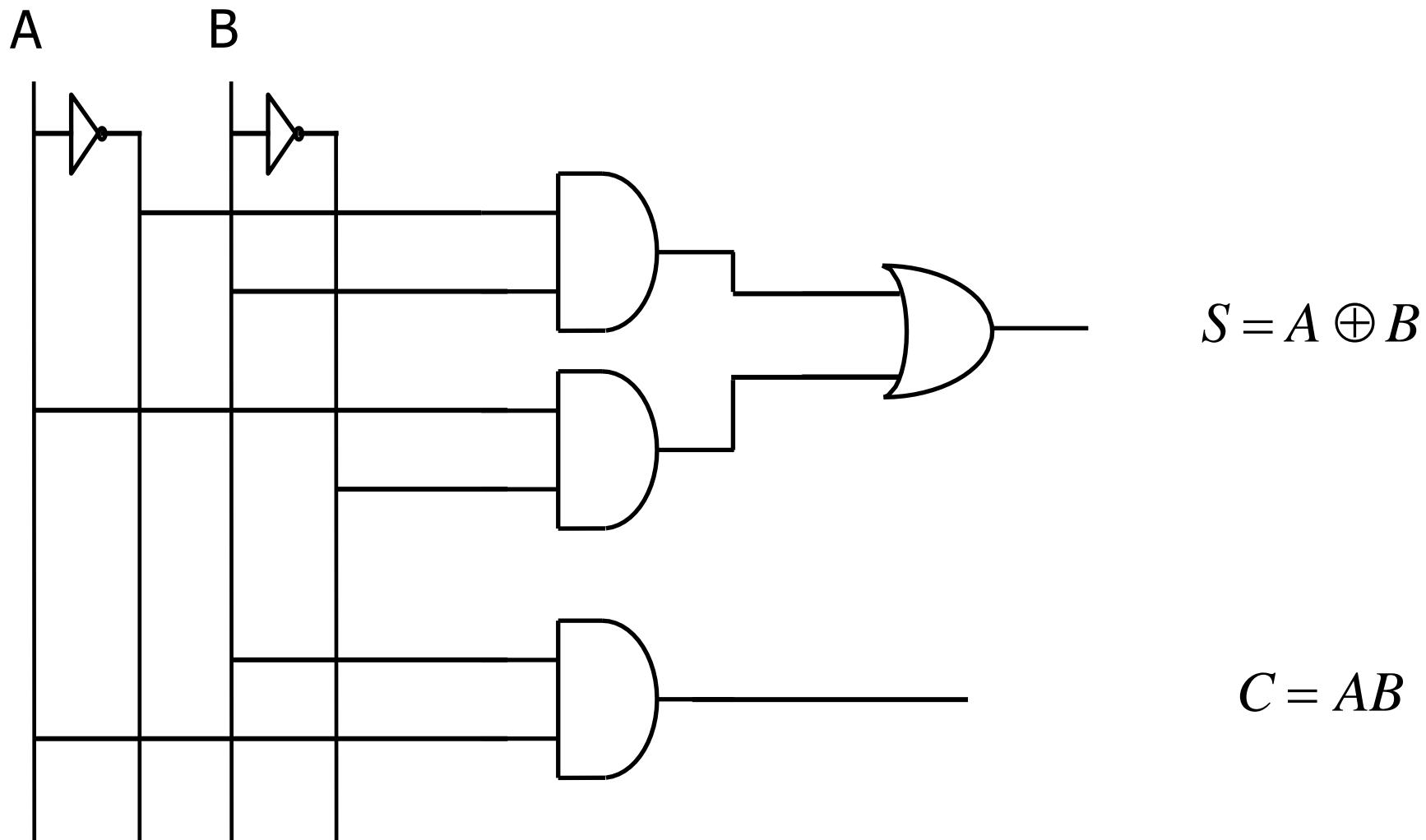
---

Logic Diagram:



# Half Adder

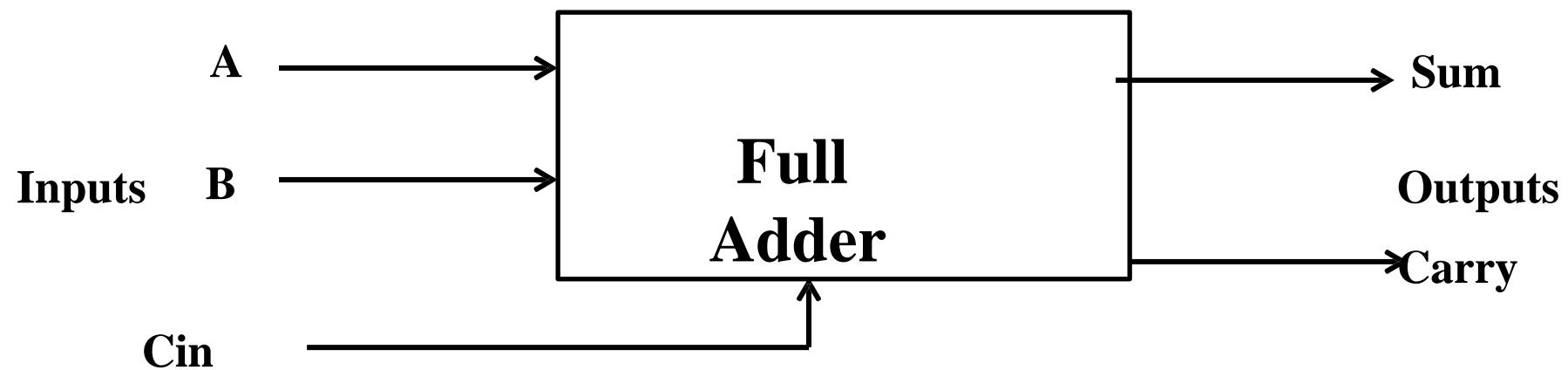
## Logic Diagram using Basic Gates:



# Full Adder

---

- ✓ Full adder is a combinational logic circuit with three inputs and two outputs.



# Full Adder

## Truth Table

Inputs			Outputs	
A	B	Cin	Sum (S)	Carry (C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

# Full Adder

K-map for Sum Output:

		$\bar{B}\bar{C}$	$\bar{B}C$	$B\bar{C}$	$BC$
		00	01	11	10
		0	1	0	1
$\bar{A}$	0	0	1	0	1
	1	1	0	1	0

Arrows point from the circled '1's in the K-map to the output terms  $A\bar{B}\bar{C}$ ,  $\bar{A}\bar{B}C$ ,  $ABC$ , and  $\bar{A}B\bar{C}$  respectively.

$$S = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C + AB\bar{C}$$

$$S = \bar{A}\bar{B}\bar{C} + ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$

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

$$\text{Let } \bar{A}B + A\bar{B} = X$$

$$\therefore S = C(\bar{X}) + \bar{C}(X)$$

$$S = C \oplus X$$

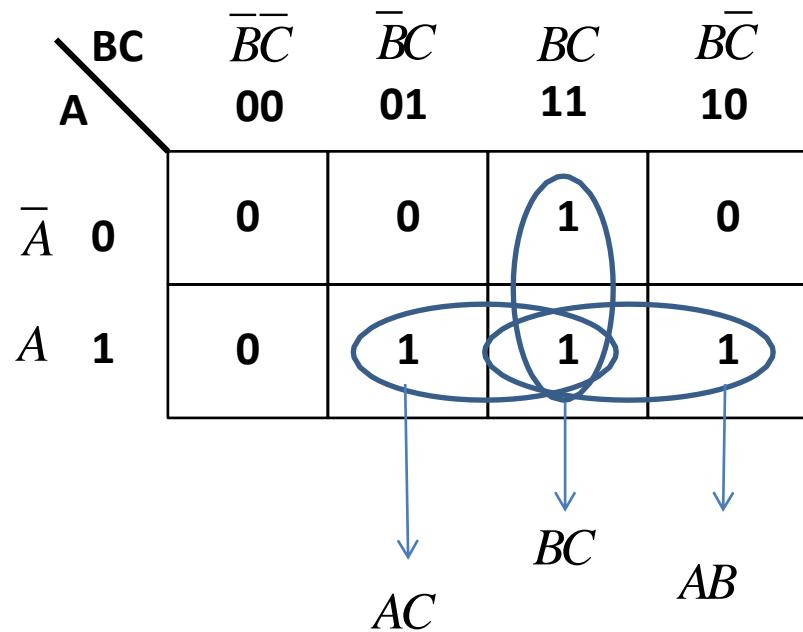
$$\text{Let } X = A \oplus B$$

$$\therefore S = C \oplus A \oplus B$$

# Full Adder

---

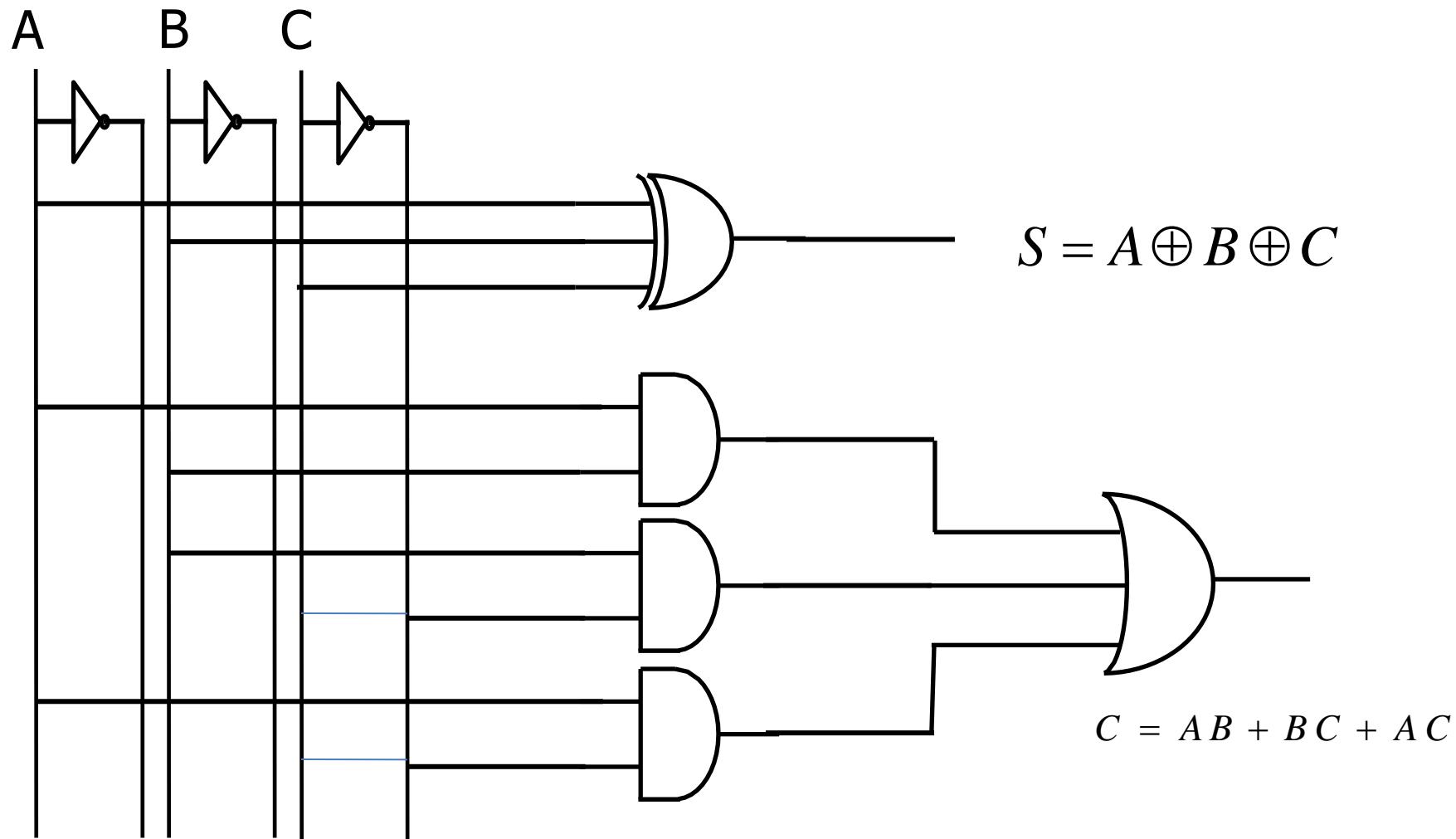
K-map for Carry Output:



$$C = AB + BC + AC$$

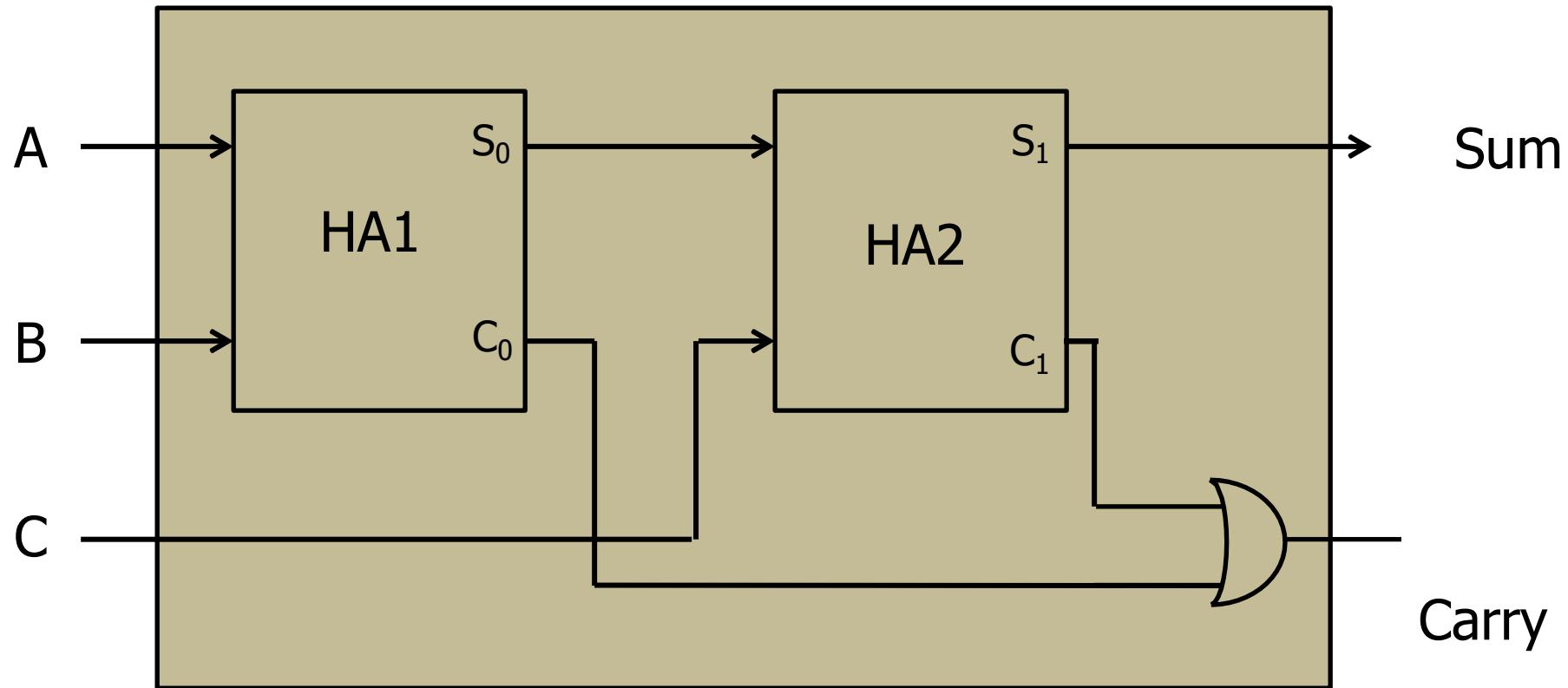
# Full Adder

## Logic Diagram:

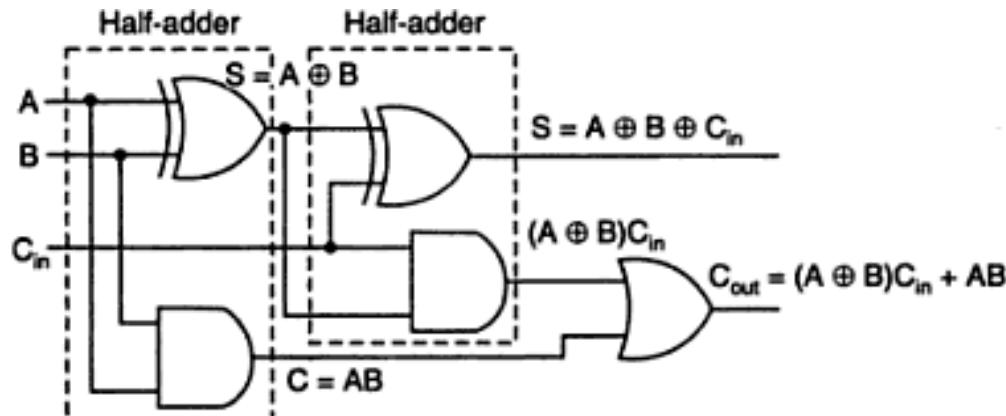


# Full Adder using Half Adders

---

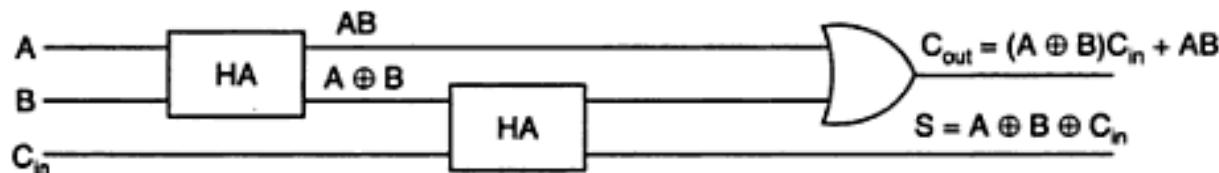


Draw the full adder circuit using two half adders.



Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is :

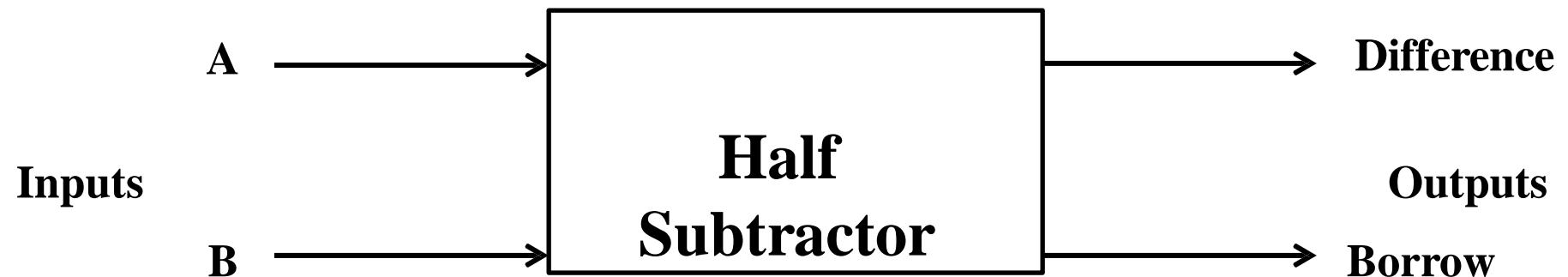


Block diagram of a full-adder using two half-adders.

## Half Subtractor

---

- ✓ Half subtractor is a combinational logic circuit with two inputs and two outputs.
- ✓ It is a basic building block for subtraction of two single bit numbers.



# Half Subtractor

---

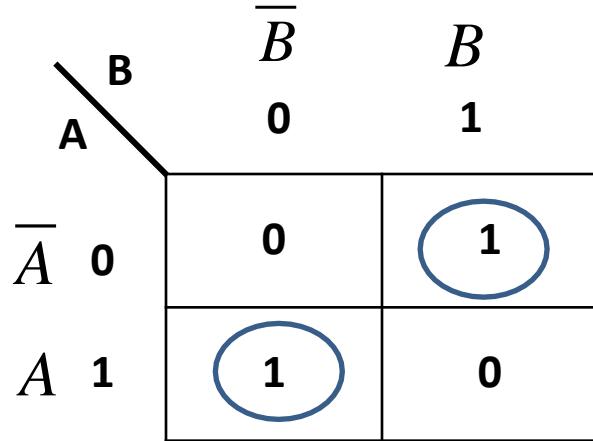
## Truth Table

Input		Output	
A	B	Difference (D)	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

# Half Subtractor

---

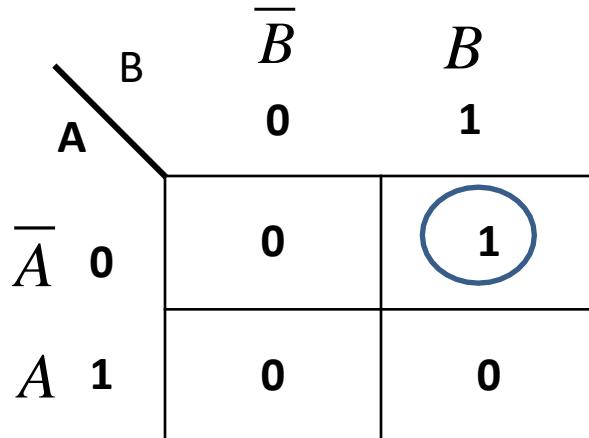
K-map for Difference Output:



$$D = \bar{A}\bar{B} + A\bar{B}$$

$$D = A \oplus B$$

K-map for Borrow Output:

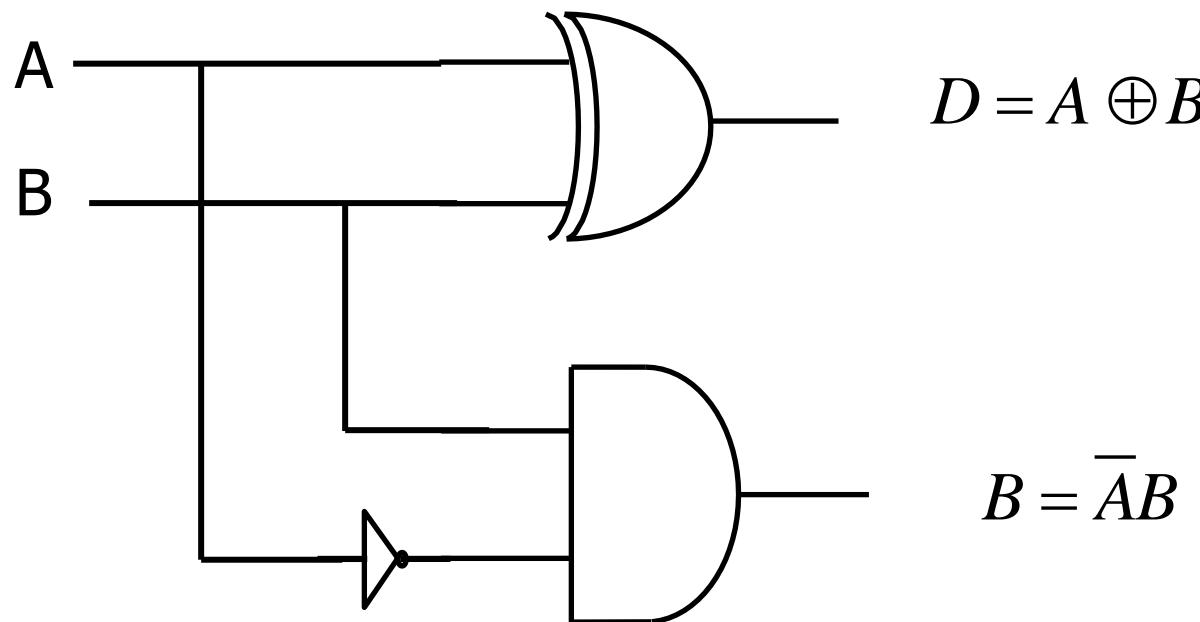


$$B = \bar{A}\bar{B}$$

# Half Subtractor

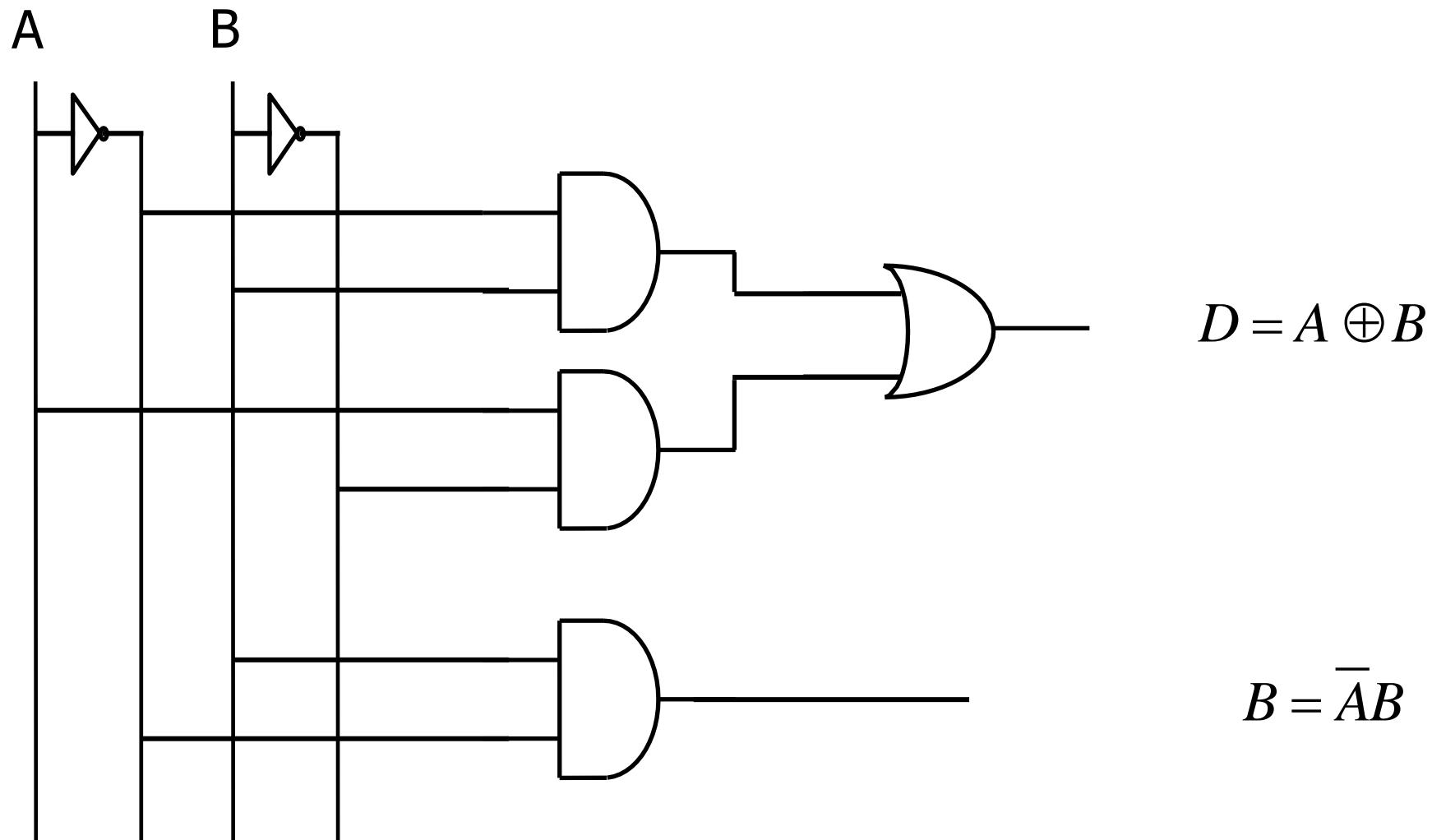
---

Logic Diagram:



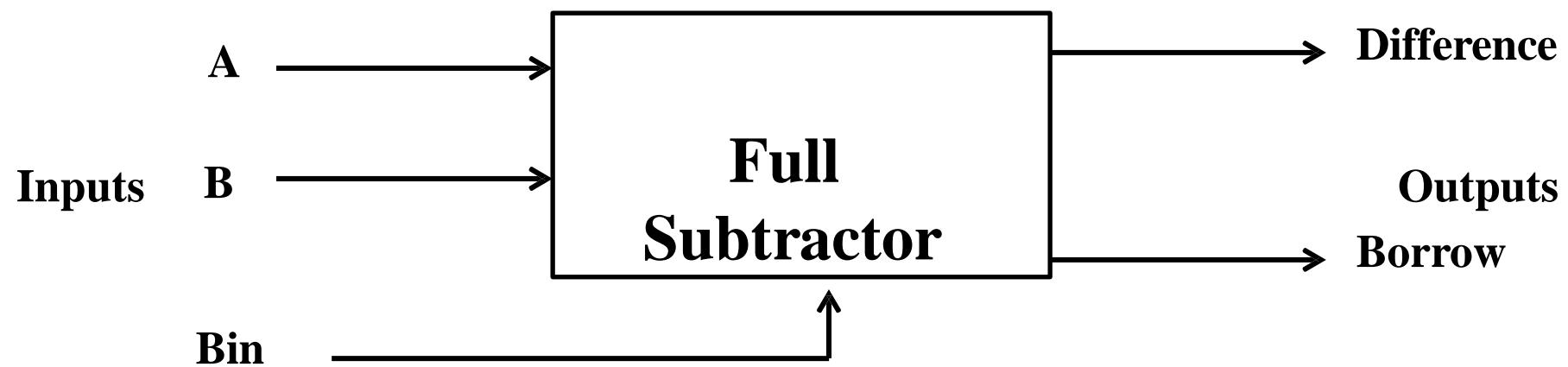
# Half Subtractor

Logic Diagram using Basic Gates:



# Full Subtractor

✓ Full subtractor is a combinational logic circuit with three inputs and two outputs.



# Full Subtractor

## Truth Table

Inputs			Outputs	
A	B	Bin (C)	Difference (D)	Borrow (B0)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

# Full Subtractor

K-map for Difference Output:

		$\bar{B}\bar{C}$	$\bar{B}C$	$BC$	$B\bar{C}$
		00	01	11	10
		0	1	0	1
$\bar{A}$	0	0	1	0	1
	1	1	0	1	0

Labels below the K-map:

- $A\bar{B}\bar{C}$  (under the first column)
- $\bar{A}\bar{B}\bar{C}$  (under the second column)
- $ABC$  (under the third column)
- $\bar{A}\bar{B}\bar{C}$  (under the fourth column)

$$D = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + ABC + A\bar{B}\bar{C}$$

$$D = \bar{A}\bar{B}\bar{C} + ABC + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$

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

Let  $\bar{A}B + A\bar{B} = X$

$$\therefore D = C(\bar{X}) + \bar{C}(X)$$

$$D = C \oplus X$$

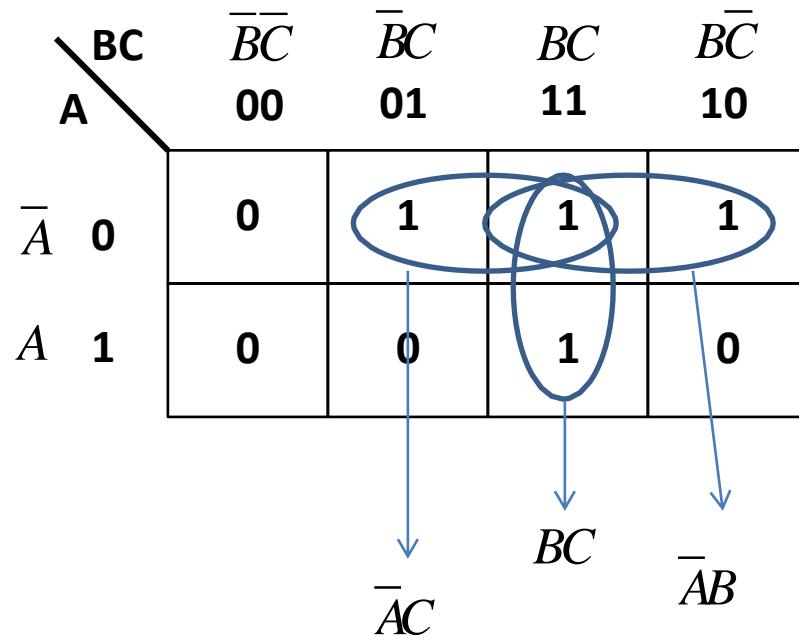
Let  $X = A \oplus B$

$$\therefore D = C \oplus A \oplus B$$

# Full Subtractor

---

K-map for Borrow Output:

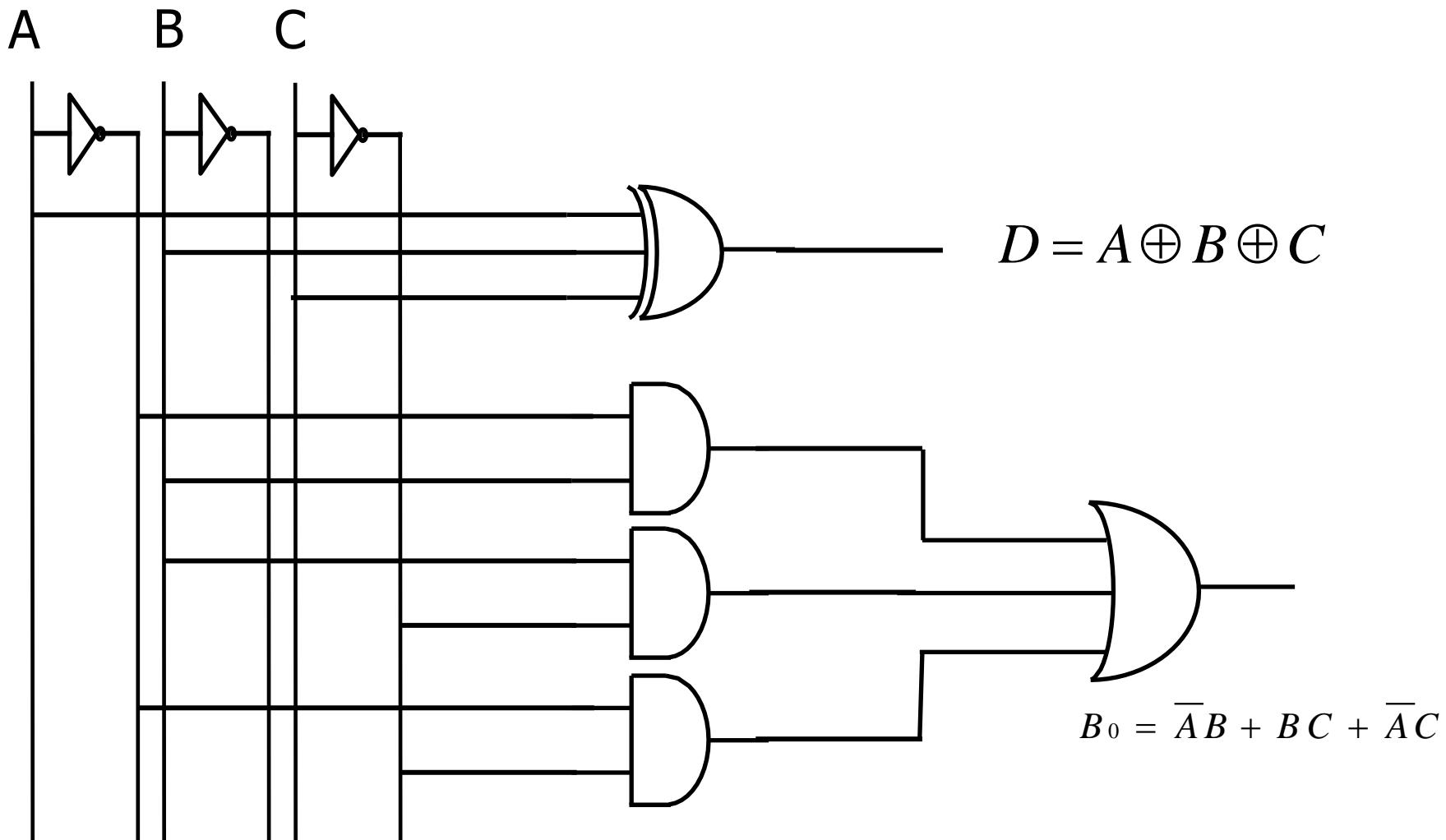


$$B_0 = \overline{A}B + BC + \overline{A}\overline{C}$$

# Full Subtractor

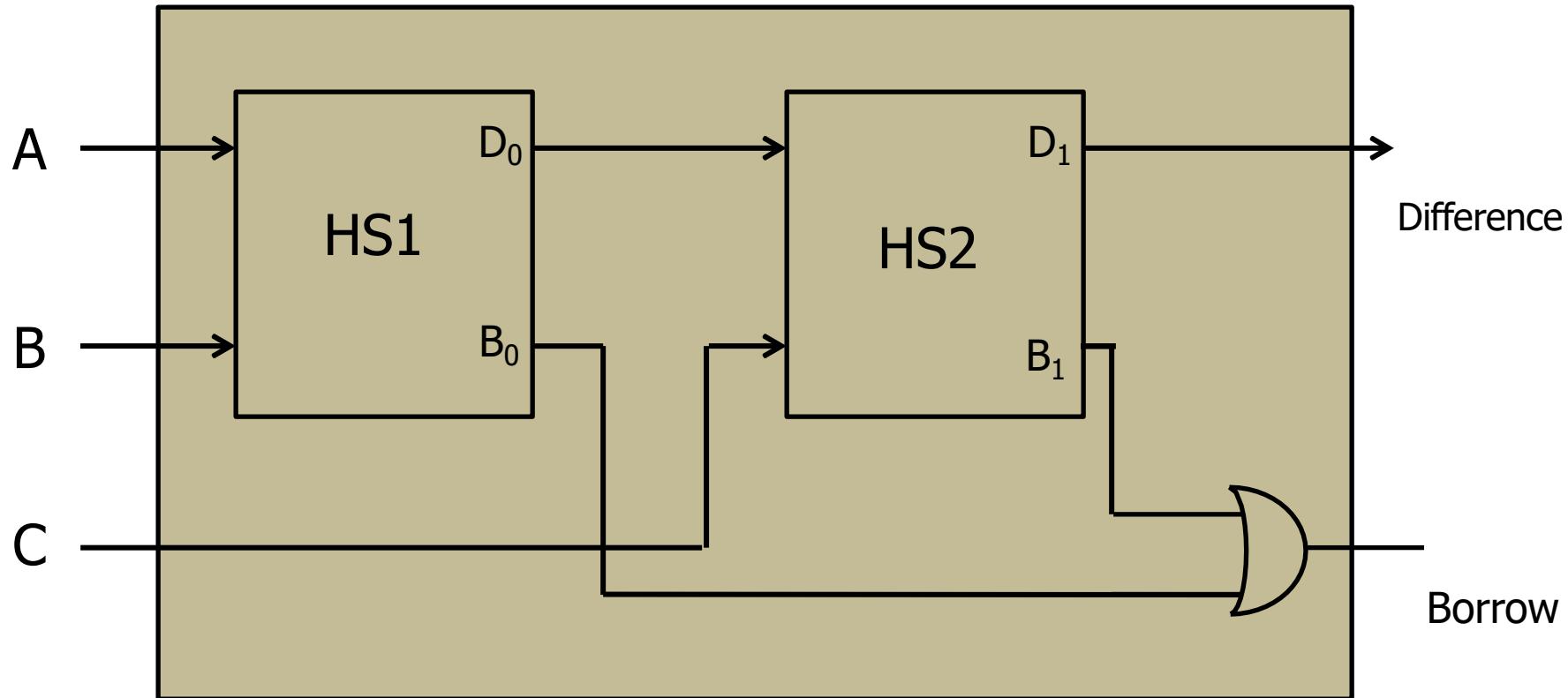
---

Logic Diagram:



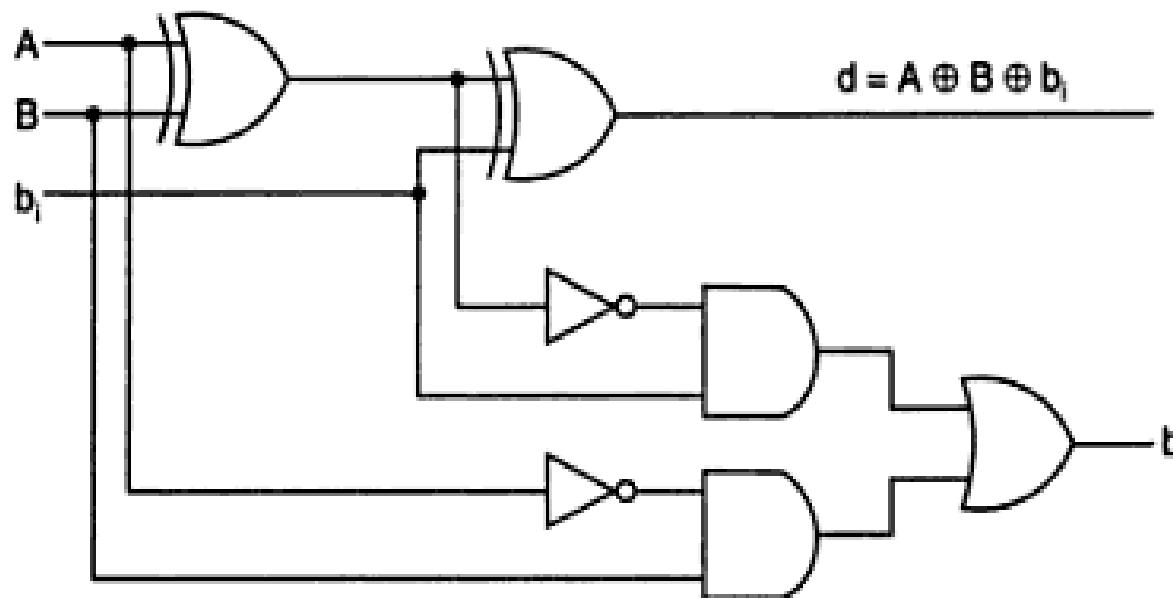
# Full Subtractor using Half Subtractor

---



**Draw a full subtractor circuit using two half subtractors.**

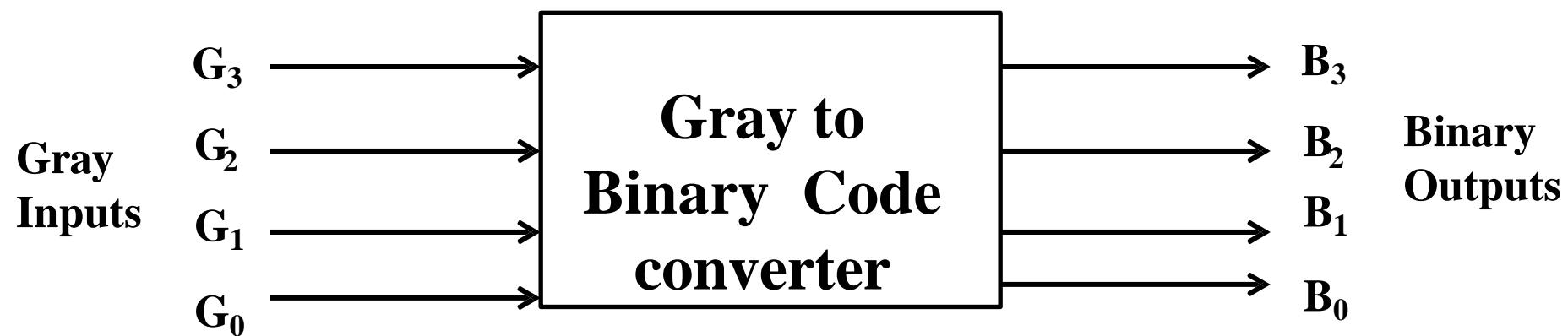
The circuit diagram is as:



Logic diagram of a full-subtractor.

# Design of Gray to Binary Code Converter

Block Diagram:



# Design of Gray to Binary Code Converter

## Truth Table :

Gray Inputs			Binary Outputs			
$G_2$	$G_1$	$G_0$	$B_3$	$B_2$	$B_1$	$B_0$
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	1
0	1	1	0	0	1	0
1	0	0	0	1	1	0
1	0	1	0	1	1	1
1	1	0	0	1	0	1
1	1	1	0	1	0	0

Gray Inputs				Binary Outputs			
$G_3$	$G_2$	$G_1$	$G_0$	$B_3$	$B_2$	$B_1$	$B_0$
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

# Design of Gray to Binary Code Converter

K-map for  $B_0$ :

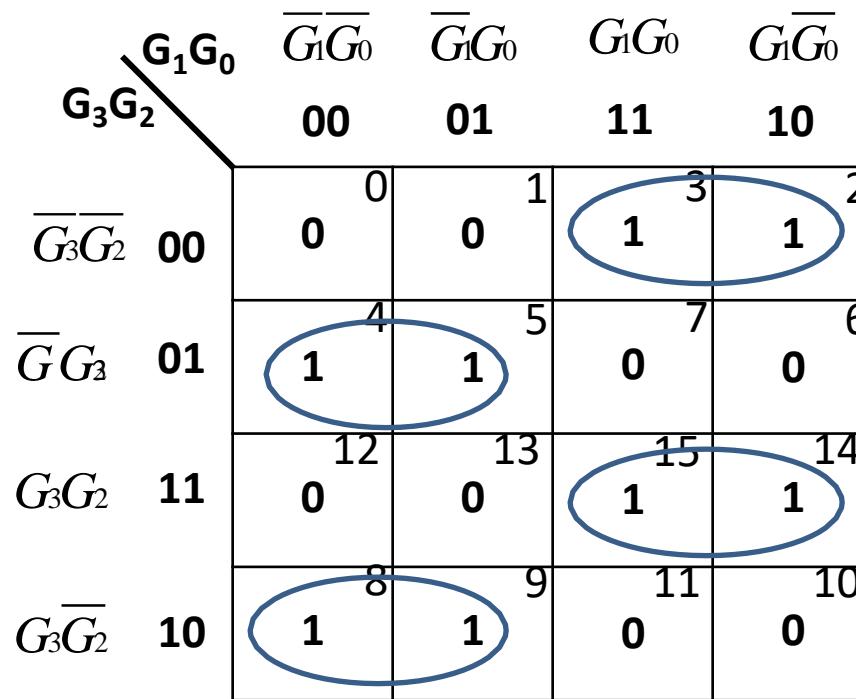
		$\overline{G_1}G_0$	$\overline{G_1}\overline{G_0}$	$\overline{G}_1G_0$	$G_1G_0$	$G_1\overline{G_0}$
		$G_3G_2$	00	01	11	10
$G_3\overline{G_2}$	00	0	1	3	2	
	01	4	5	7	6	
	11	12	13	15	14	
	10	8	9	11	10	

$$\begin{aligned}B_0 &= \overline{G_3}\overline{G_2}\overline{G_1}G_0 + \overline{G_3}\overline{G_2}G_1\overline{G_0} + \overline{G_3}G_2\overline{G_1}\overline{G_0} + \overline{G_3}G_2G_1G_0 \\&\quad + G_3\overline{G_2}\overline{G_1}G_0 + G_3\overline{G_2}G_1G_0 + G_3G_2\overline{G_1}G_0 + G_3G_2G_1\overline{G_0}\end{aligned}$$

$$B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$$

# Design of Gray to Binary Code Converter

K-map for  $B_1$ :

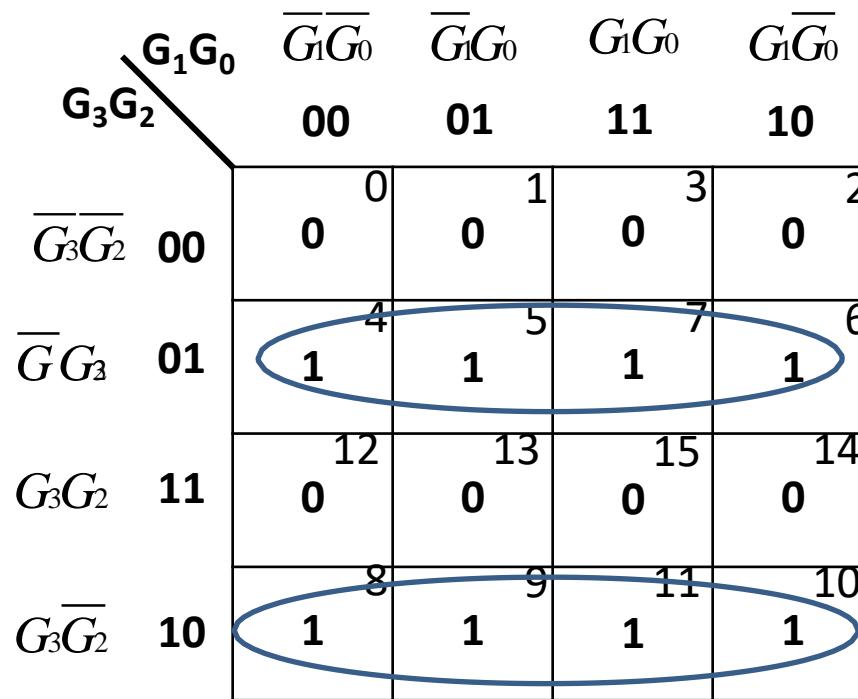


$$B_1 = \overline{G_3}\overline{G_2}G_1 + \overline{G_3}G_2\overline{G}_1 + G_3G_2G_1 + G_3\overline{G}_2\overline{G}_1$$

$$B_1 = G_3 \oplus G_2 \oplus G_1$$

# Design of Gray to Binary Code Converter

K-map for  $B_2$ :



$$B_2 = \overline{G_3}G_2 + G_3\overline{G_2}$$

$$B_1 = G_3 \oplus G_2$$

# Design of Gray to Binary Code Converter

---

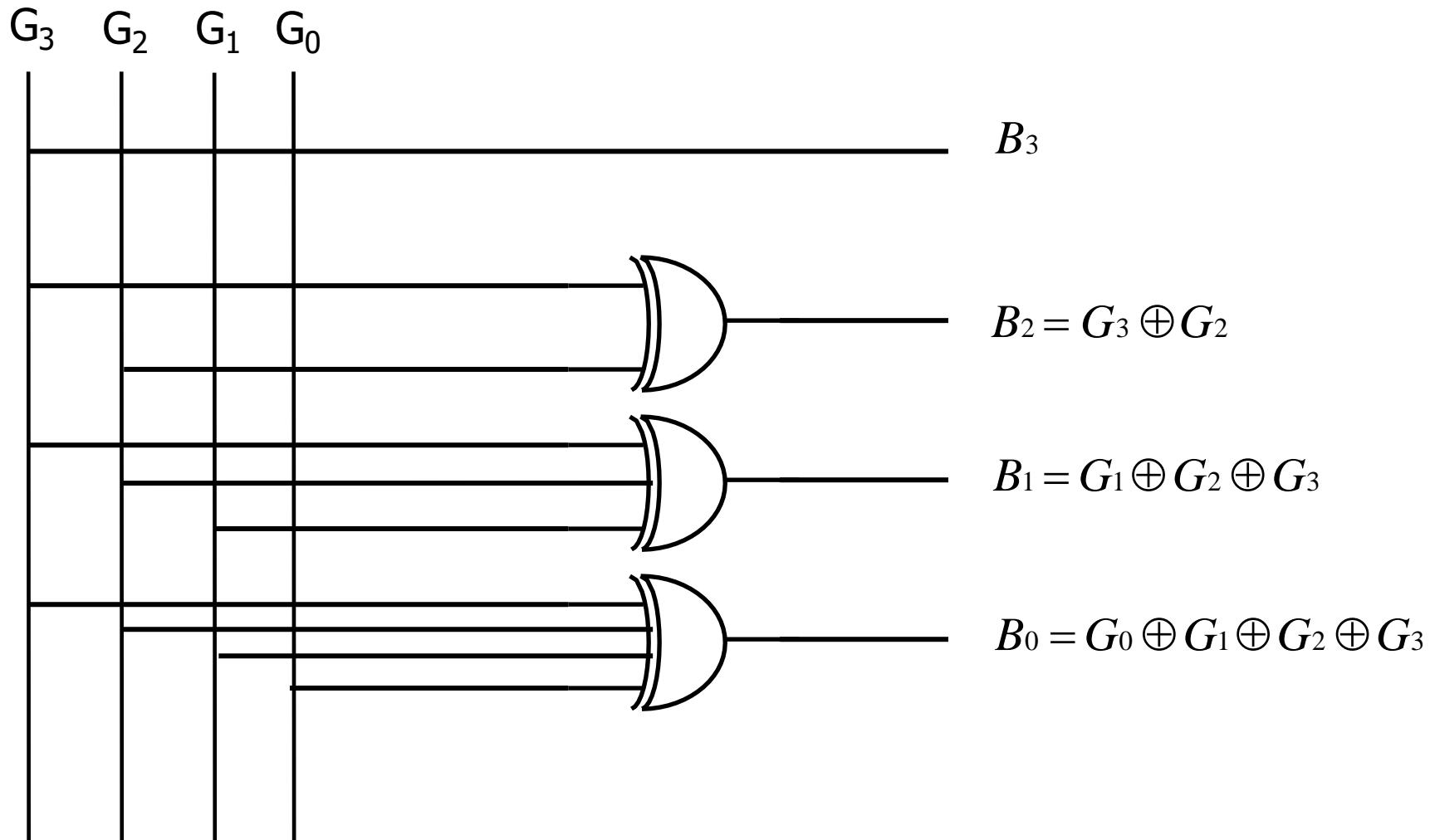
K-map for  $B_3$ :

		$G_1 G_0$	$\overline{G_1} \overline{G_0}$	$\overline{G}_1 G_0$	$G_1 G_0$	$G_1 \overline{G}_0$
		00	01	11	10	
$G_3 G_2$	00	0	1	3	2	
	01	4	5	7	6	
$G_3 G_2$	11	12	13	15	14	
	10	1	1	1	1	

$$B_3 = G_3$$

# Design of Gray to Binary Code Converter

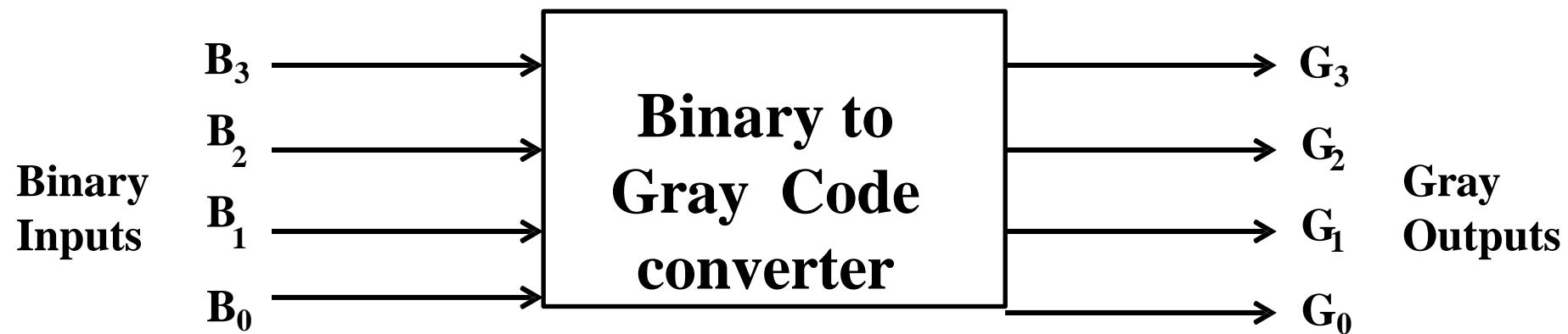
Logic Diagram:



# Design of Binary to Gray Code Converter

---

Block Diagram:



# Design of Binary to Gray Code Converter

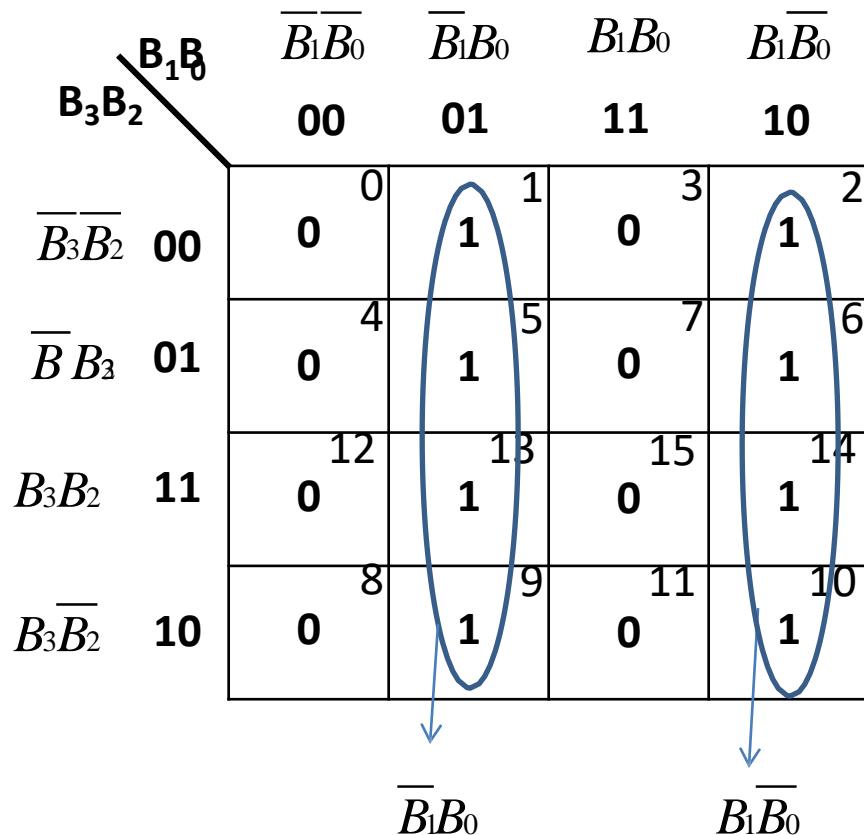
Truth Table :

Binary Inputs				Gray Outputs			
B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	G <sub>3</sub>	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	1	0	1	0

Binary Inputs				Gray Outputs			
B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	G <sub>3</sub>	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

# Design of Binary to Gray Code Converter

K-map for  $G_0$ :

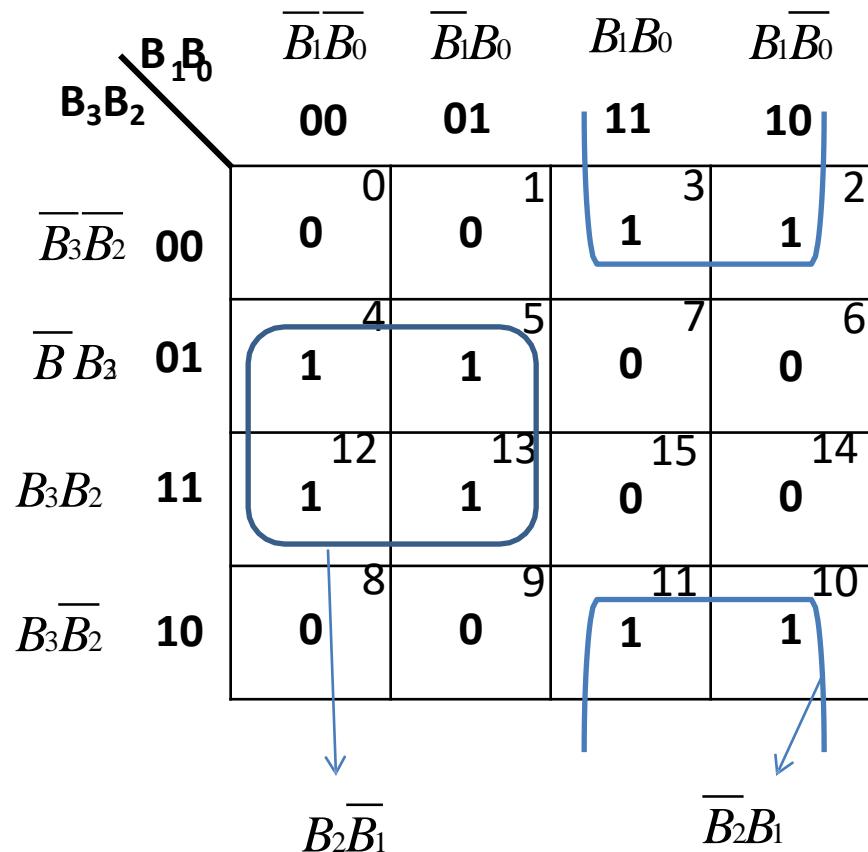


$$G_0 = \overline{B_1}\overline{B_0} + B_1\overline{B_0}$$

$$\therefore G_0 = B_0 \oplus B_1$$

# Design of Binary to Gray Code Converter

K-map for  $G_1$ :

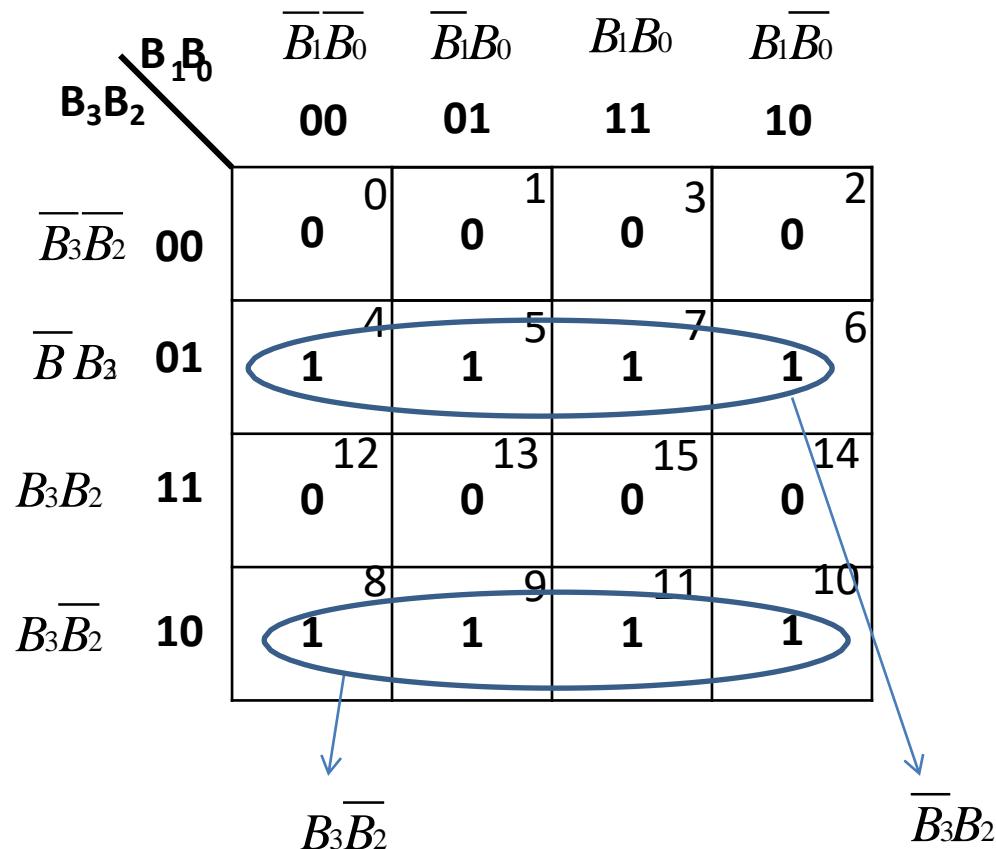


$$G_1 = \overline{B}_2B_1 + B_2\overline{B}_1$$

$$\therefore G_1 = B_2 \oplus B_1$$

# Design of Binary to Gray Code Converter

K-map for  $G_2$ :

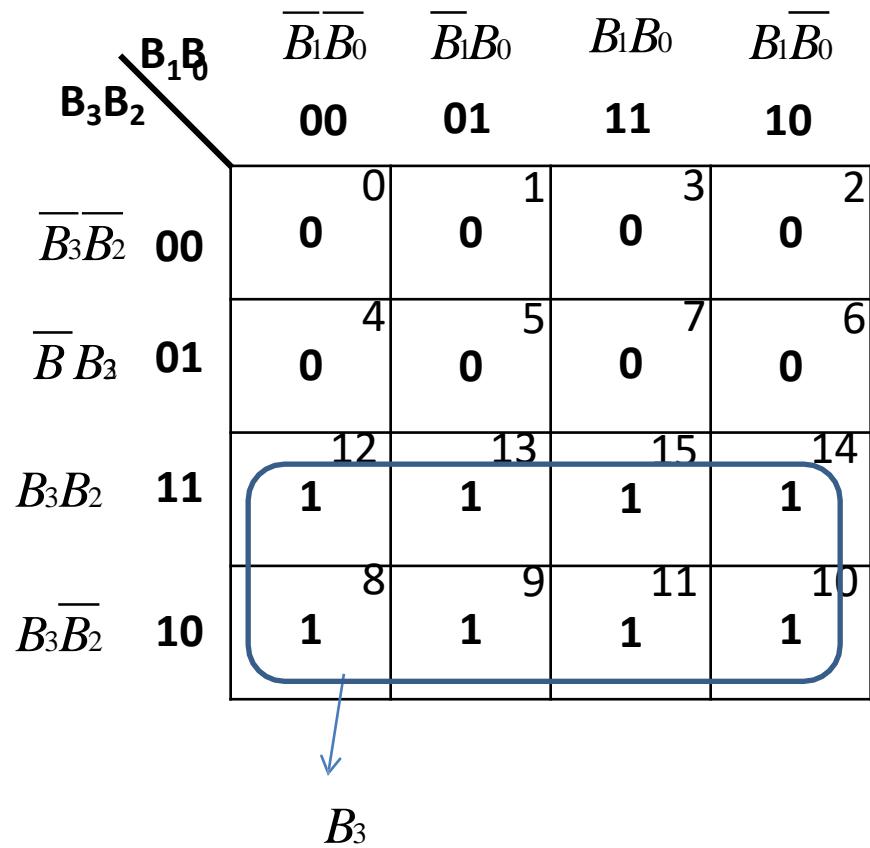


$$G_2 = \overline{B}_3B_2 + B_3\overline{B}_2$$

$$\therefore G_2 = B_3 \oplus B_2$$

# Design of Binary to Gray Code Converter

K-map for  $G_3$ :

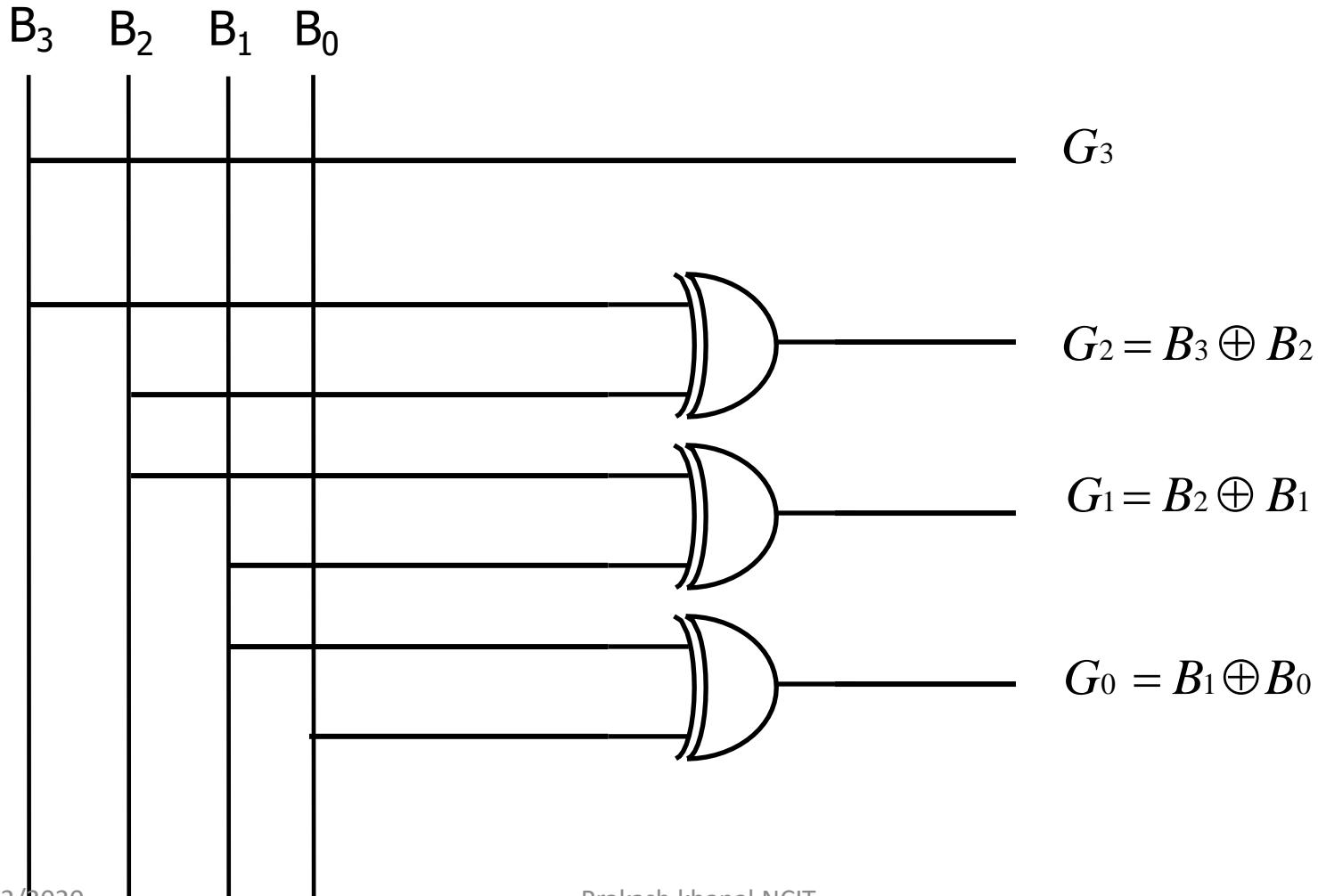


$$G_3 = B_3$$

# Design of Binary to Gray Code Converter

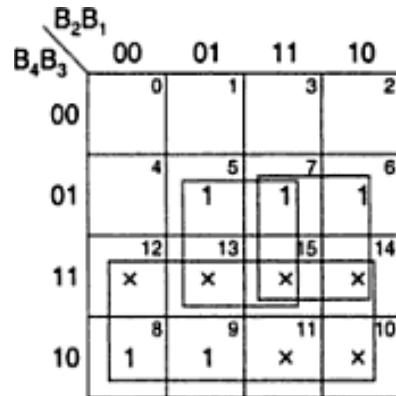
---

## Logic Diagram:



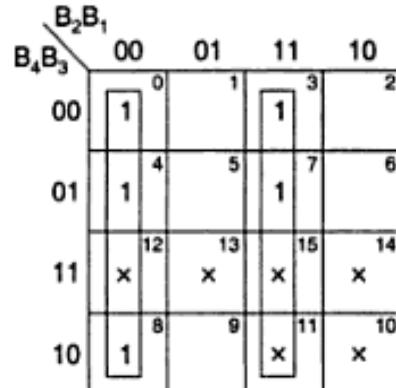
## Design of a 4-bit BCD to XS-3 code converter:

8421 code				XS-3 code			
$B_4$	$B_3$	$B_2$	$B_1$	$X_4$	$X_3$	$X_2$	$X_1$
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



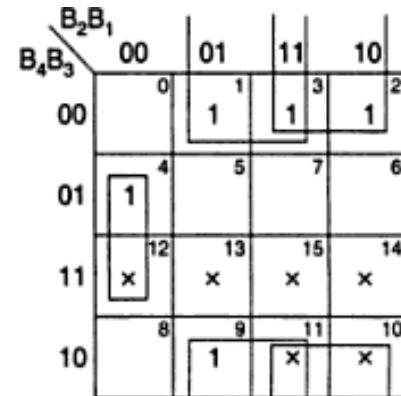
$$X_4 = B_4 + B_3B_2 + B_3B_1$$

K-map for  $X_4$



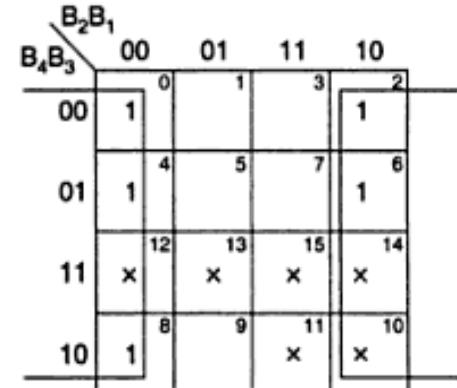
$$X_2 = \bar{B}_2\bar{B}_1 + B_2B_1$$

K-map for  $X_2$



$$X_3 = B_3\bar{B}_2\bar{B}_1 + \bar{B}_3B_1 + \bar{B}_3B_2$$

K-map for  $X_3$



(c) K-maps

4-bit BCD-to-XS-3 code converter.

The o/p functions and logic diagram is as:

The minimal expressions are

$$X_4 = B_4 + B_3B_2 + B_3B_1$$

$$X_3 = B_3\bar{B}_2\bar{B}_1 + \bar{B}_3B_1 + \bar{B}_3B_2$$

$$X_2 = \bar{B}_2\bar{B}_1 + B_2B_1$$

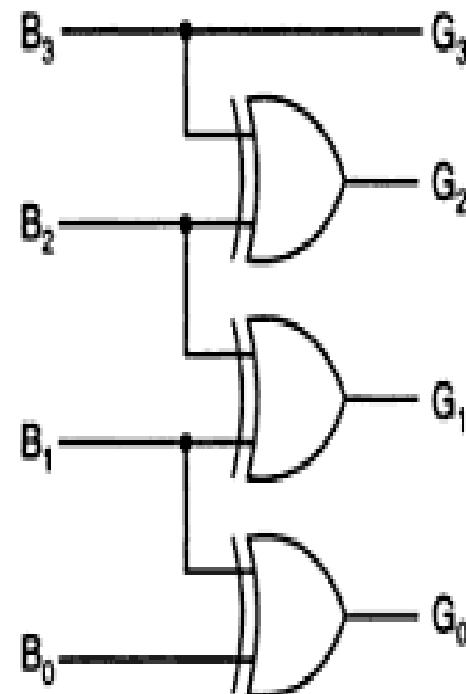
$$X_1 = \bar{B}_1$$

- Draw the logic diagram yourself.

## Design of a BCD to gray code converter:

BCD code				Gray code			
B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	G <sub>3</sub>	G <sub>2</sub>	G <sub>1</sub>	G <sub>0</sub>
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

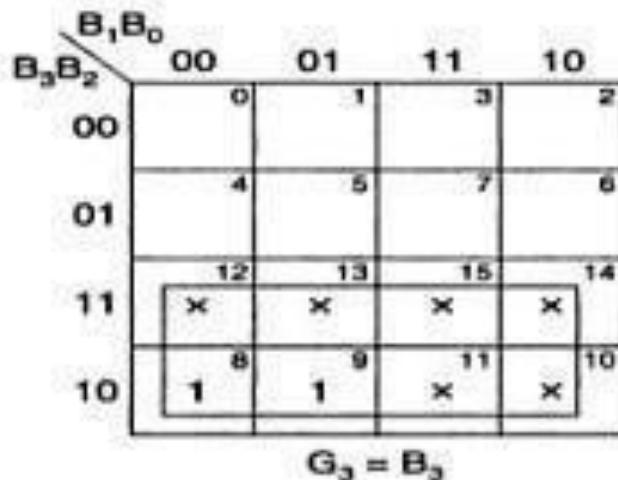
(a) BCD-to-Gray code conversion table



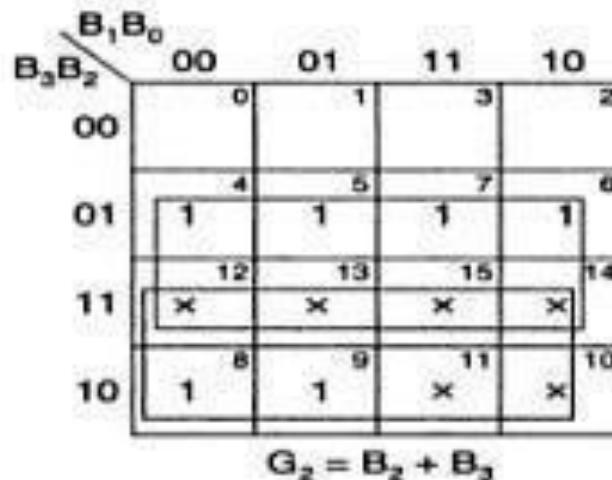
(b) Logic diagram

BCD-to-Gray code converter.

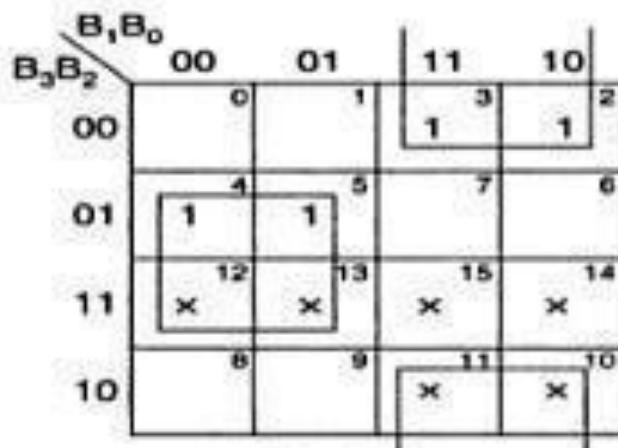
# Using K-map,



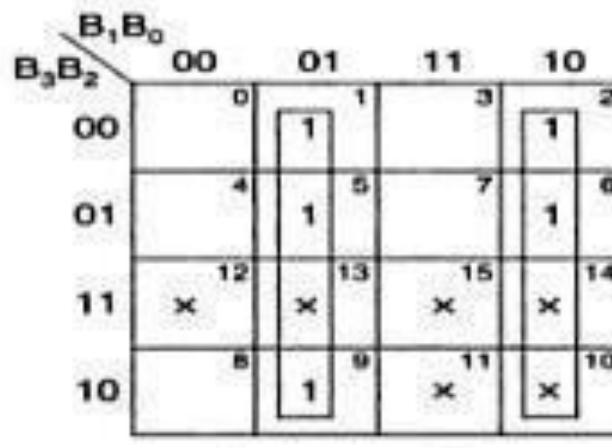
$$G_3 = B_3$$



$$G_2 = B_2 + B_3$$



$$G_1 = B_2B_3 + B_2B_1 = B_2 \oplus B_1$$



$$G_0 = B_1B_0 + B_1 \cdot \bar{B}_0 = B_1 \oplus B_0$$

K-maps for a BCD-to-Gray code converter.

Prakash Khanal,NCHT

## Design of a Combinational circuit to produce the 2's complement of a 4-bit binary number:

Input				Output			
A	B	C	D	E	F	G	H
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0
0	0	1	1	1	1	0	1
0	1	0	0	1	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

(a) Conversion table

Conversion table and K-maps for the circuit

- Simplify the o/p and draw the logic diagram yourself.

K Map for:

AB		CD	
00	01	11	10
00	1	1	1
01	1	1	1
11			
10	1		

$\rightarrow \bar{A}B$   
 $\rightarrow \bar{A}C$   
 $\rightarrow \bar{A}D$

$$AB\bar{C}\bar{D}$$

$$\therefore E = \bar{A}B + \bar{A}C + \bar{A}D + AB\bar{C}\bar{D}$$

F:

AB		CD	
00	01	11	10
00	1	1	1
01	1		
11			
10			

$B\bar{C}D$      $\bar{B}D$      $\bar{B}\bar{C}$

$$\therefore F = \bar{B}D + \bar{B}\bar{C} + B\bar{C}D$$

G:

AB		CD	
00	01	11	10
00	1		1
01	1		1
11	1		1
10	1		1

$\bar{D}$      $C\bar{D}$

$$\therefore G = \bar{C}D + C\bar{D}$$

$$= C \oplus D$$

H:

AB		CD	
00	01	11	10
00	1	1	
01	1	1	
11		1	1
10		1	1

$$\therefore H = D$$

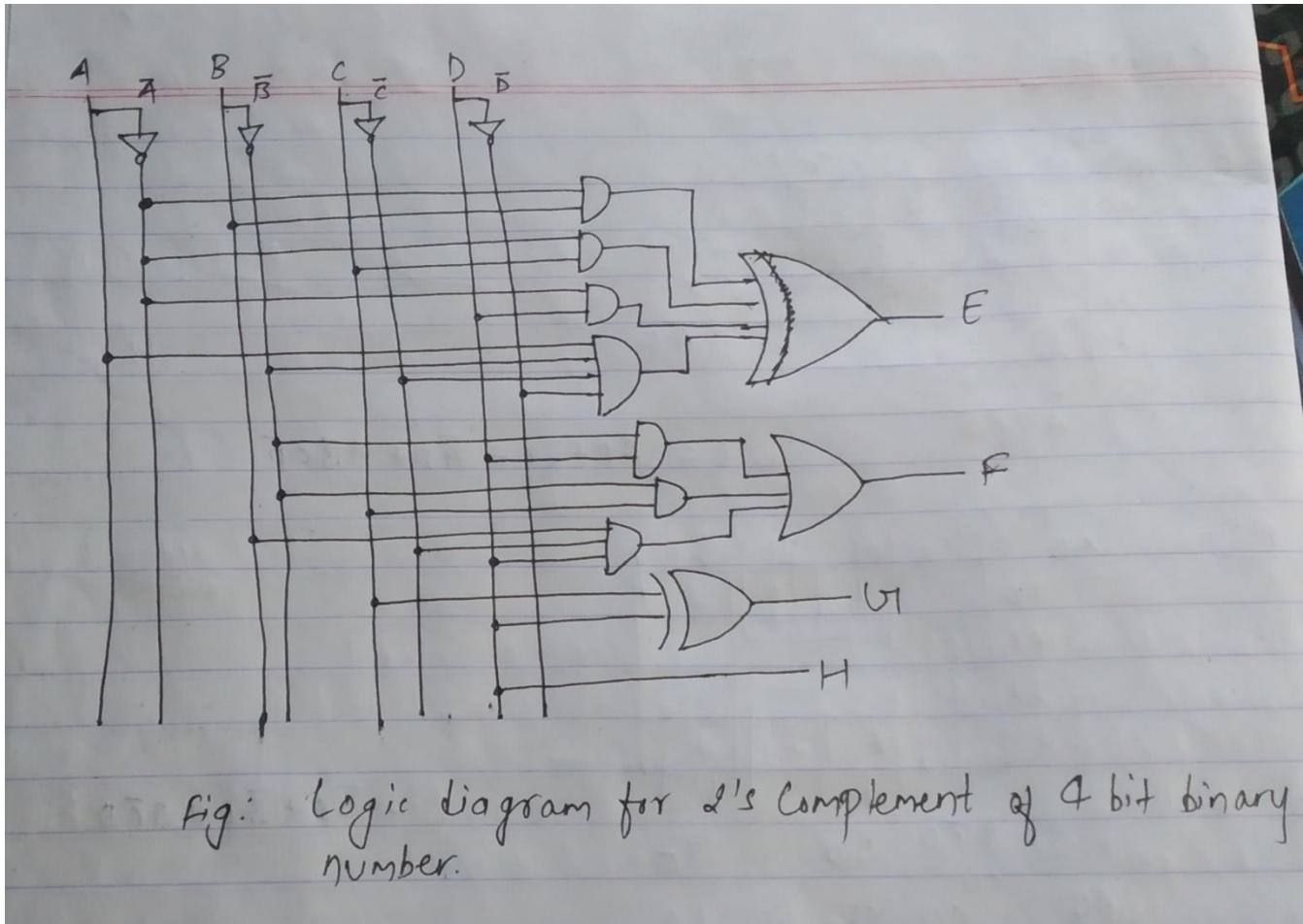


Fig: Logic diagram for 2's Complement of 4 bit binary number.

# Parity method for error detection and correction:

- Parity bit is an extra bit included with a binary message to make the number of 1's either odd or even.
- The message including the parity bit, is transmitted and checked at the receiving end for errors.
- An error is detected if the checked parity does not correspond to the one transmitted.
- The circuit that generates the parity bit in the transmitter is called a parity generator, the circuit that checks the parity in the receiver is called a parity checker.

# Design a circuit for parity generator for 3 bit message with even and odd parity:

x	y	z	Odd Parity bit generated(p)
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Simplify the parity using k map and design the circuit.

# Design a circuit for odd parity check:

Four-bits received				Parity-error check
x	y	z	P	C
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Simplify the k map for output and draw the logic diagram.

Four-bits received				Parity-error check
$x$	$y$	$z$	$P$	$C$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

# Unit-6: LSI and MSI component on combinational logic

- Binary adder and subtractor, decimal adder,
- Magnitude comparator,
- decoder and encoder,
- Multiplexer and demultiplexer,
- Read-only memory (ROM), programmable, Logic Array (PLA), Programmable Array Logic (PAL).

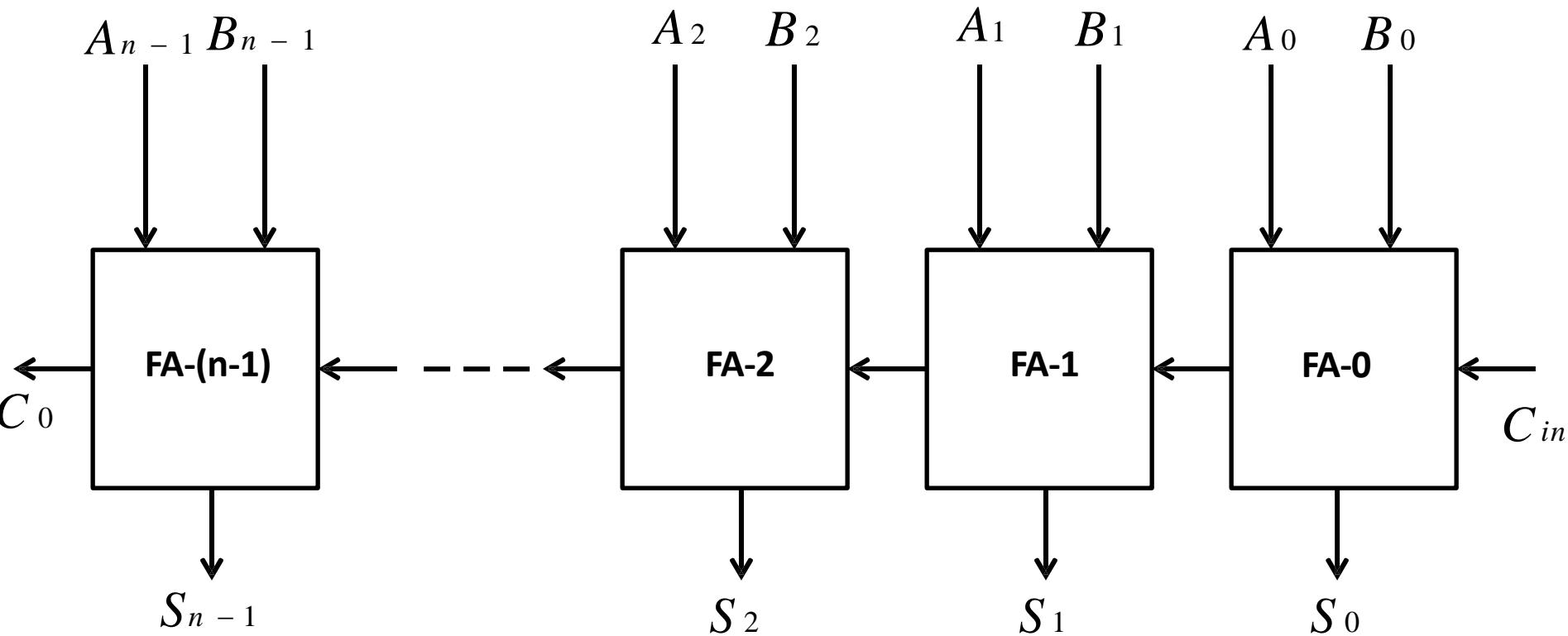
## N – Bit Parallel Adder

---

- ✓ The full adder is capable of adding two single digit binary numbers along with a carry input.
- ✓ But in practice we need to add binary numbers which are much longer than one bit.
- ✓ To add two n-bit binary numbers we need to use the n-bit parallel adder.
- ✓ It uses a number of full adders in cascade.
- ✓ The carry output of the previous full adder is connected to the carry input of the next full adder..

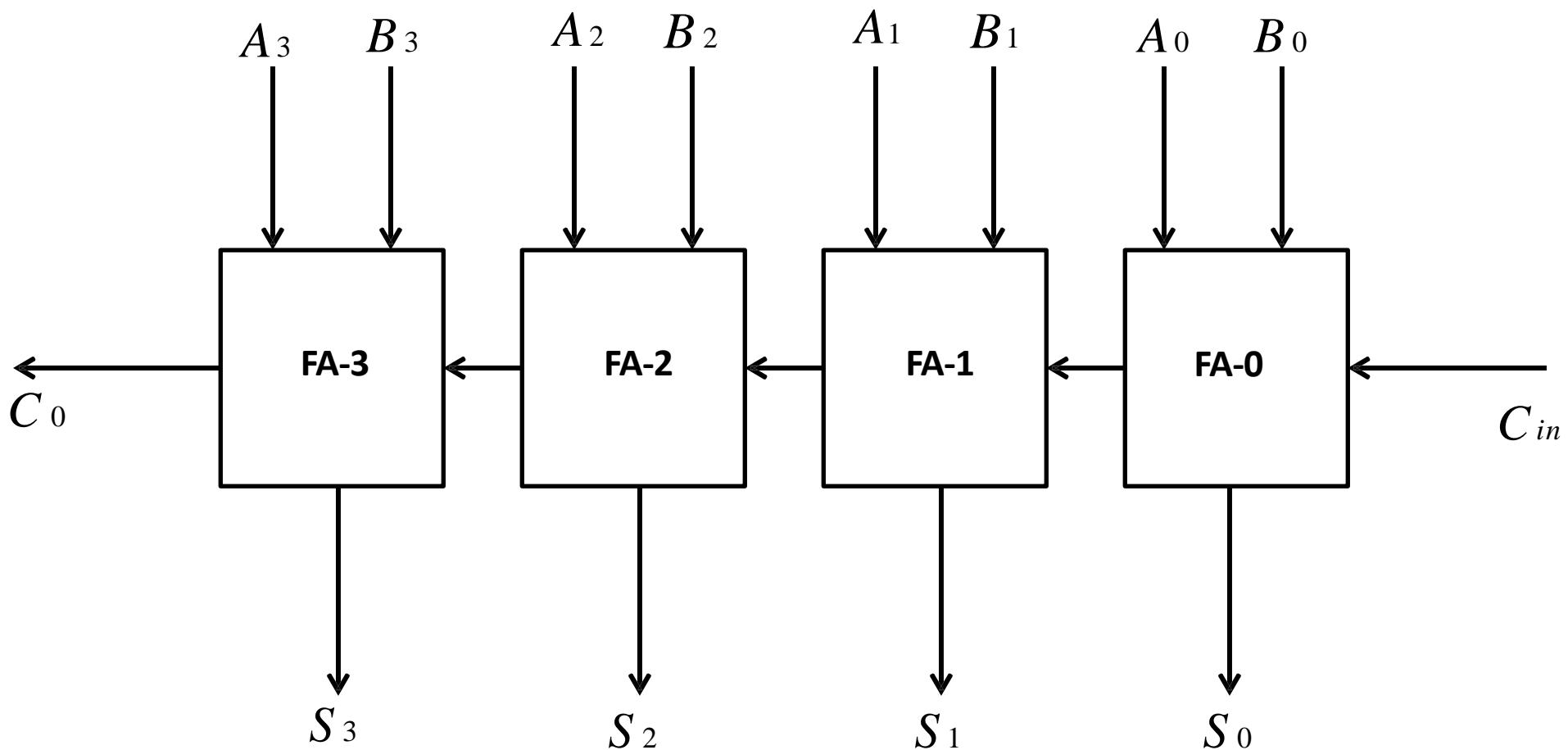
# N – Bit Parallel Adder

---

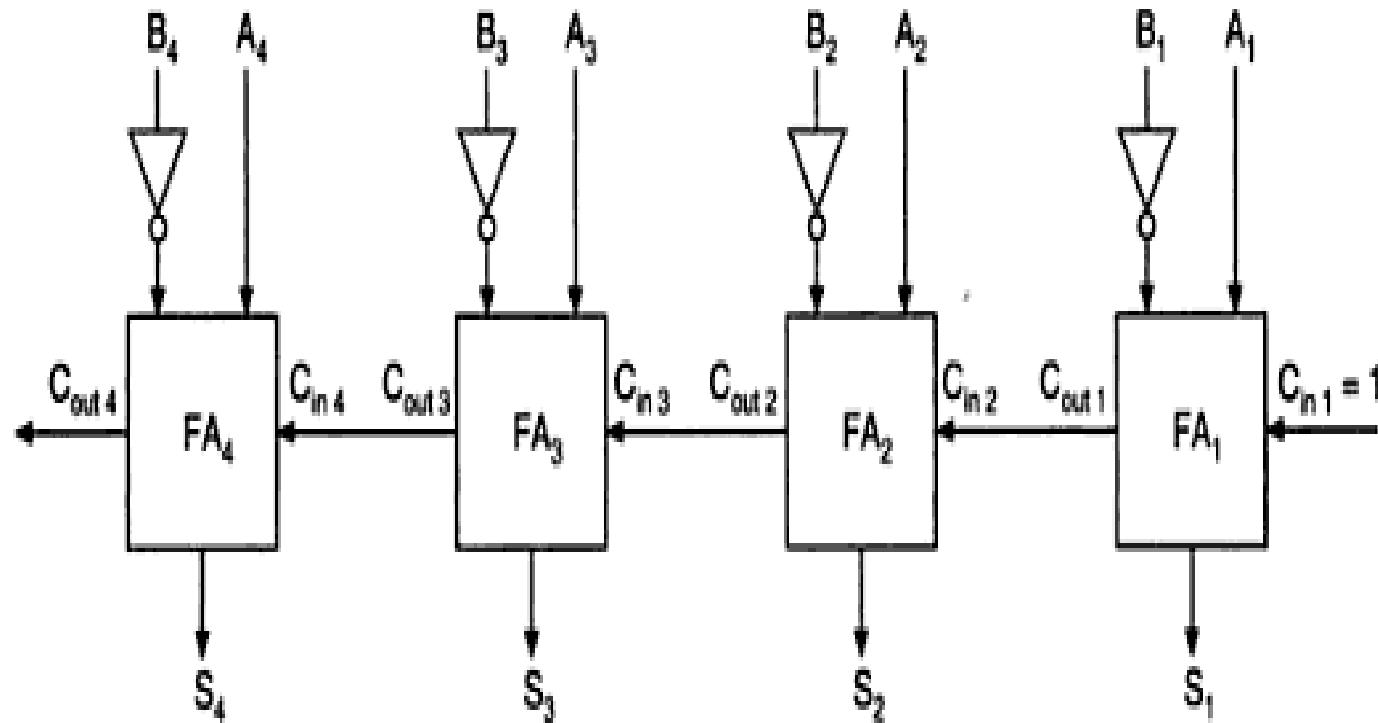


# 4 – Bit Parallel Adder using full adder

---

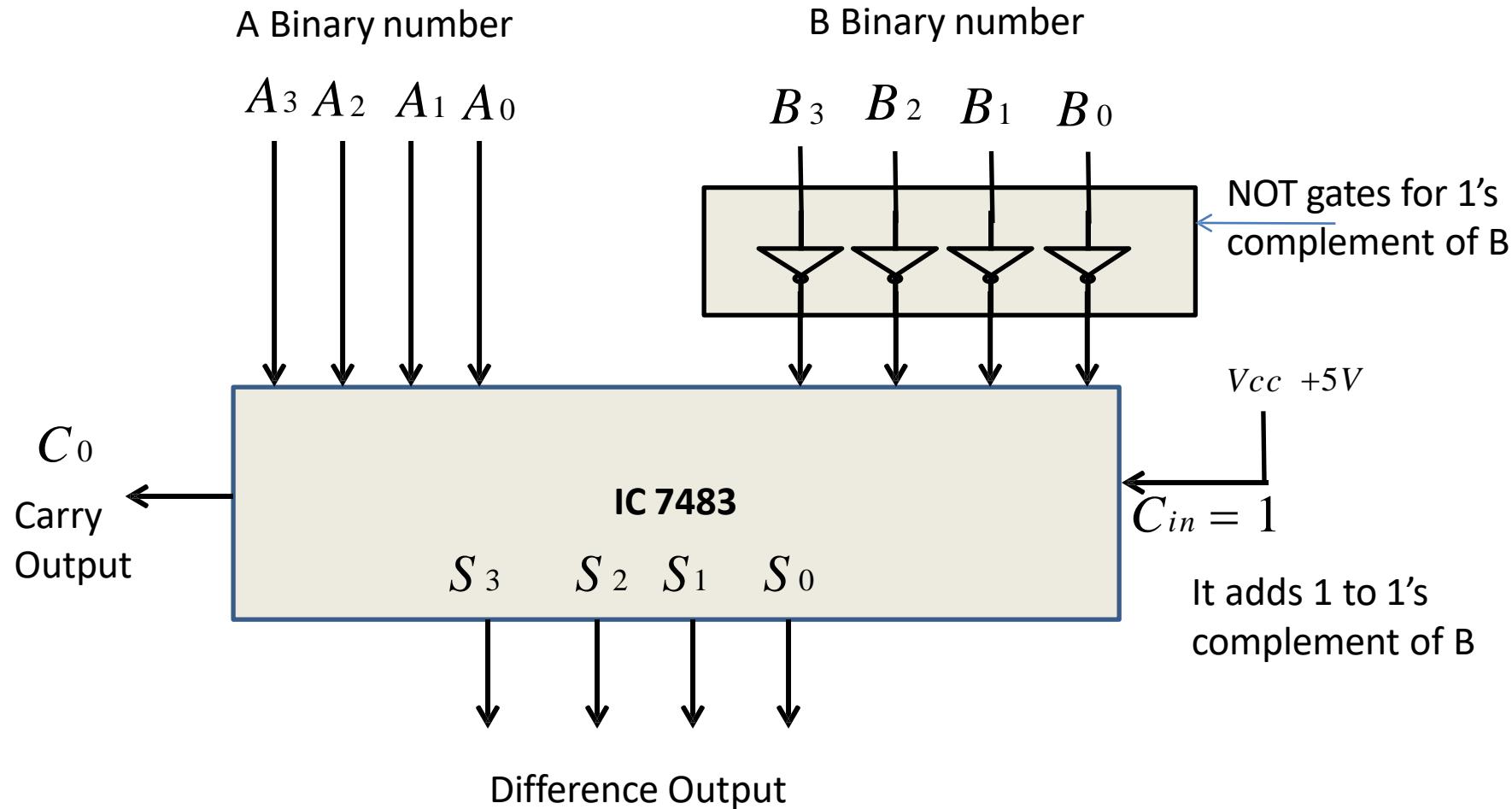


# 4 bit parallel subtractor:

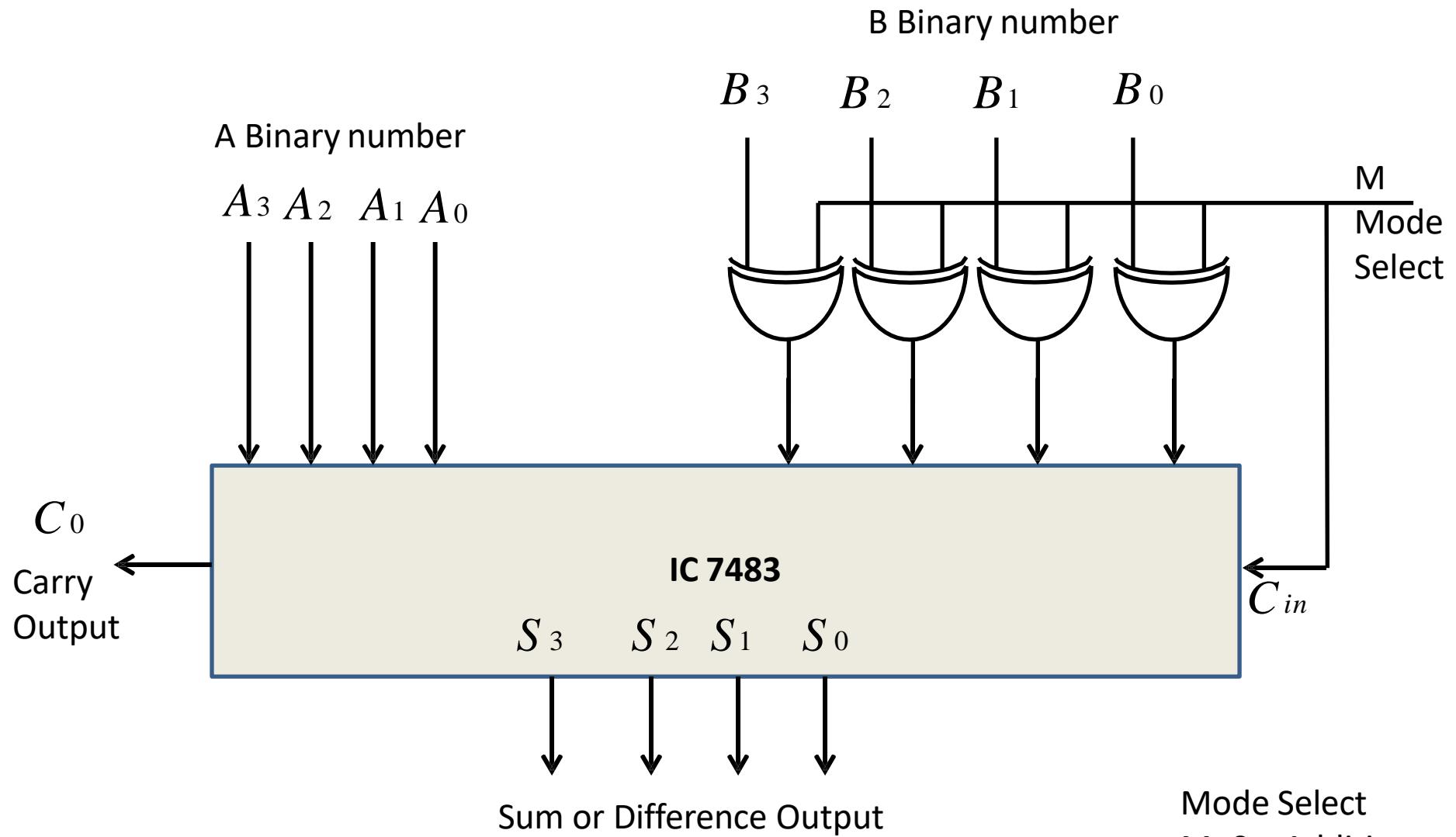


Logic diagram of a 4-bit parallel subtractor.

# 4 Bit Binary Parallel Subtractor using IC 7483

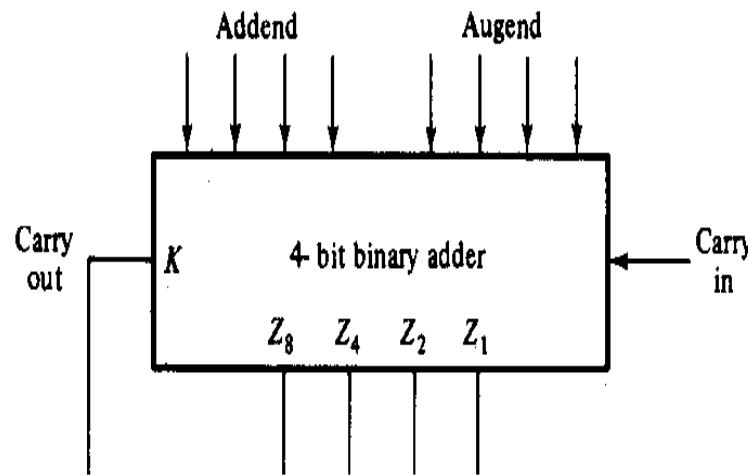


# IC 7483 as Parallel Adder/Subtractor



# BCD adder/ Decimal adder:

- If we add two decimal digits, the maximum result will be 19 (9+9+1).



- The above circuit gives the binary sum. Now we have to convert the binary sum into BCD sum. For this, we have to design a circuit which converts the binary sum into BCD sum. The truth table is as:

Binary sum					BCD sum					Decimal
K	Z <sub>1</sub>	Z <sub>2</sub>	Z <sub>3</sub>	Z <sub>4</sub>	C	S <sub>1</sub>	S <sub>2</sub>	S <sub>3</sub>	S <sub>4</sub>	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

For 0-9=>Binary sum = BCD sum

For 10-19=> Binary sum + 0110= BCD sum.

- When C=1, it is necessary to add binary 0110 to the binary sum and provide an o/p carry to the next stage.
- A correction is needed when the binary sum has an o/p carry K=1.
- Binary 1010-1111 also needs correction and have a 1 in position Z8.
- To distinguish them from binary 1000 and 1001, which also have a 1 in Z8 position, we specify further that either Z4 or Z2 must have a 1.
- The condition for the correction and o/p carry can be:

$$C = K + Z8Z4 + Z8Z2 = 1$$

(Or it can be derived using K-map for 5 variables from the truth table.)

The circuit diagram of BCD adder is as:

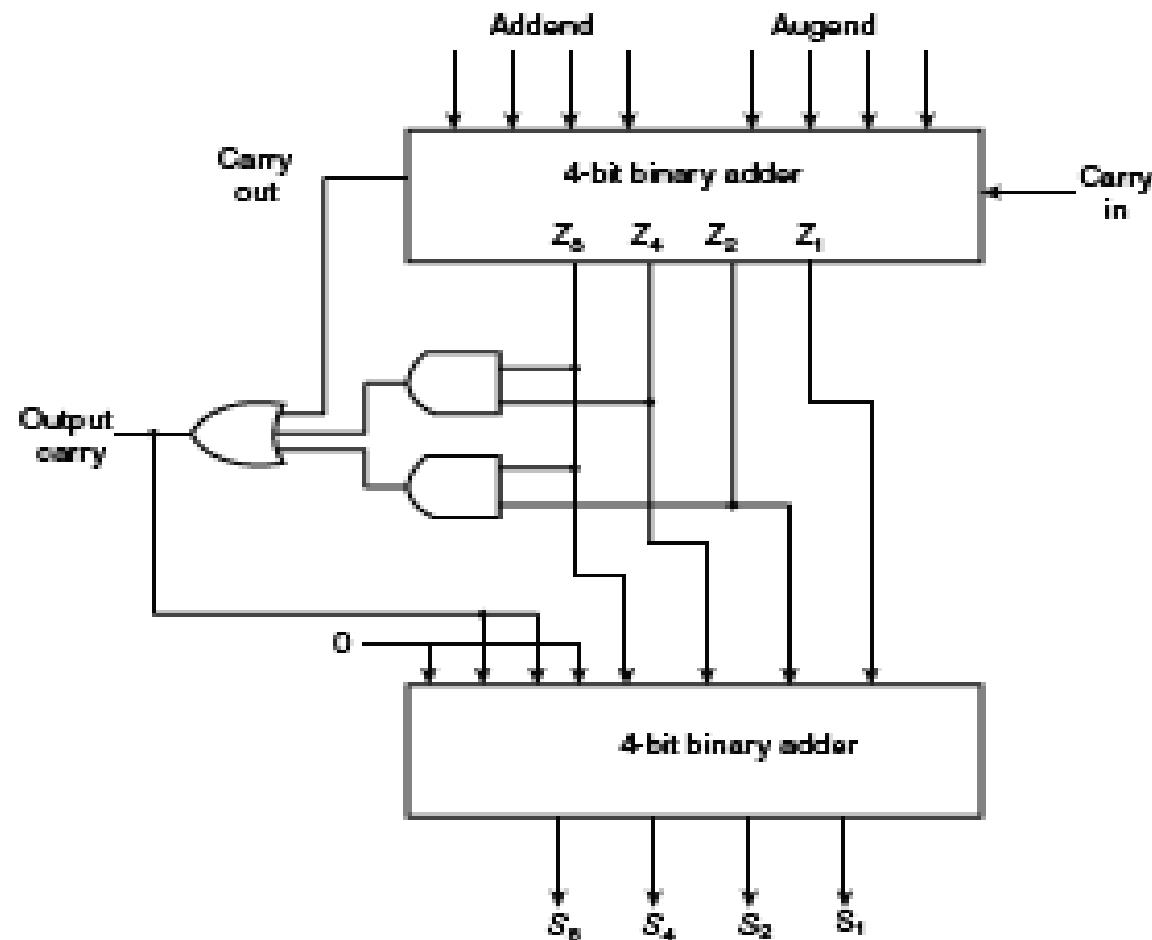


Figure 5-6 Block diagram of a BCD adder

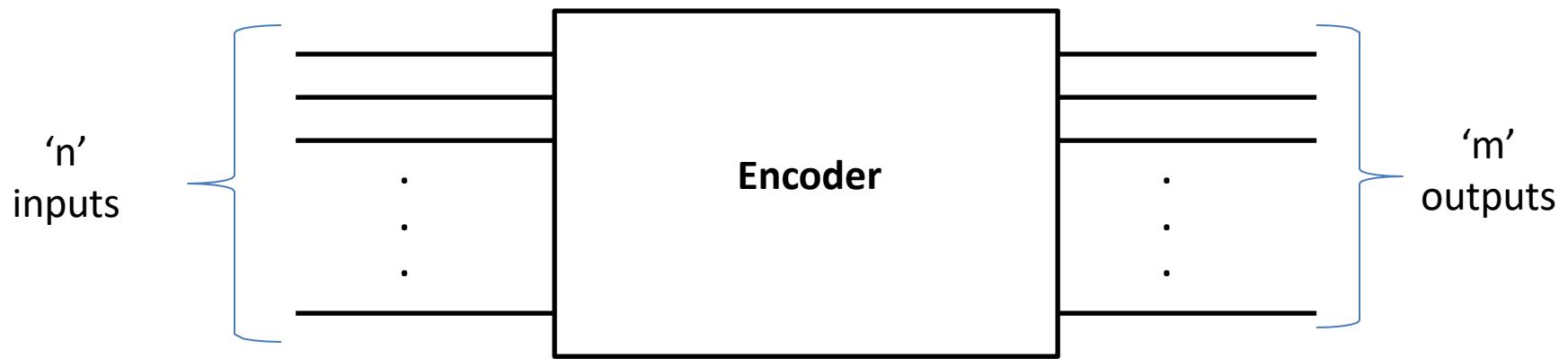
## Encoder

---

- ✓ Encoder is a combinational circuit which is designed to perform the inverse operation of decoder.
- ✓ An encoder has ‘n’ number of input lines and ‘m’ number of output lines.
- ✓ An encoder produces an  $m$  bit binary code corresponding to the digital input number.
- ✓ The encoder accepts an  $n$  input digital word and converts it into  $m$  bit another digital word

# Encoder

---



# Types of Encoders

---

- ✓ Priority Encoder
- ✓ Decimal to BCD Encoder
- ✓ Octal to BCD Encoder
- ✓ Hexadecimal to Binary Encoder

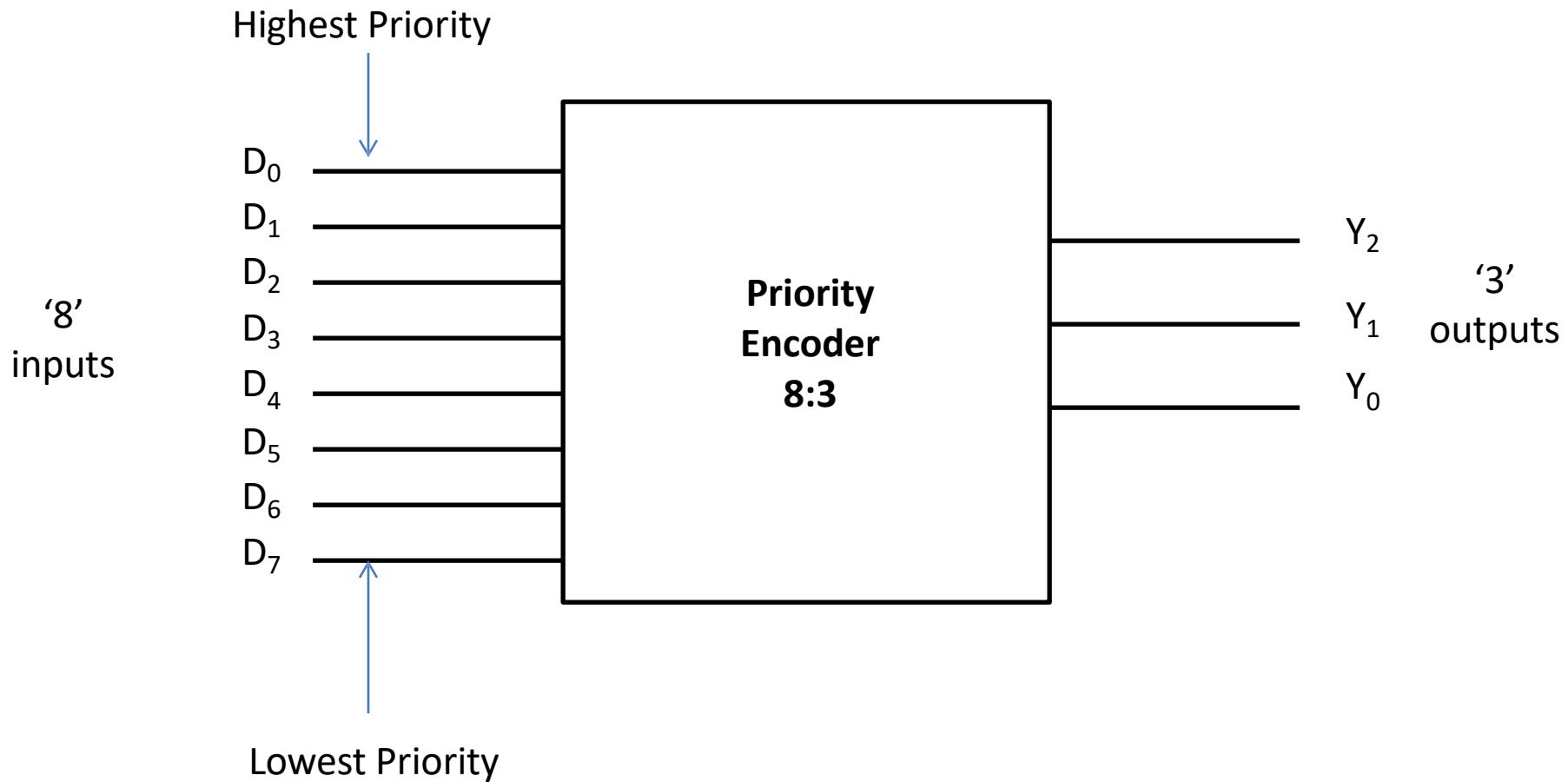
# Priority Encoder

---

- ✓ This is a special type of encoder.
- ✓ Priorities are given to the input lines.
- ✓ If two or more input lines are “1” at the same time, then the input line with highest priority will be considered.

# Priority Encoder 8:3

---



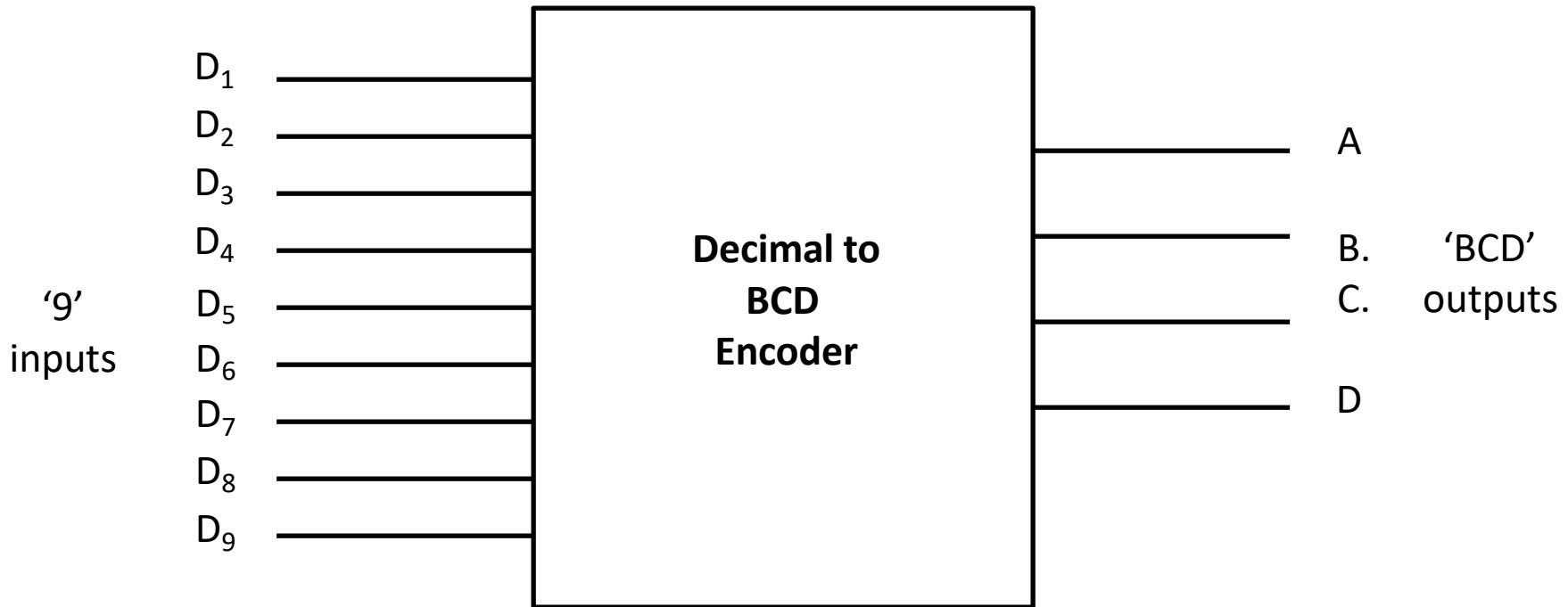
# Priority Encoder 8:3

Truth Table:

Inputs								Outputs		
$D_7$	$D_6$	$D_5$	$D_4$	$D_3$	$D_2$	$D_1$	$D_0$	$Y_2$	$Y_1$	$Y_0$
0	0	0	0	0	0	0	0	X	X	X
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

# Decimal to BCD Encoder

---



# Decimal to BCD Encoder

Truth Table:

Inputs										Outputs			
D <sub>9</sub>	D <sub>8</sub>	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>		D	C	B	A
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	X	0	0	1	0	0
0	0	0	0	0	0	1	X	X	0	0	1	1	1
0	0	0	0	0	1	X	X	X	0	1	0	0	0
0	0	0	0	1	X	X	X	X	0	1	0	0	1
0	0	0	1	X	X	X	X	X	0	1	1	0	0
0	0	1	X	X	X	X	X	X	0	1	1	1	1
0	1	X	X	X	X	X	X	X	1	0	0	0	0
1	X	X	X	X	X	X	X	X	1	0	0	0	1

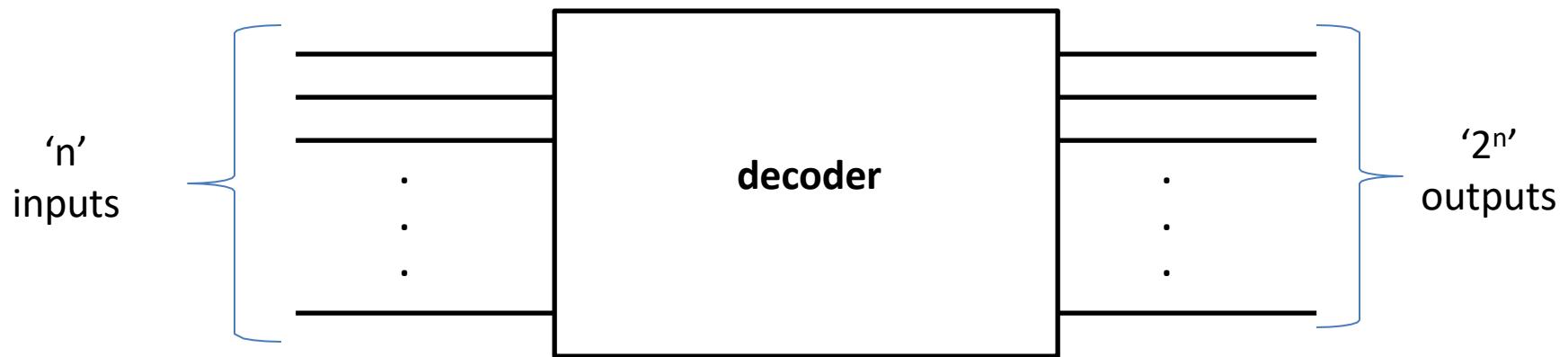
## Decoder

---

- ✓ Decoder is a combinational circuit which is designed to perform the inverse operation of encoder.
- ✓ An decoder has ‘n’ number of input lines and maximum ‘ $2^n$ ’ number of output lines.
- ✓ Decoder is identical to a demultiplexer without data input.

# Decoder

---



# Typical applications of Decoders

---

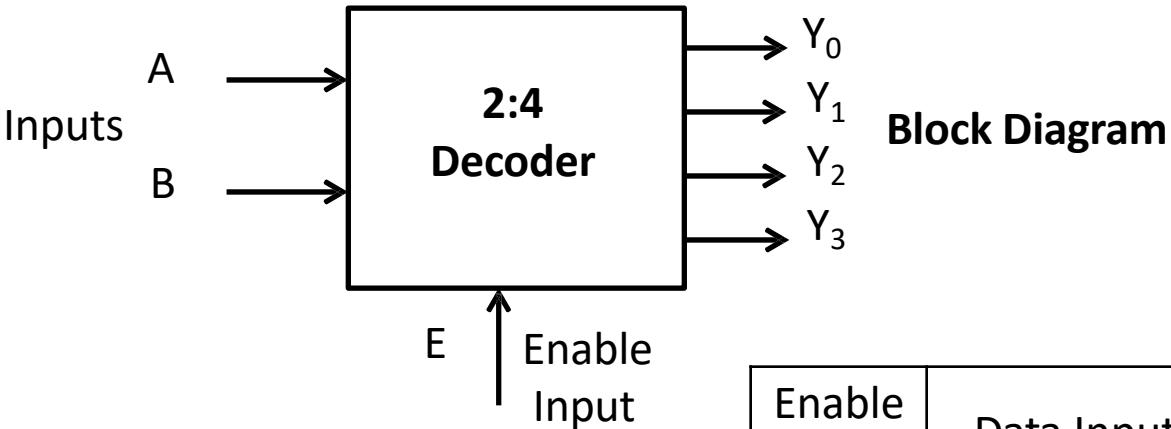
- ✓ Code Converters
- ✓ BCD to 7 segment decoders
- ✓ Nixie tube decoders
- ✓ Relay actuators

# Types of Decoders

---

- ✓ 2 to 4 line Decoder
- ✓ 3 to 8 line Decoder
- ✓ BCD to 7 Segment Decoder

# 2 to 4 Line Decoder

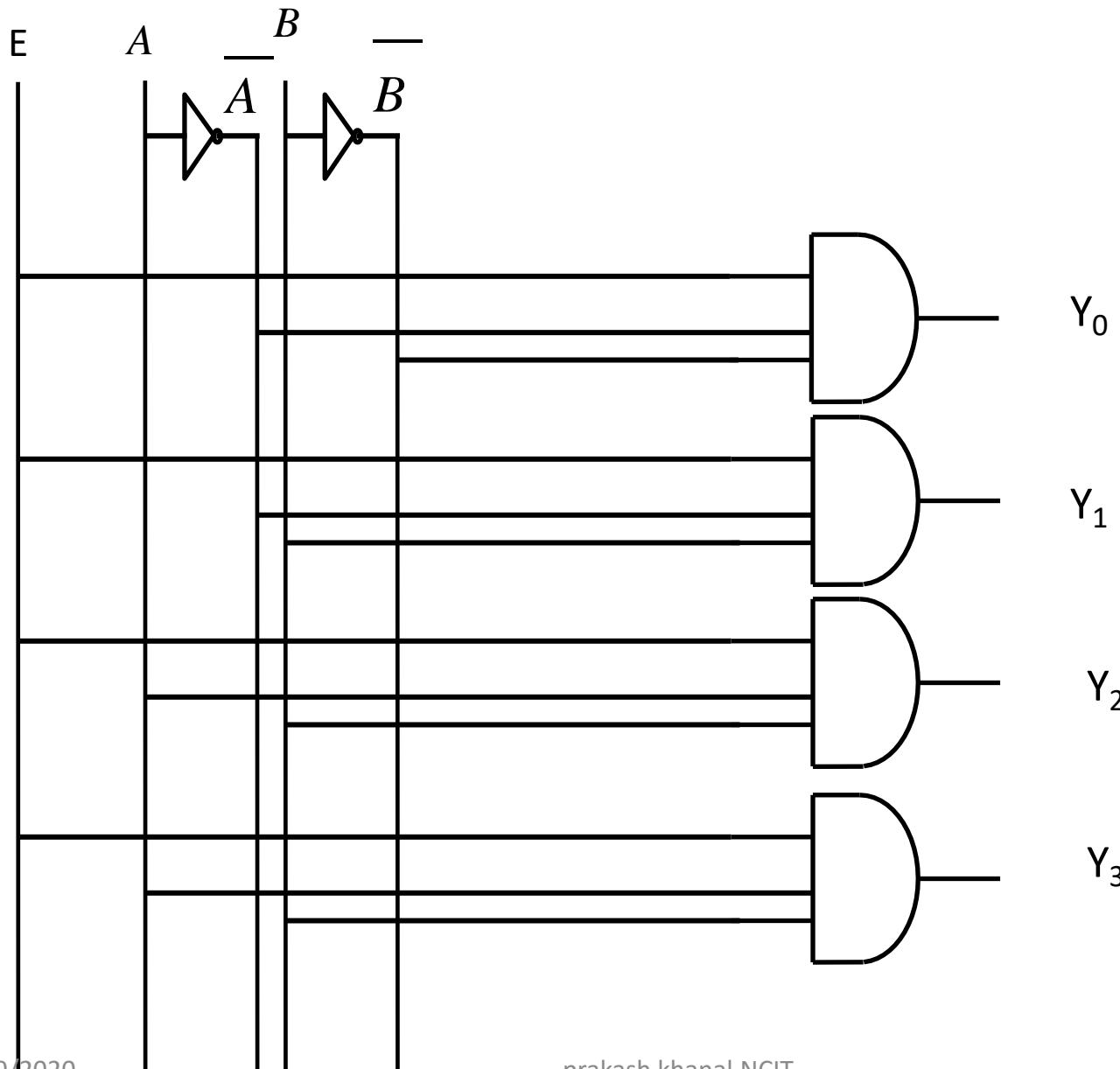


Truth Table

Enable i/p	Data Inputs		Outputs				
	E	A	B	Y <sub>0</sub>	Y <sub>1</sub>	Y <sub>2</sub>	Y <sub>3</sub>
0	X	X		0	0	0	0
1	0	0		1	0	0	0
1	0	1		0	1	0	0
1	1	0		0	0	1	0
1	1	1		0	0	0	1

# 2 to 4 Line Decoder

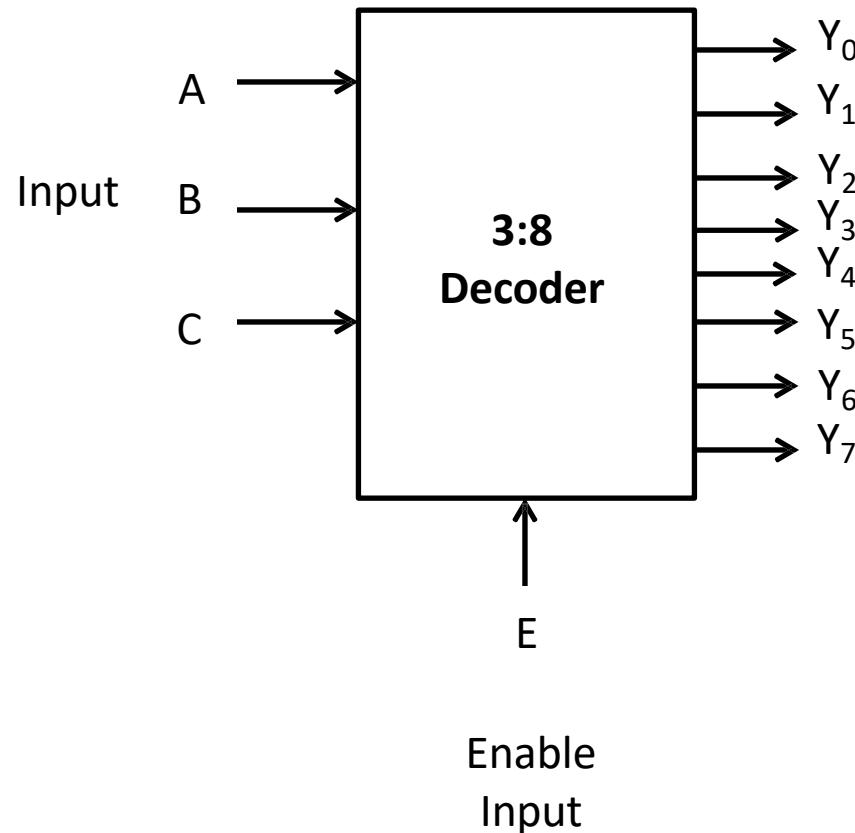
---



# 3 to 8 Line Decoder

---

Block Diagram



# 3 to 8 Line Decoder

Truth Table

Enabl e i/p	Inputs			Outputs								
	E	A	B	C	Y <sub>7</sub>	Y <sub>6</sub>	Y <sub>5</sub>	Y <sub>4</sub>	Y <sub>3</sub>	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>
0	X	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	0	1	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
1 11/29/2021	1	1	1	1	prakash khanal, NCIT	0	0	0	0	0	0	27 0

# Implement a full adder using decoder

## Example: Implement full adder using decoder.

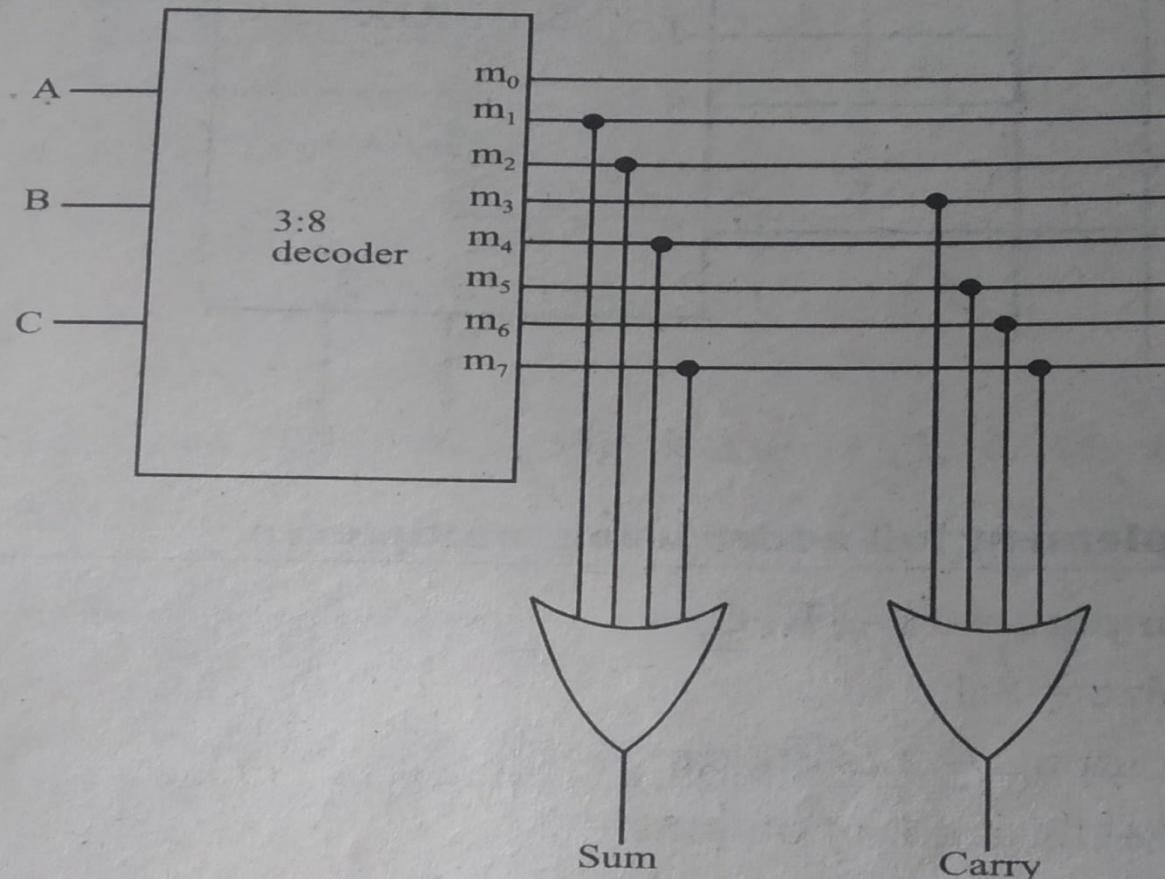
⇒ For full adder, no. of inputs = 3, (A, B, C)

Decoder size = 3:8

No. of outputs = 2 (sum, carry)

Sum =  $\Sigma m(1, 2, 4, 7)$

Carry =  $\Sigma m(3, 5, 6, 7)$



# Implement the following Boolean function using decoder

$$F_1 = \sum m(0, 4, 6)$$

$$F_2 = \sum m(0, 5)$$

$F_3 = \sum m(1, 2, 3, 7)$  where F is a function of A, B, and C

$$Y_1 = 2^0 \cdot 1 + 2^1 \cdot 0 + 2^2 \cdot 1$$

**Example: Implement the following Boolean expression using decoder.**

$$F_1 = \Sigma m(0, 4, 6)$$

$$F_2 = \Sigma m(0, 5)$$

$F_3 = \Sigma m(1, 2, 3, 7)$  where F is a function of A, B, and C

$\Rightarrow$  No. of inputs = 3

Decoder size =  $3:2^3 = 3:8$

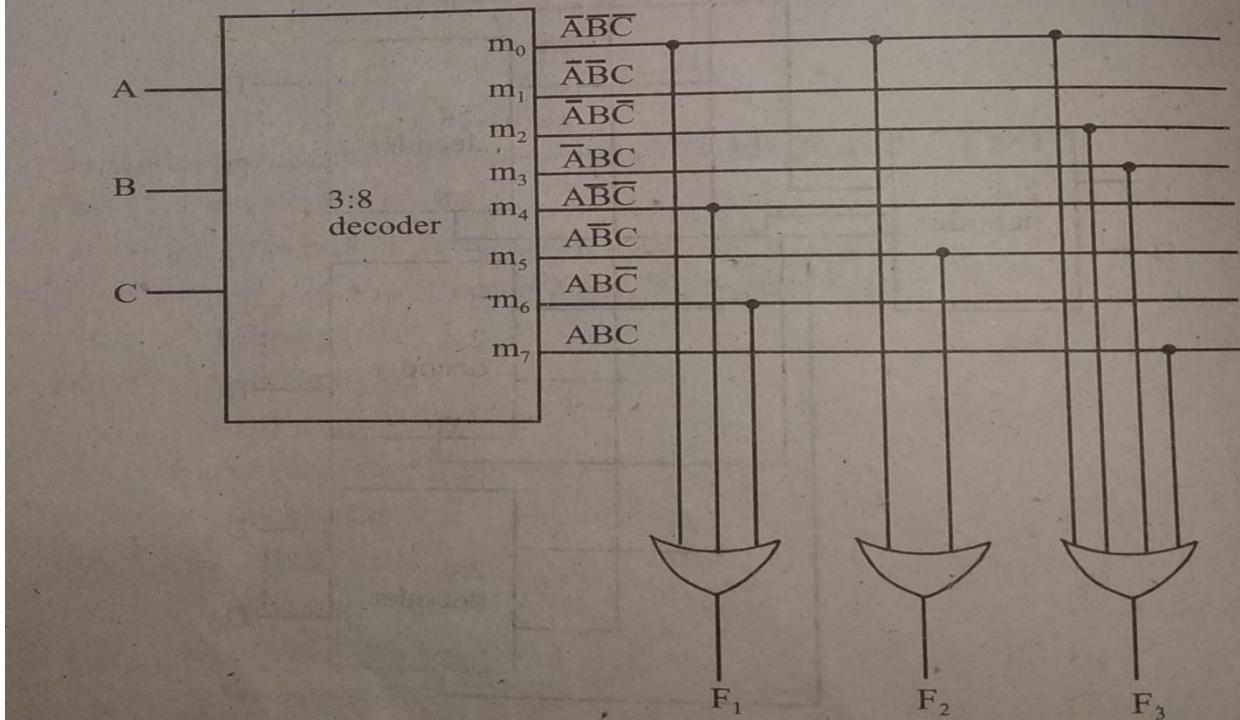
No. of outputs = 3 ( $F_1, F_2, F_3$ )

No. of OR gates = 3

$$F_1(A, B, C) = \Sigma m(0, 4, 6)$$

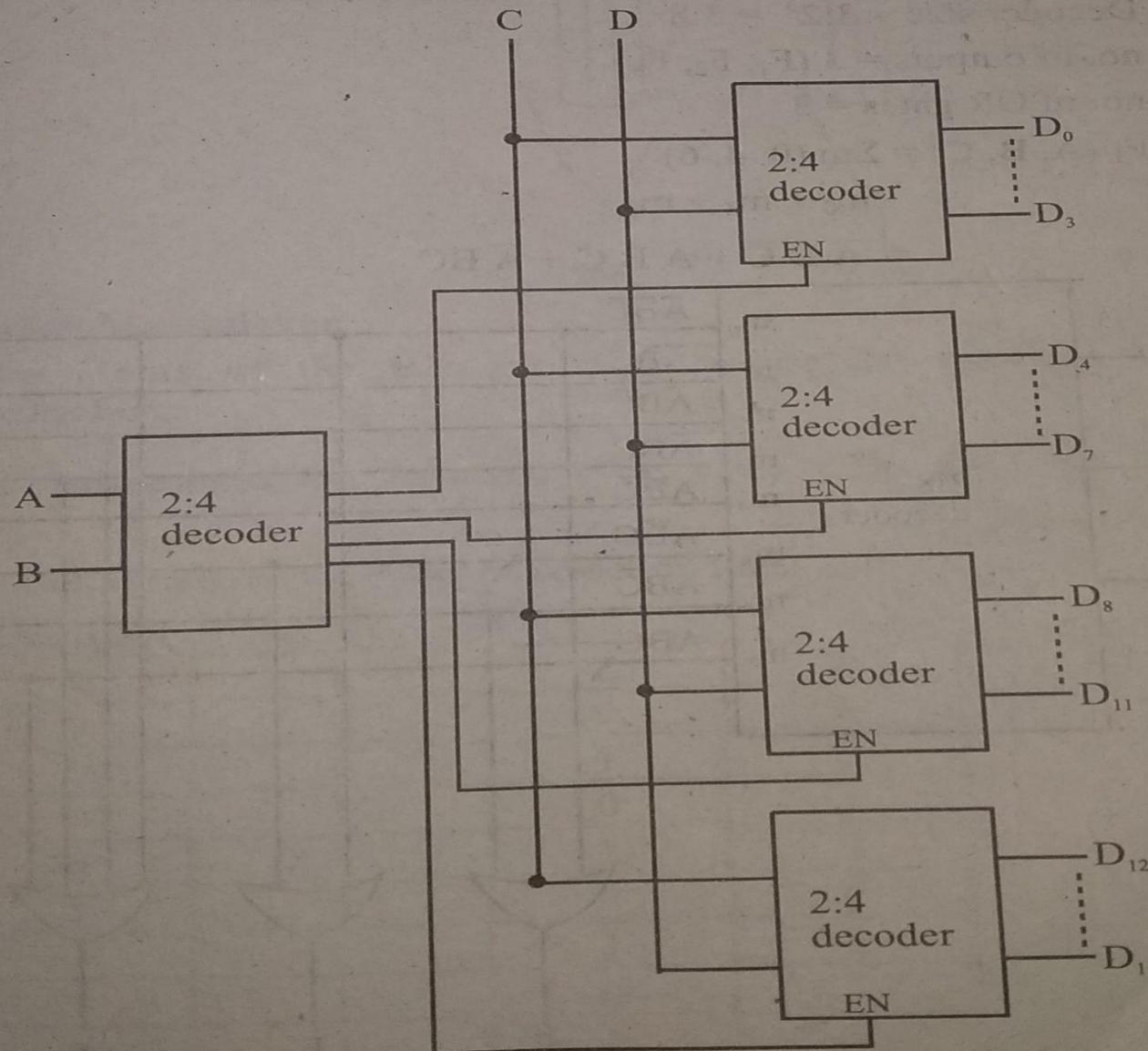
$$= m_0 + m_4 + m_6$$

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



# Construct 4:16 line decoder with five 2:4 decoder with enable

## Construction of 4:16 line decoder with five 2:4 line decoder with enable:

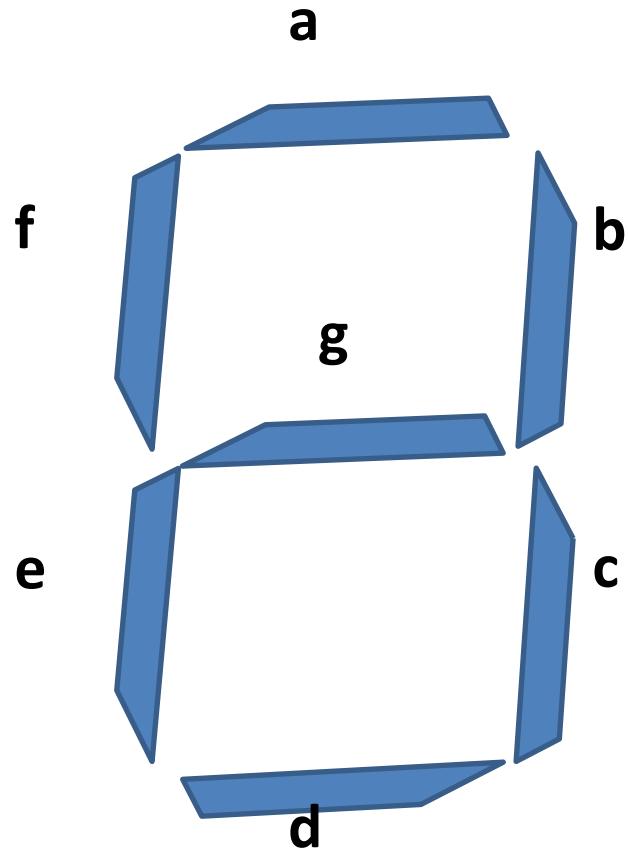


# Comparison between Encoder & Decoder

Sr. No.	Parameter	Encoder	Decoder
1	Input applied	Active input signal (original message signal)	Coded binary input
2	Output generated	Coded binary output	Active output signal (original message)
3	Input lines	$2^n$	n
4	Output lines	N	$2^n$
5	Operation	Simple	Complex
6	Applications	E-mail , video encoders etc.	Microprocessors, memory chips etc.

# BCD to 7 Segment Decoder - Seven Segment Display

---

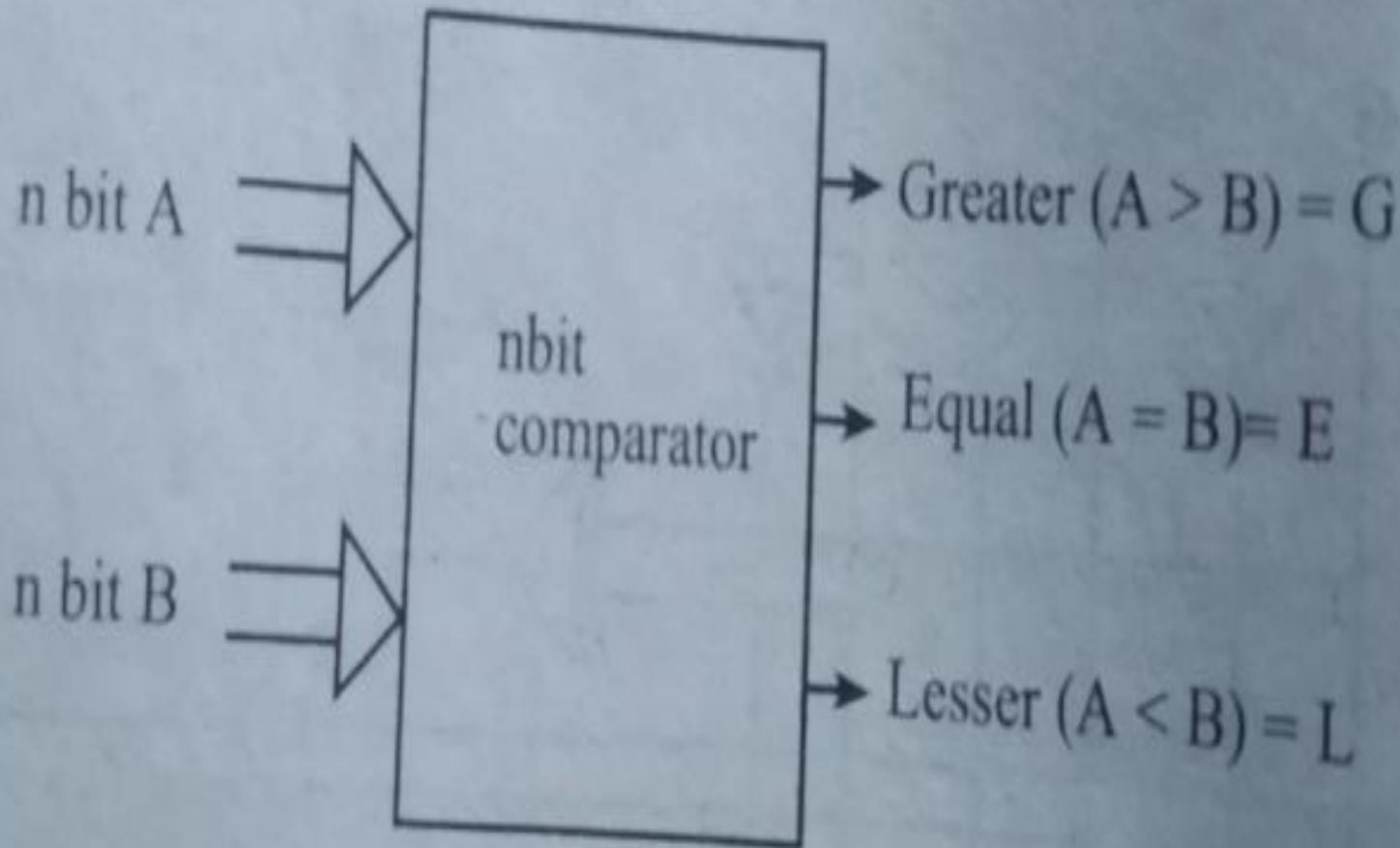


# Seven Segment Display

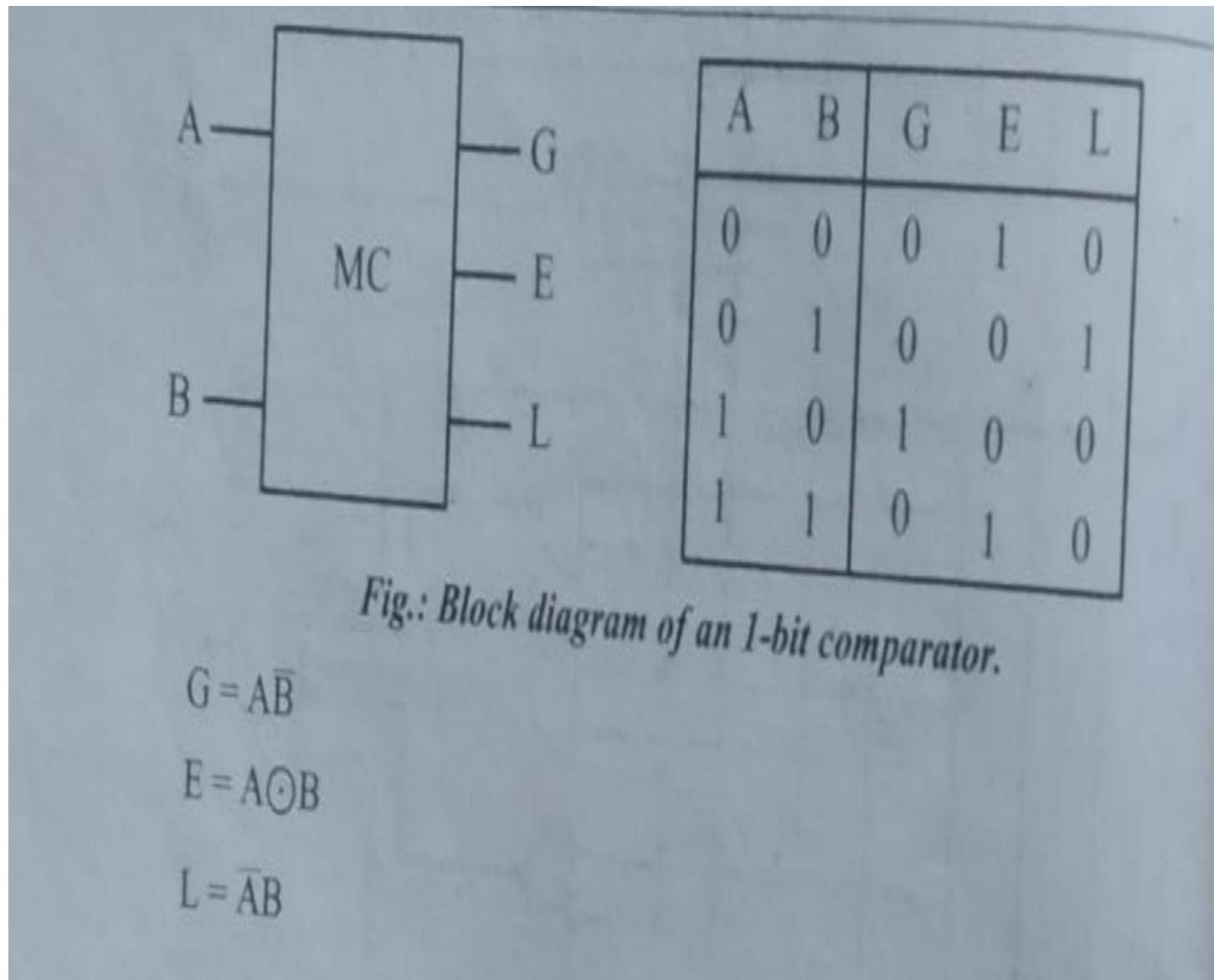
Segments							Display Number	Seven Segment Display
a	b	c	d	e	f	g		
ON	ON	ON	ON	ON	ON	OFF	0	
OFF	ON	ON	OFF	OFF	OFF	OFF	1	
ON	ON	OFF	ON	ON	OFF	ON	2	
ON	ON	ON	ON	OFF	OFF	ON	3	
OFF	ON	ON	OFF	OFF	ON	ON	4	
ON	OFF	ON	ON	OFF	ON	ON	5	
ON	OFF	ON	ON	ON	ON	ON	6	
ON	ON	ON	OFF	OFF	OFF	OFF	7	
ON	ON	ON	ON	ON	ON	ON	8	
ON	ON	ON	ON	OFF	ON	ON	9	

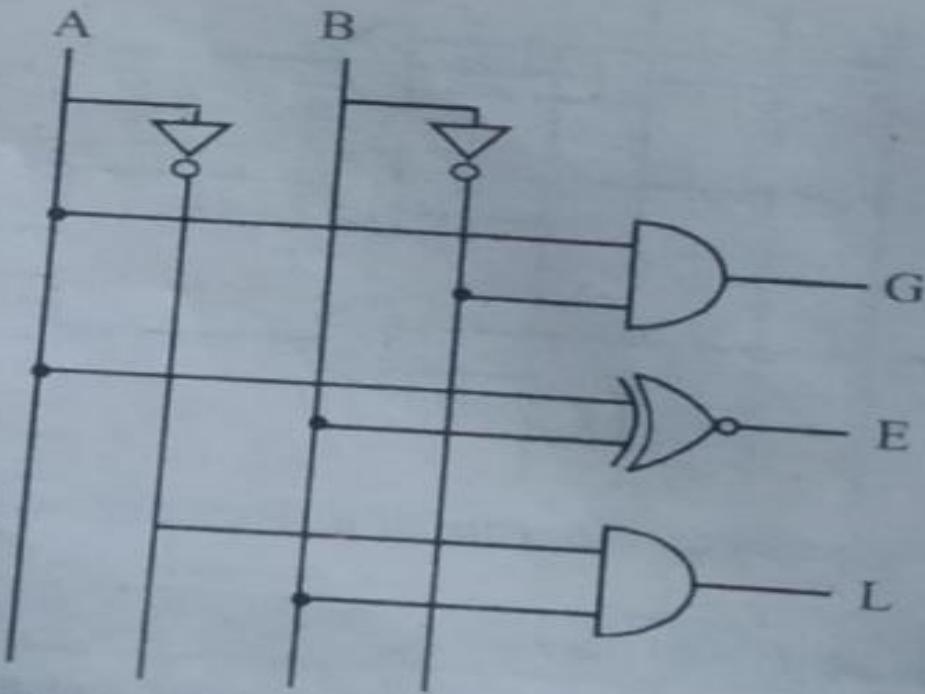
# Magnitude comparator:

- Magnitude comparator is a combinational circuit, designed to compare two n bits words applied as its input.
- The comparator has three output namely greater( $A > B$ ), lesser( $A < B$ ) and equal ( $A = B$ ). Depending upon the result of comparison, one of these outputs will go high.



*Fig.: Block diagram of an n-bit comparator.*





*Fig.: Logic diagram of 1-bit magnitude comparator*

# Multiplexers

---

- ✓ Multiplexer is a circuit which has a number of inputs but only one output.
- ✓ Multiplexer is a circuit which transmits large number of information signals over a single line.
- ✓ Multiplexer is also known as “Data Selector” or MUX.

## Necessity of Multiplexers

---

- ✓ In most of the electronic systems, the digital data is available on more than one lines. It is necessary to route this data over a single line.
- ✓ Under such circumstances we require a circuit which select one of the many inputs at a time.
- ✓ This circuit is nothing but a multiplexer. Which has many inputs, one output and some select lines.
- ✓ Multiplexer improves the reliability of the digital system because it reduces the number of external wired connections.

## Advantages of Multiplexers

---

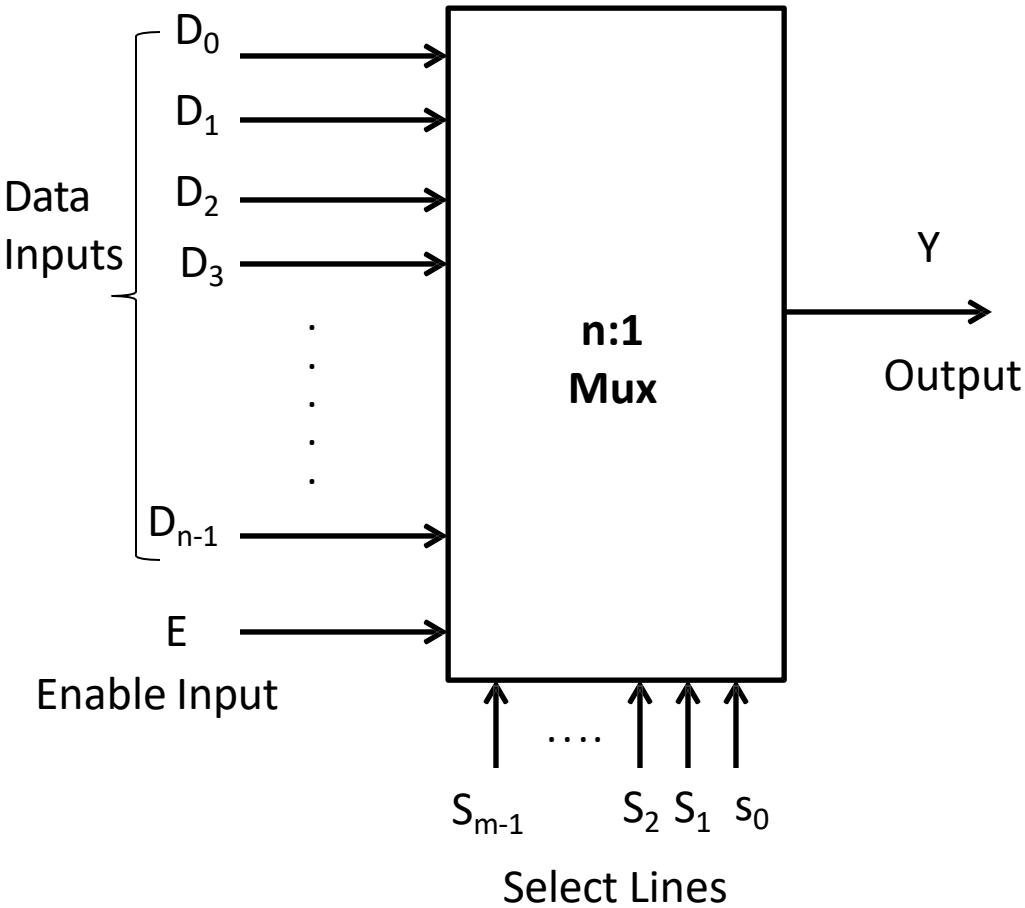
- ✓ It reduces the number of wires.
- ✓ So it reduces the circuit complexity and cost.
- ✓ We can implement many combinational circuits using Mux.
- ✓ It simplifies the logic design.
- ✓ It does not need the k-map and simplification.

# Applications of Multiplexers

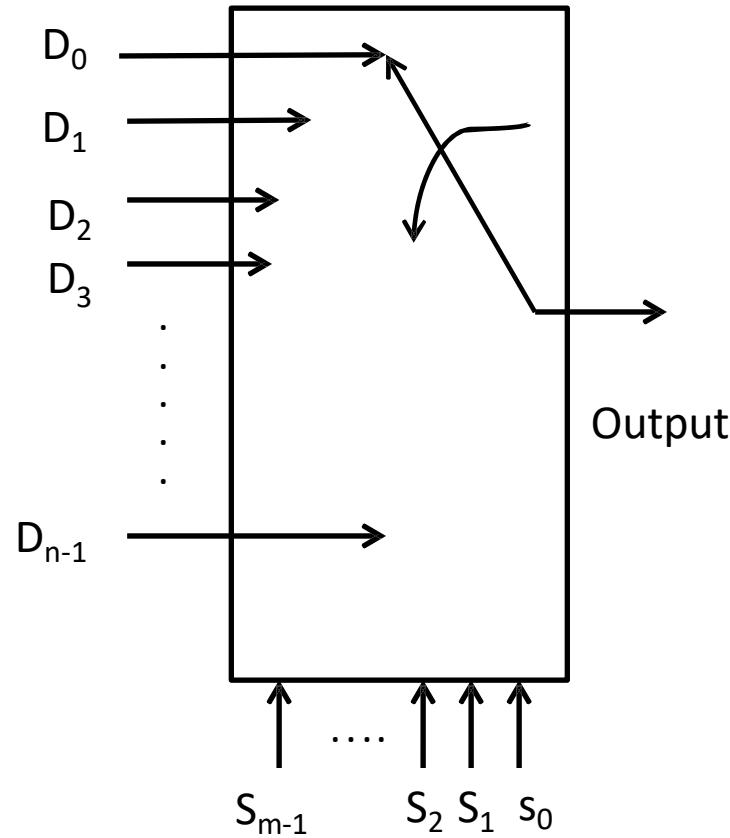
---

- ✓ It is used as a data selector to select one out of many data inputs.
- ✓ It is used for simplification of logic design.
- ✓ It is used in data acquisition system.
- ✓ In designing the combinational circuits.
- ✓ In D to A converters.
- ✓ To minimize the number of connections.

# Block Diagram of Multiplexer



**Fig. General Block Diagram**



**Fig. Equivalent Circuit**

## Relation between Data Input Lines & Select Lines

---

- ✓ In general multiplexer contains , n data lines, one output line and m select lines.
- ✓ To select n inputs we need m select lines such that  $2^m=n$ .

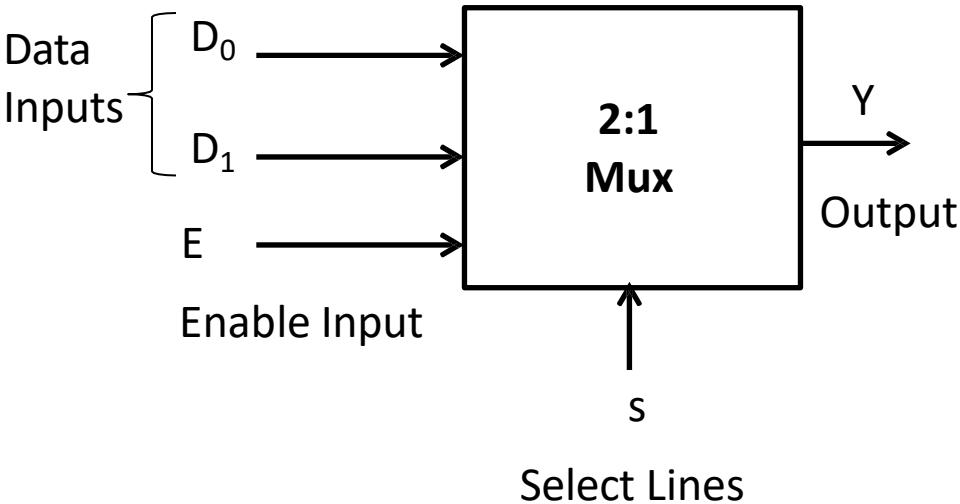
# Types of Multiplexers

---

- ✓ 2:1 Multiplexer
- ✓ 4:1 Multiplexer
- ✓ 8:1 Multiplexer
- ✓ 16:1 Multiplexer
- ✓ 32:1 Multiplexer
- ✓ 64:1 Multiplexer

and so on.....

# 2:1 Multiplexer



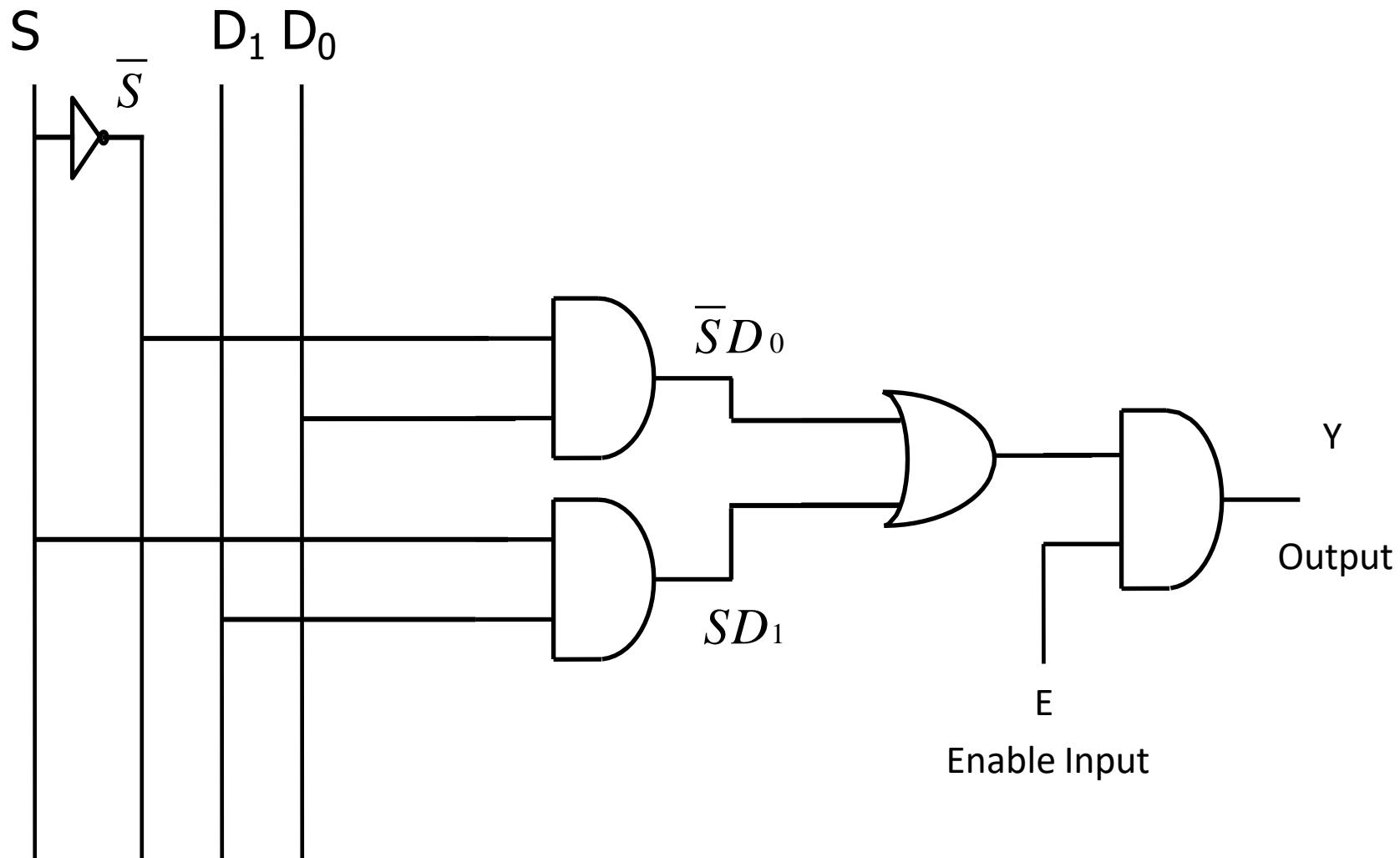
Block Diagram

Enable i/p (E)	Select i/p (S)	Output (Y)
0	X	0
1	0	$D_0$
1	1	$D_1$

Truth Table

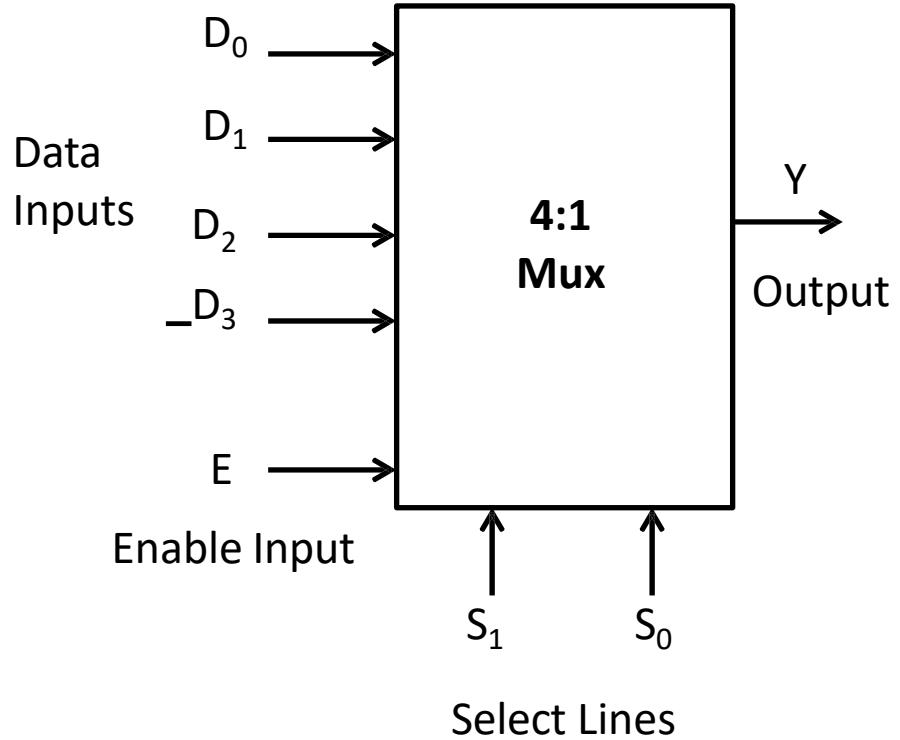
# Realization of 2:1 Mux using gates

---



# 4:1 Multiplexer

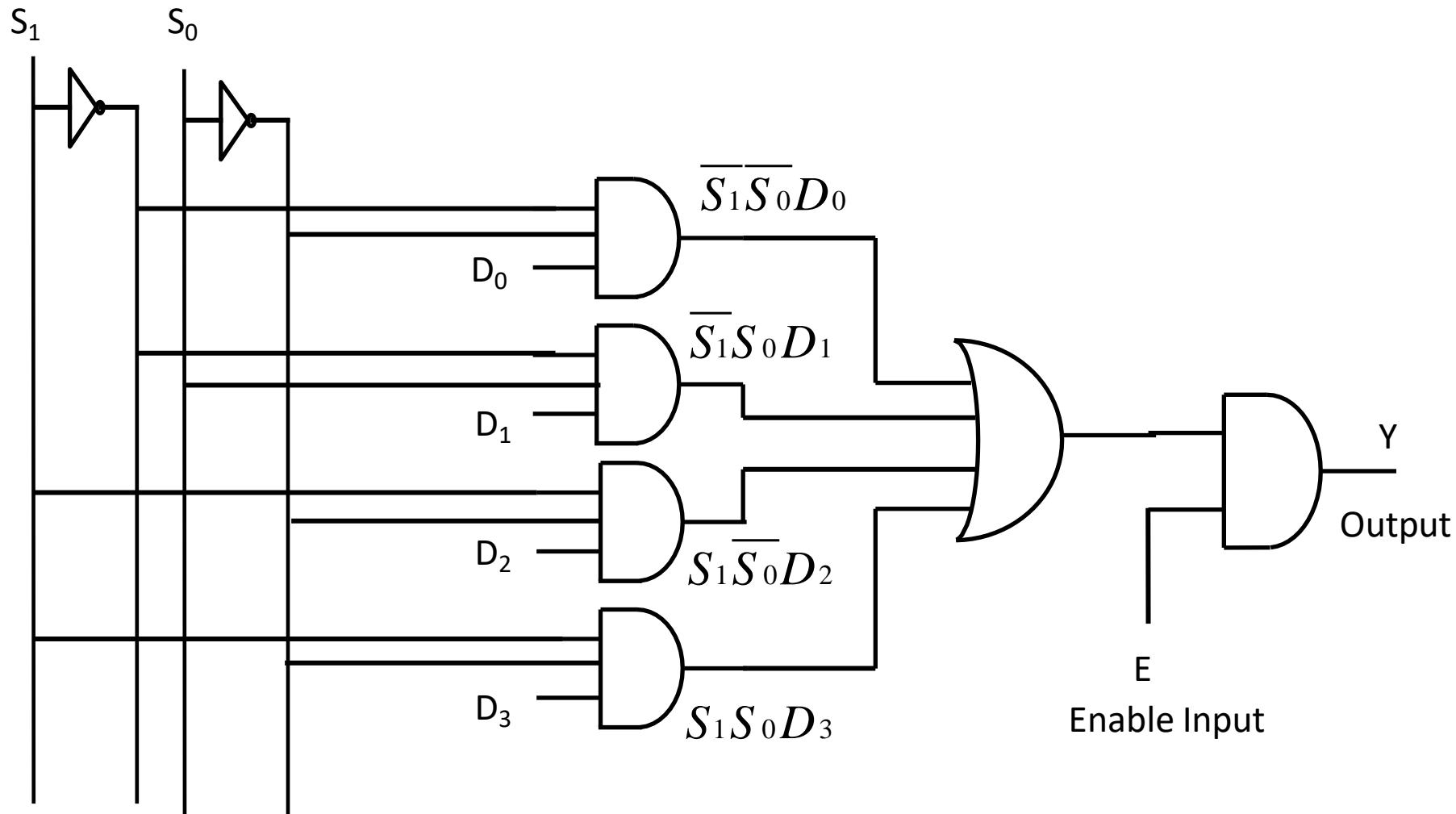
Block Diagram



Truth Table

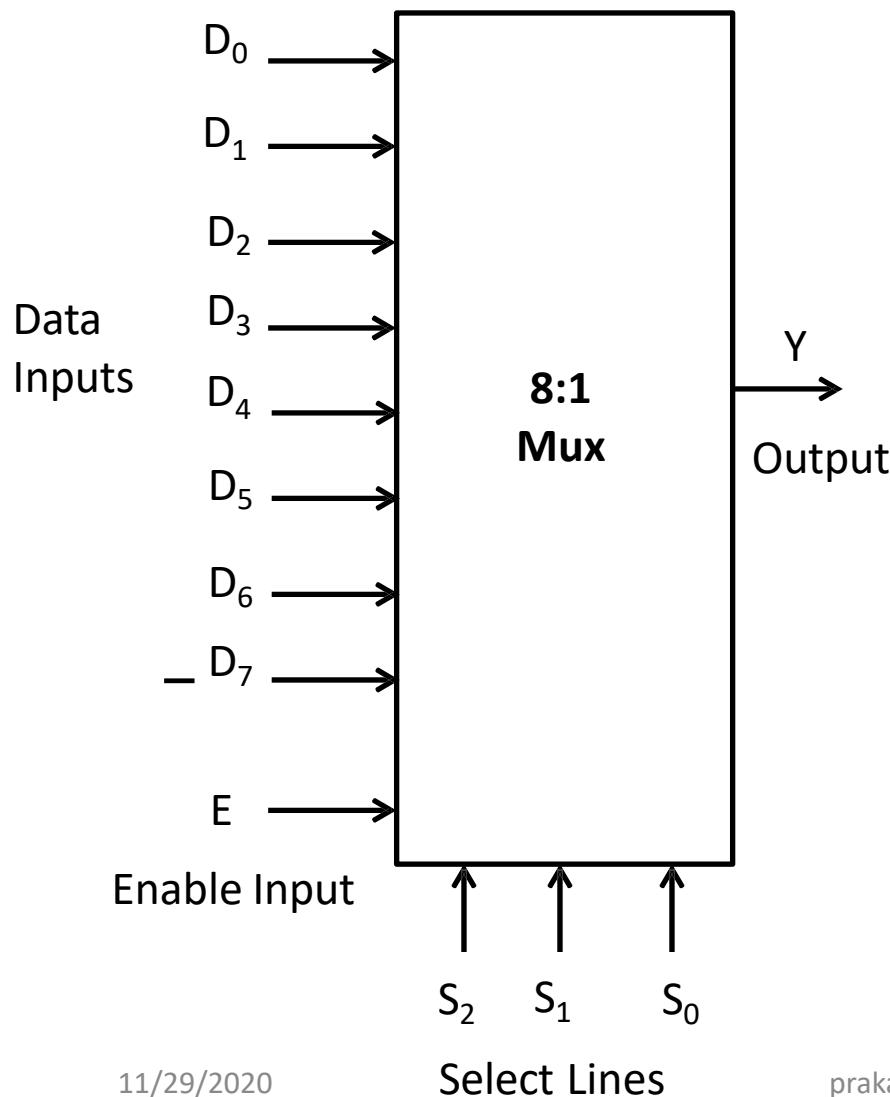
Enable i/p	Select i/p		Output
$E$	$S_1$	$S_0$	$Y$
0	X	X	0
1	0	0	$D_0$
1	0	1	$D_1$
1	1	0	$D_2$
1	1	1	$D_3$

# Realization of 4:1 Mux using gates



# 8:1 Multiplexer

Block Diagram

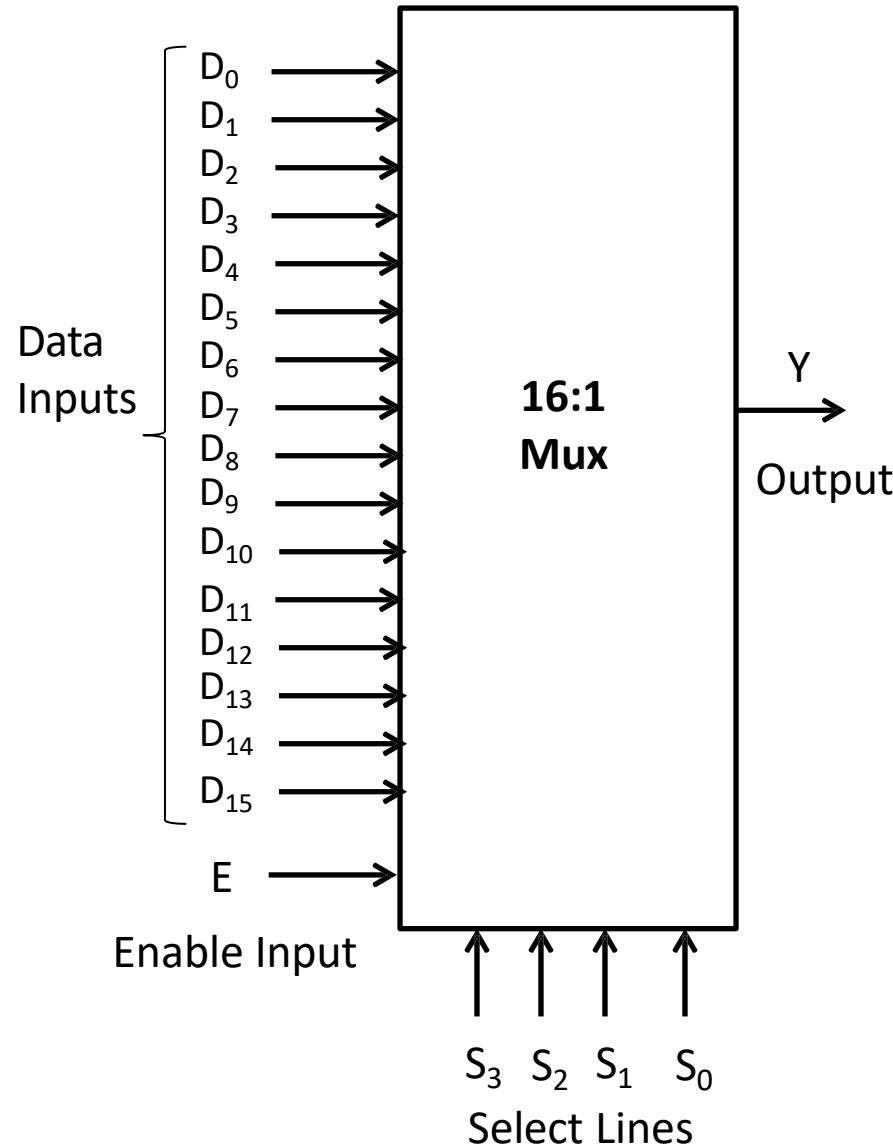


Truth Table

Enable i/p	Select i/p			Output
E	$S_2$	$S_1$	$S_0$	Y
0	X	X	X	0
1	0	0	0	$D_0$
1	0	0	1	$D_1$
1	0	1	0	$D_2$
1	0	1	1	$D_3$
1	1	0	0	$D_4$
1	1	0	1	$D_5$
1	1	1	0	$D_6$
1	1	1	1	$D_7$

# 16:1 Multiplexer

Block Diagram



# 16:1 Multiplexer

Truth Table

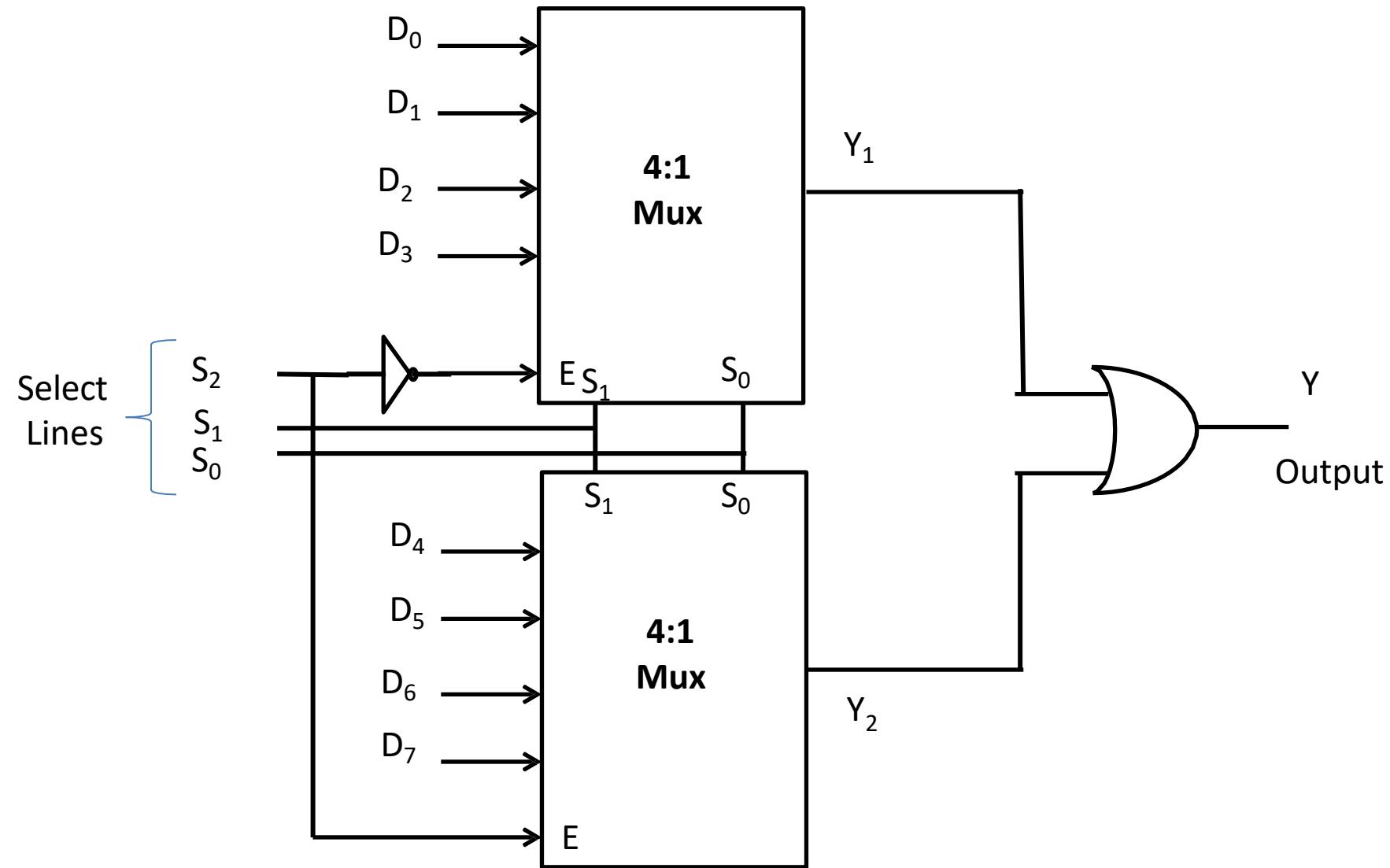
Enable	Select Lines				Output Y
	$S_3$	$S_2$	$S_1$	$S_0$	
0	X	X	X	X	0
1	0	0	0	0	$D_0$
1	0	0	0	1	$D_1$
1	0	0	1	0	$D_2$
1	0	0	1	1	$D_3$
1	0	1	0	0	$D_4$
1	0	1	0	1	$D_5$
1	0	1	1	0	$D_6$
1	0	1	1	1	$D_7$
1	1	0	0	0	$D_8$
1	1	0	0	1	$D_9$
1	1	0	1	0	$D_{10}$
1	1	0	1	1	$D_{11}$
1	1	1	0	0	$D_{12}$
1	1	1	0	1	$D_{13}$
1	1	1	1	0	$D_{14}$
1	1	1	1	1	$D_{15}$

## Mux Tree

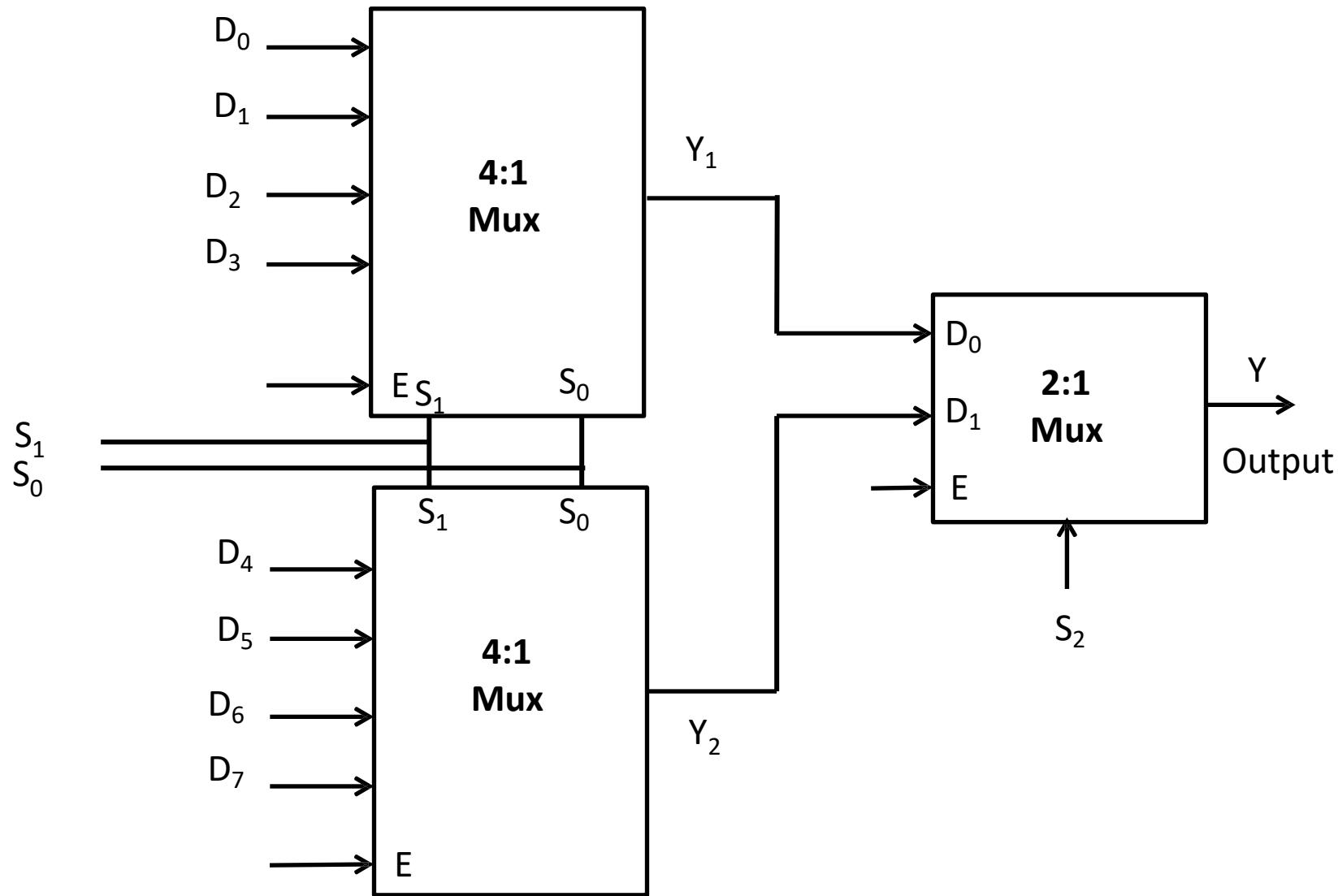
---

- ✓ The multiplexers having more number of inputs can be obtained by cascading two or more multiplexers with less number of inputs. This is called as Multiplexer Tree.
- ✓ For example, 32:1 mux can be realized using two 16:1 mux and one 2:1 mux.

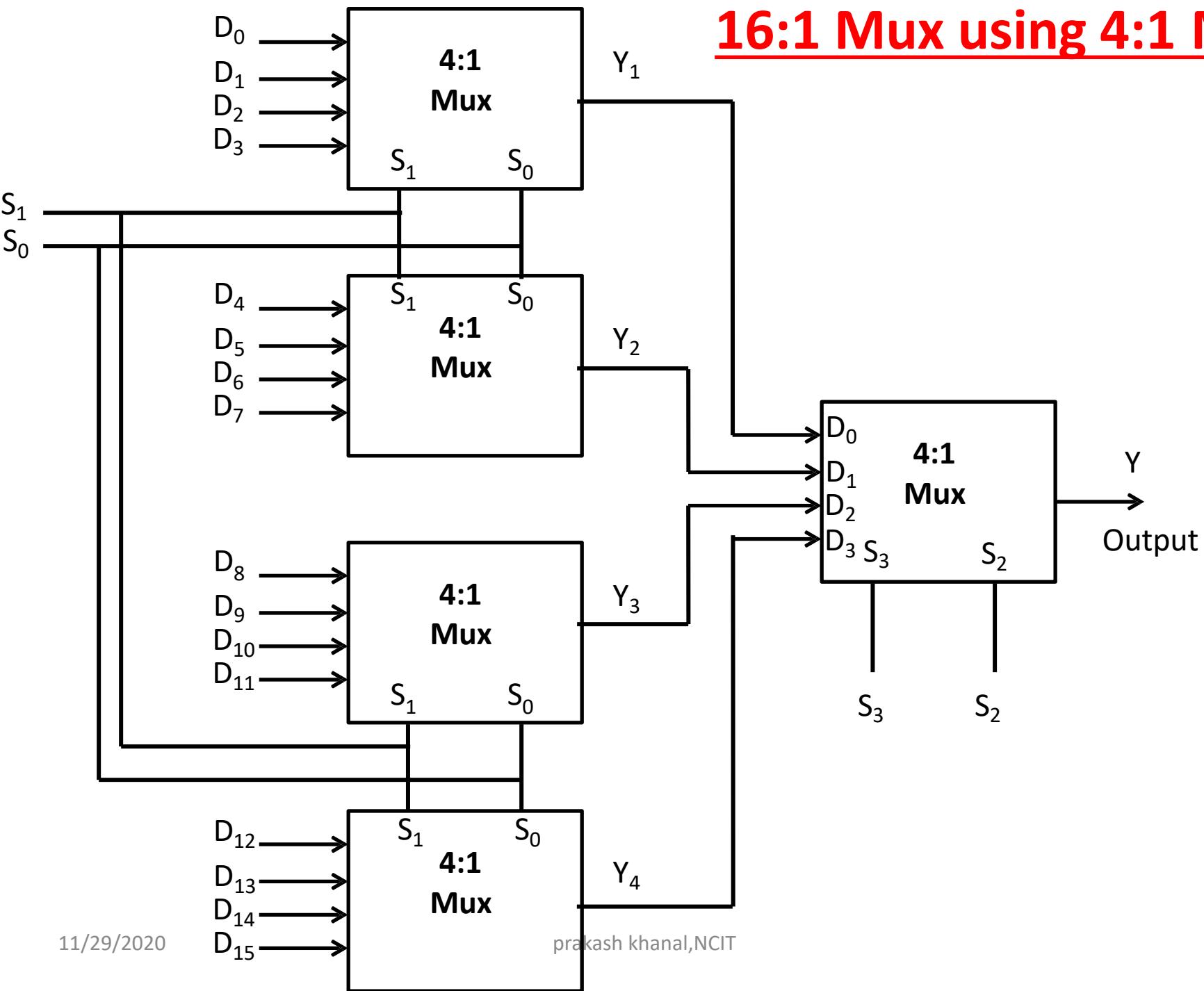
# 8:1 Multiplexer using 4:1 Multiplexer



# 8:1 Multiplexer using 4:1 Multiplexer



# 16:1 Mux using 4:1 Mux



# Realization of Boolean expression using Mux

---

- ✓ We can implement any Boolean expression using Multiplexers.
- ✓ It reduces circuit complexity.
- ✓ It does not require any simplification

## Example 1

---

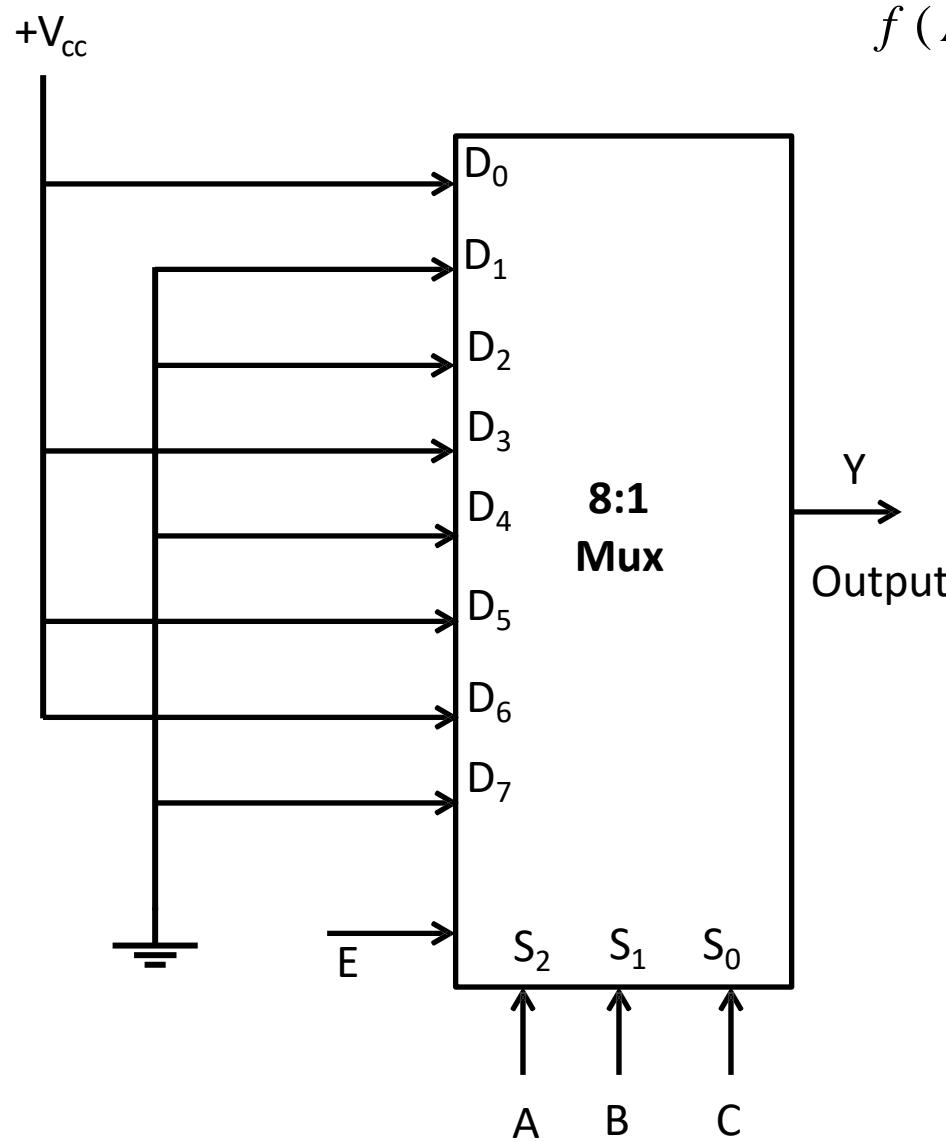
Implement following Boolean expression using multiplexer

$$f(A, B, C) = \sum m(0, 3, 5, 6)$$

- ✓ Since there are three variables, therefore a multiplexer with three select input is required i.e. 8:1 multiplexer is required
- ✓ The 8:1 multiplexer is configured as below to implement given Boolean expression

# Example 1

continue.....



## Example 2

---

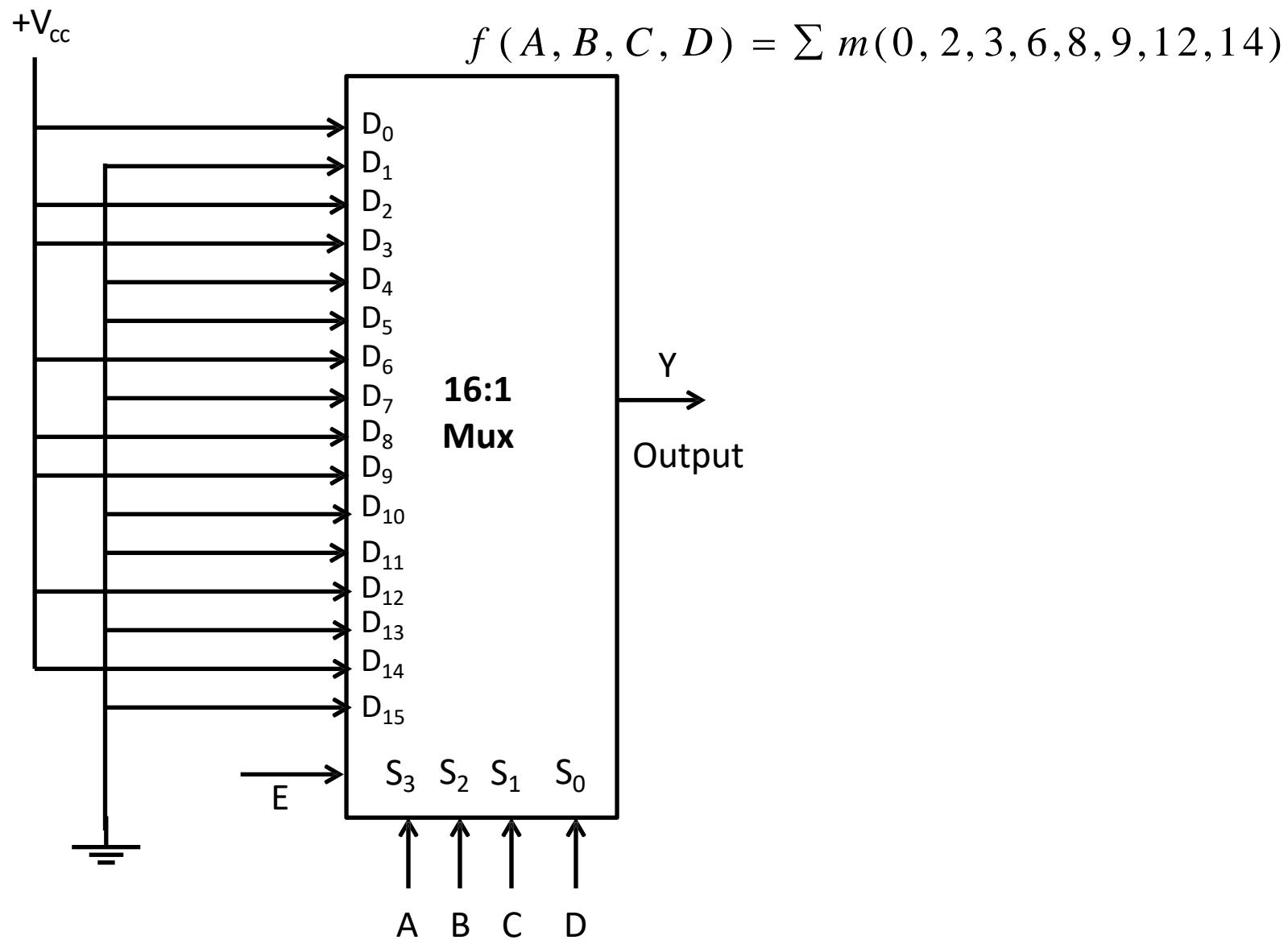
Implement following Boolean expression using multiplexer

$$f(A, B, C, D) = \sum m(0, 2, 3, 6, 8, 9, 12, 14)$$

- ✓ Since there are four variables, therefore a multiplexer with four select input is required i.e. 16:1 multiplexer is required
- ✓ The 16:1 multiplexer is configured as below to implement given Boolean expression

## Example 2

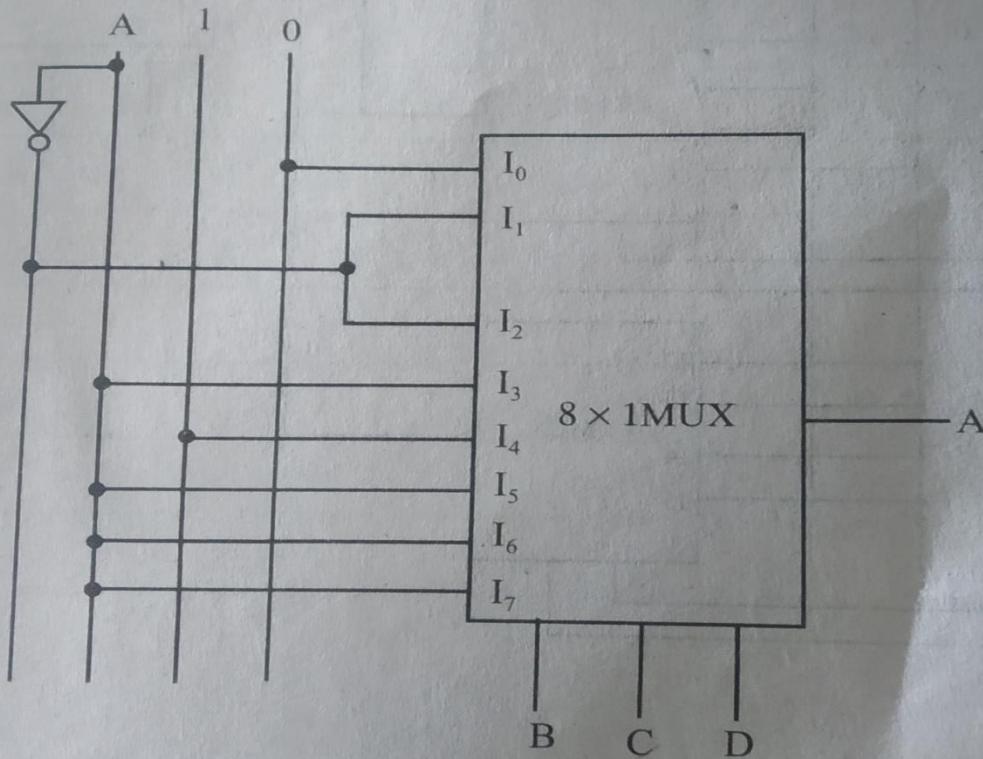
continue.....



**Example: Implement  $F(A, B, C, D) = \Sigma m(1, 2, 4, 11, 12, 13, 14, 15)$  using  $8 \times 1$  MUX.**

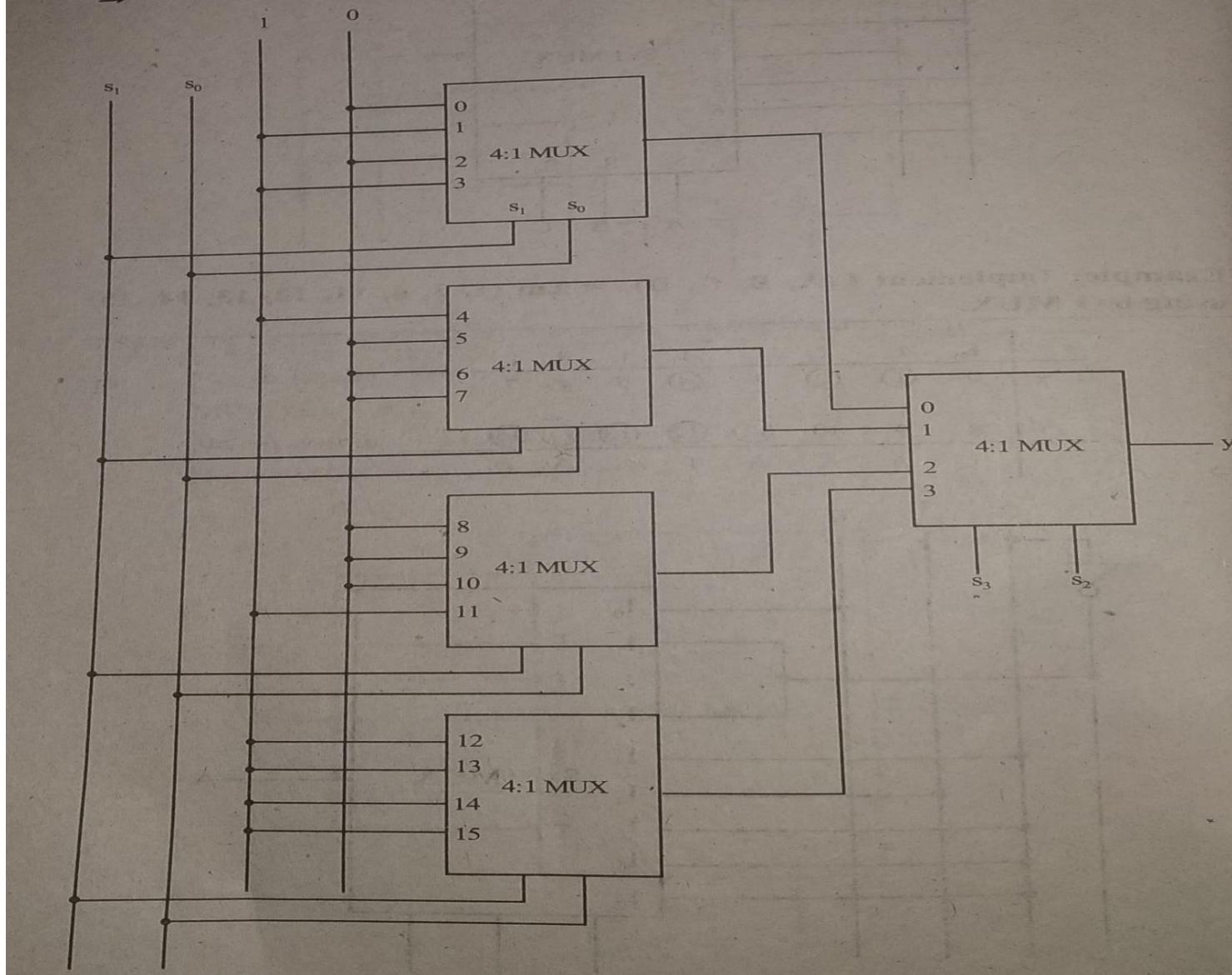
	$I_0$	$I_1$	$I_2$	$I_3$	$I_4$	$I_5$	$I_6$	$I_7$
$\bar{A}$	0	①	②	3	④	5	⑥	7
A	8	9	10	⑪	⑫	⑬	⑭	⑮
	1	$\bar{A}$	$\bar{A}$	A	1	A	A	A

$\Rightarrow$



**Example: Implement  $F(A, B, C, D) = \sum m(1, 3, 4, 11, 12, 13, 14, 15)$  using  $4 \times 1$  MUX.**

$\Rightarrow$  No. of selection lines = 4 ( $S_3, S_2, S_1, S_0$ )

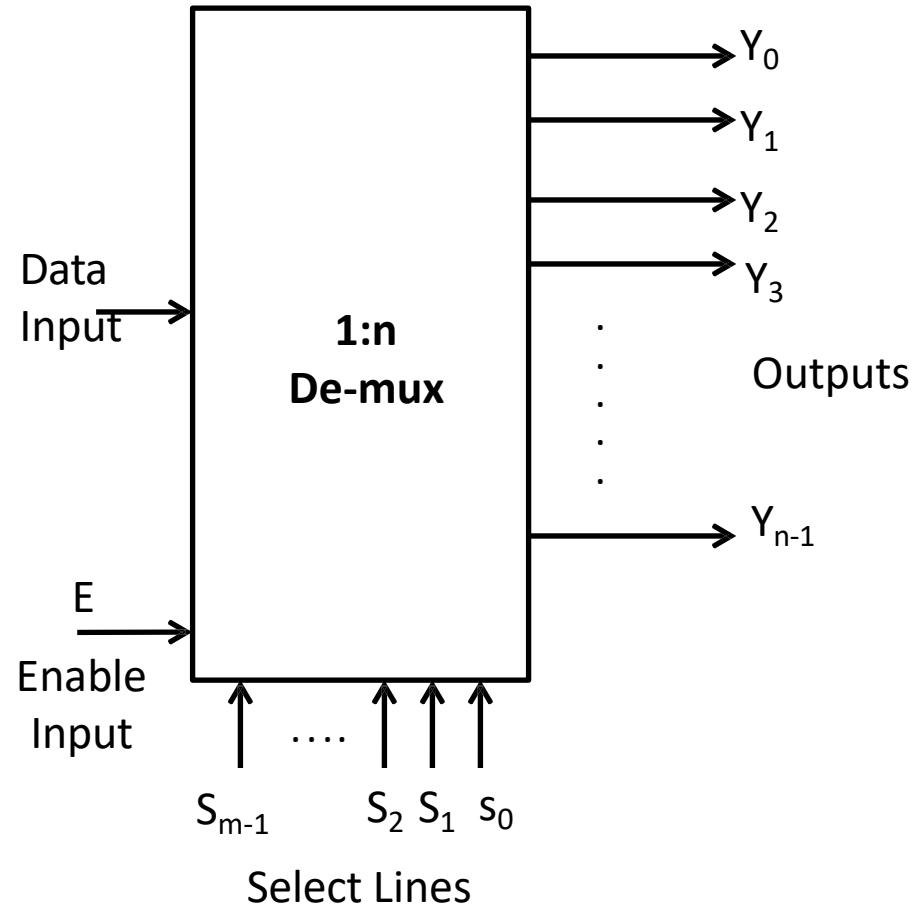


## De-multiplexer

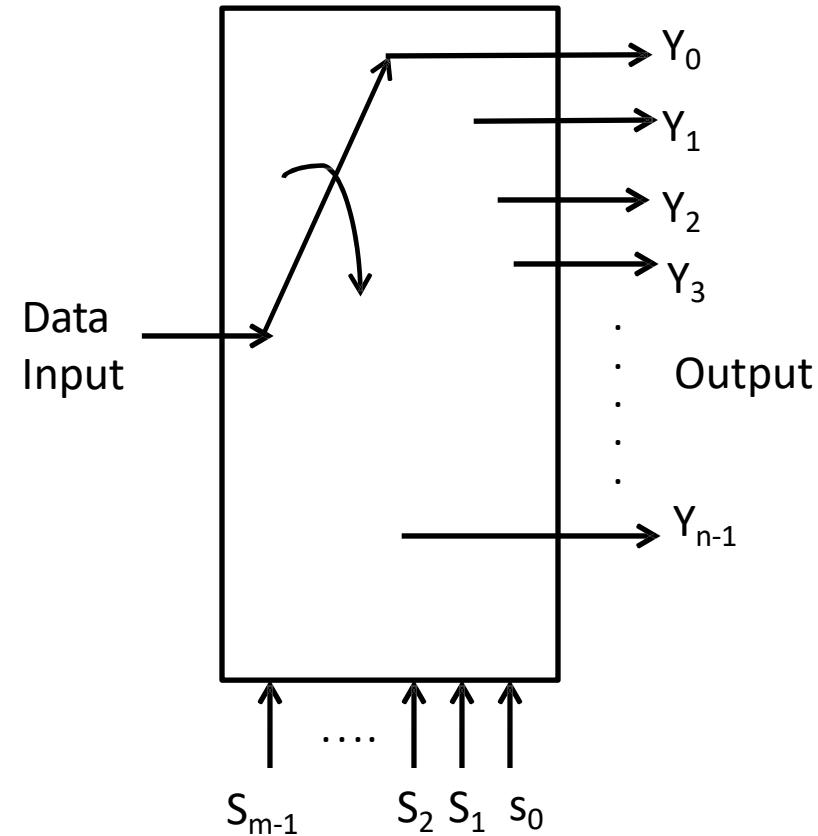
---

- ✓ A de-multiplexer performs the reverse operation of a multiplexer i.e. it receives one input and distributes it over several outputs.
- ✓ At a time only one output line is selected by the select lines and the input is transmitted to the selected output line.
- ✓ It has only one input line, n number of output lines and m number of select lines.

# Block Diagram of De-multiplexer



**Fig. General Block Diagram**



**Fig. Equivalent Circuit**

## Relation between Data Output Lines & Select Lines

---

- ✓ In general de-multiplexer contains , n output lines, one input line and m select lines.
- ✓ To select n outputs we need m select lines such that  $n=2^m$ .

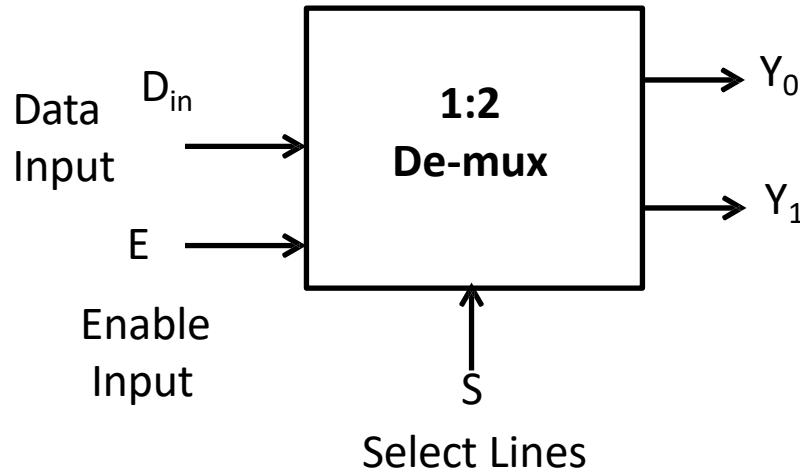
## Types of De-multiplexers

---

- ✓ 1:2 De-multiplexer
- ✓ 1:4 De-multiplexer
- ✓ 1:8 De-multiplexer
- ✓ 1:16 De-multiplexer
- ✓ 1:32 De-multiplexer
- ✓ 1:64 De-multiplexer

and so on.....

# 1: 2 De-multiplexer



Block Diagram

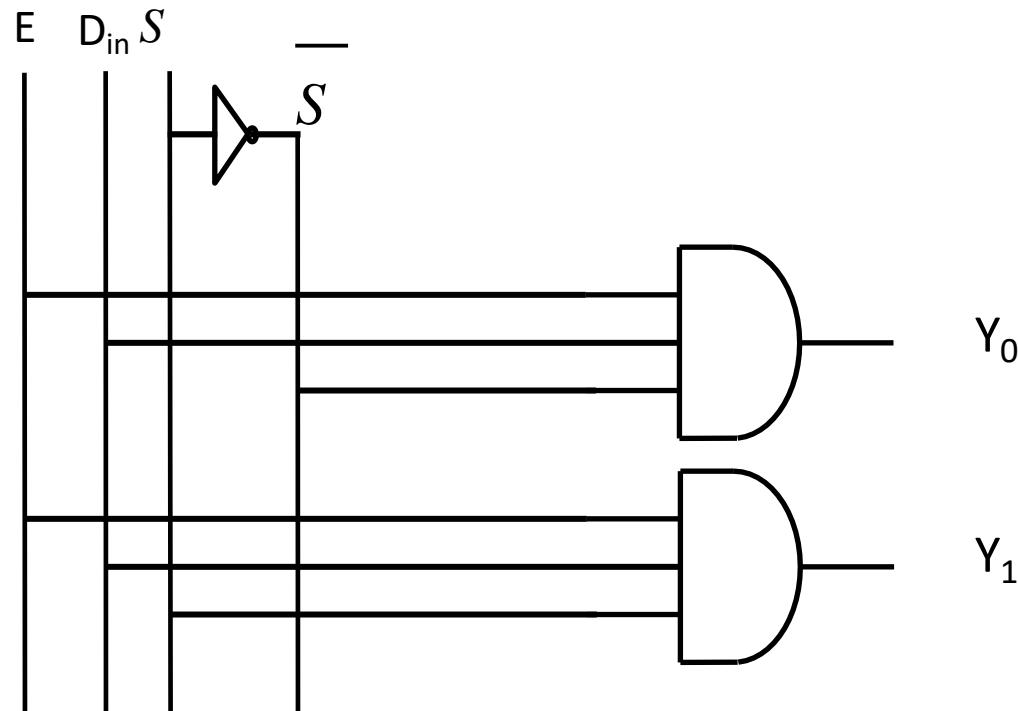
Select Lines

Truth Table

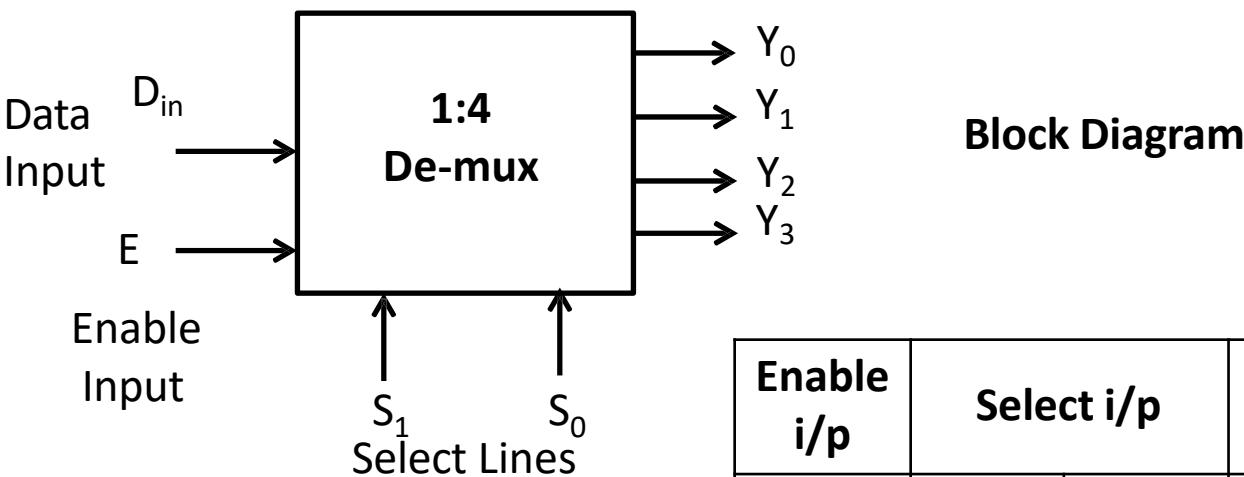
Enable i/p	Select i/p	Outputs	
$E$	$S$	$Y_0$	$Y_1$
0	X	0	0
1	0	$D_{in}$	0
1	1	0	$D_{in}$

# 1:2 De-mux using basic gates

---



# 1: 4 De-multiplexer

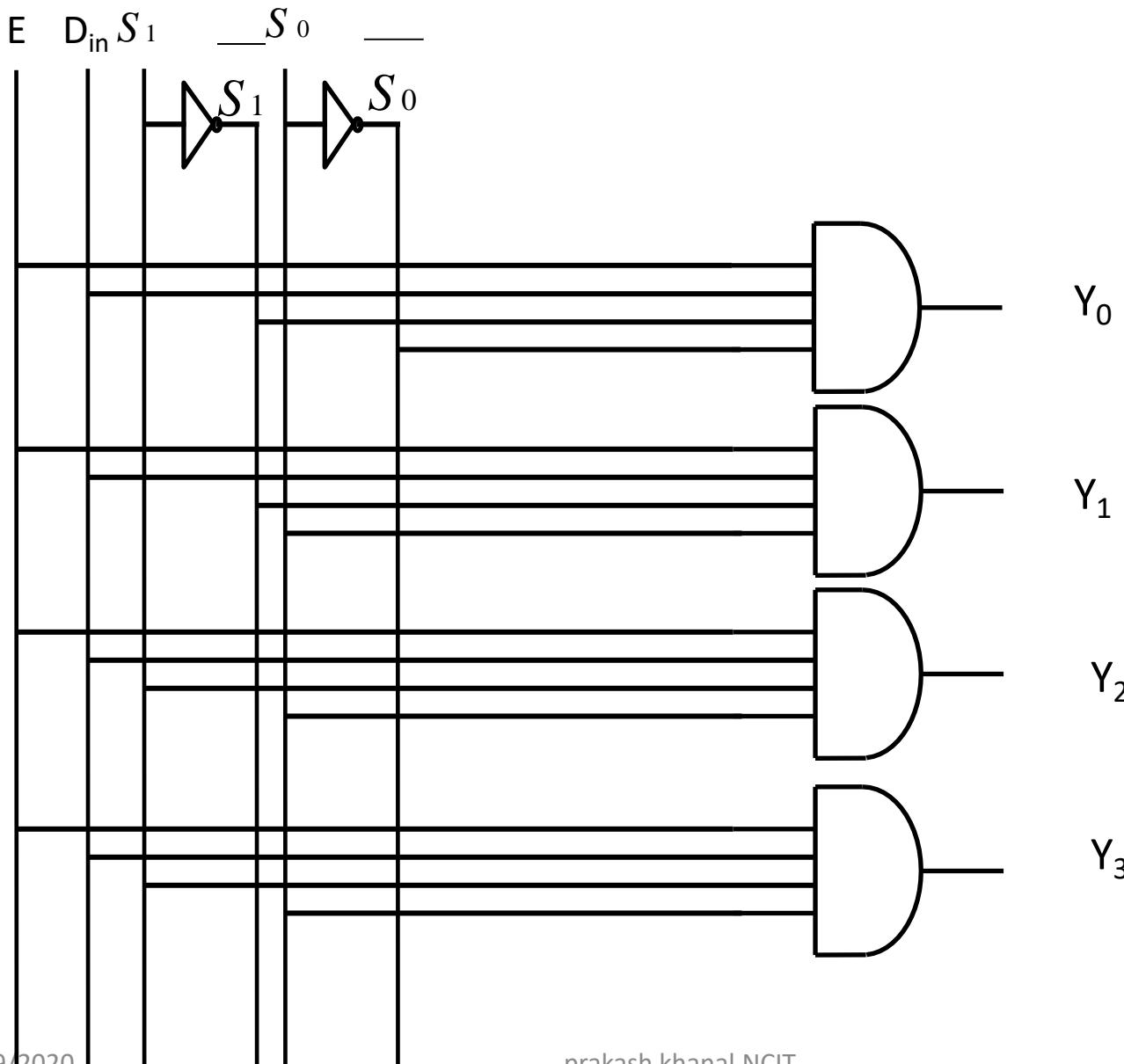


Truth Table

Enable i/p	Select i/p		Outputs				
	E	S <sub>1</sub>	S <sub>0</sub>	Y <sub>0</sub>	Y <sub>1</sub>	Y <sub>2</sub>	Y <sub>3</sub>
0	X	X		0	0	0	0
1	0	0		D <sub>in</sub>	0	0	0
1	0	1		0	D <sub>in</sub>	0	0
1	1	0		0	0	D <sub>in</sub>	0
1	1	1		0	0	0	D <sub>in</sub>

# 1:4 De-mux using basic gates

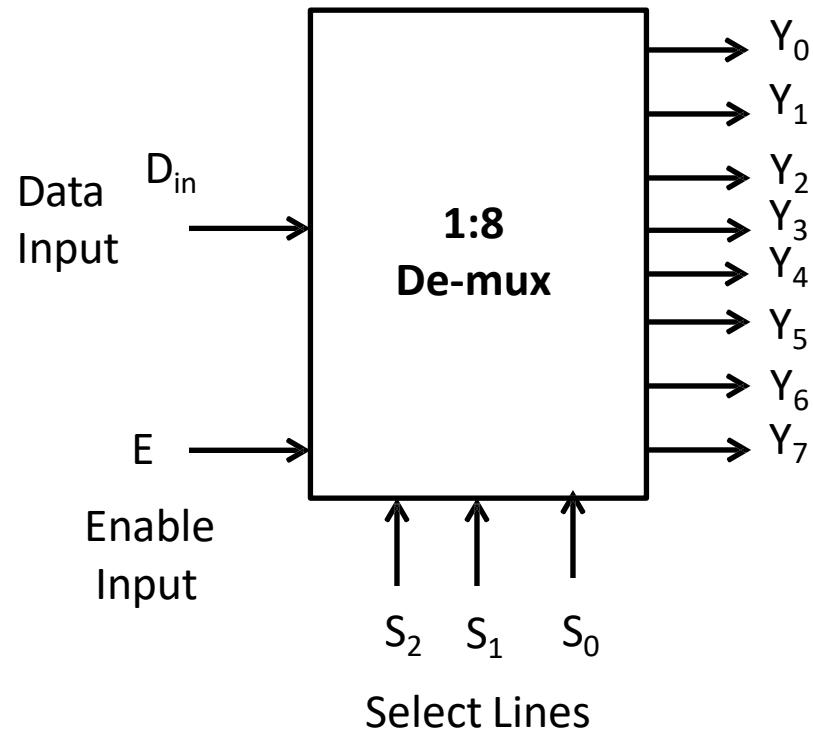
---



# 1: 8 De-multiplexer

---

Block Diagram



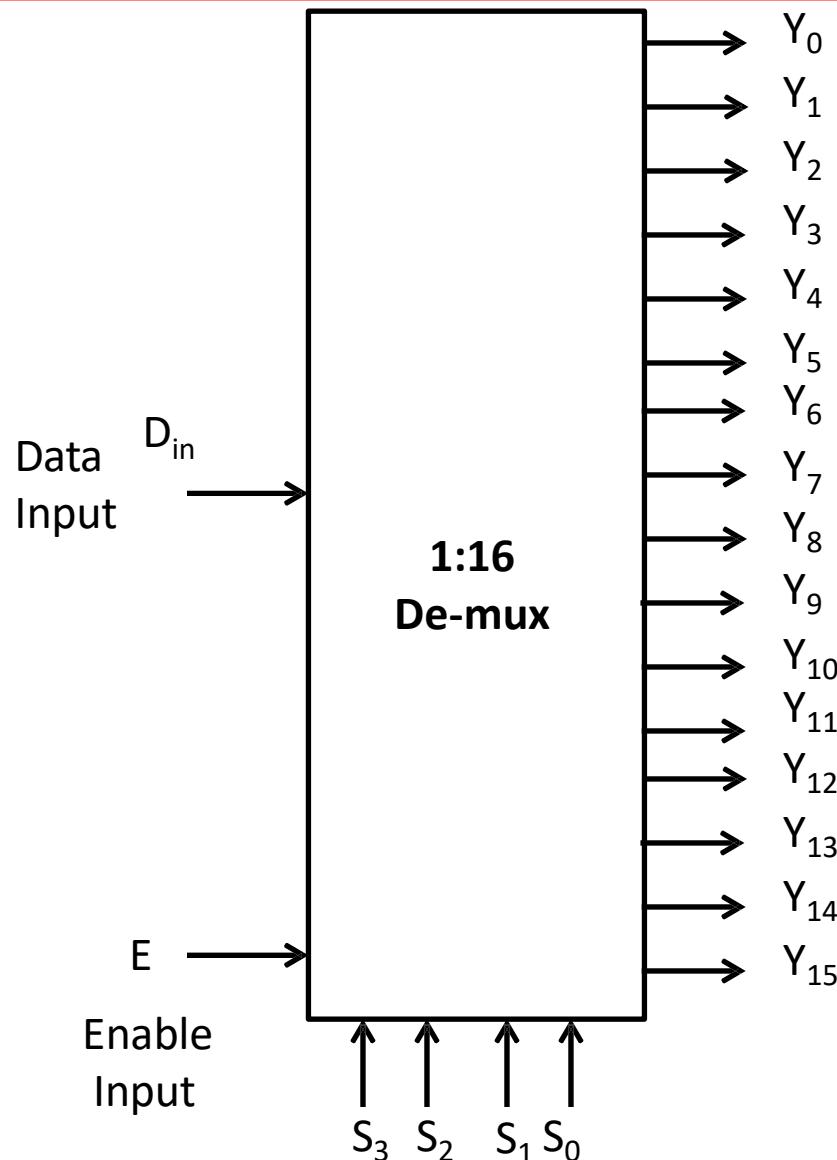
# 1: 8 De-multiplexer

Truth Table

Enabl e i/p	Select i/p			Outputs								
	E	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	Y <sub>7</sub>	Y <sub>6</sub>	Y <sub>5</sub>	Y <sub>4</sub>	Y <sub>3</sub>	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>
0	X	X	X	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	D <sub>in</sub>
1	0	0	1	0	0	0	0	0	0	0	D <sub>in</sub>	0
1	0	1	0	0	0	0	0	0	0	D <sub>in</sub>	0	0
1	0	1	1	0	0	0	0	0	D <sub>in</sub>	0	0	0
1	1	0	0	0	0	0	0	D <sub>in</sub>	0	0	0	0
1	1	0	1	0	0	D <sub>in</sub>	0	0	0	0	0	0
1	1	1	0	0	D <sub>in</sub>	0	0	0	0	0	0	0
1	11/29/2020	1	1	D <sub>in</sub>	0	0	0	0	0	0	0	75 0

# 1: 16 De-multiplexer

Block Diagram



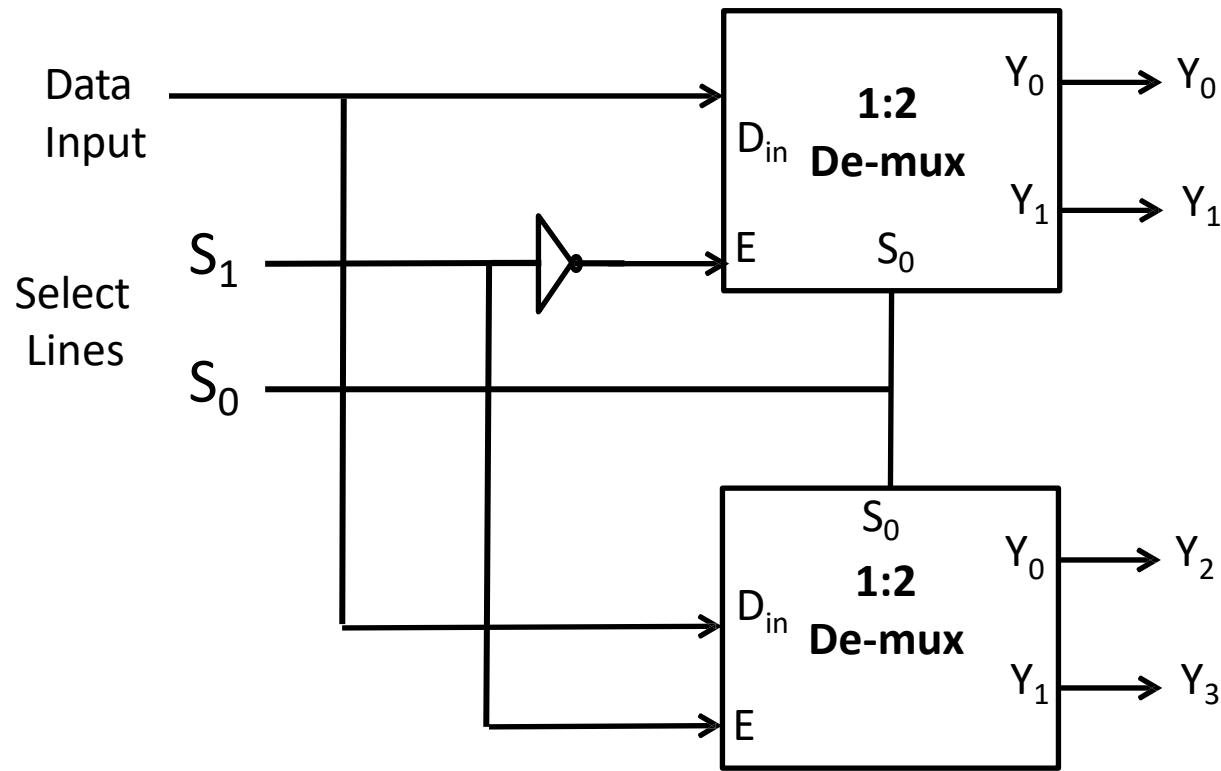
## De-mux Tree

---

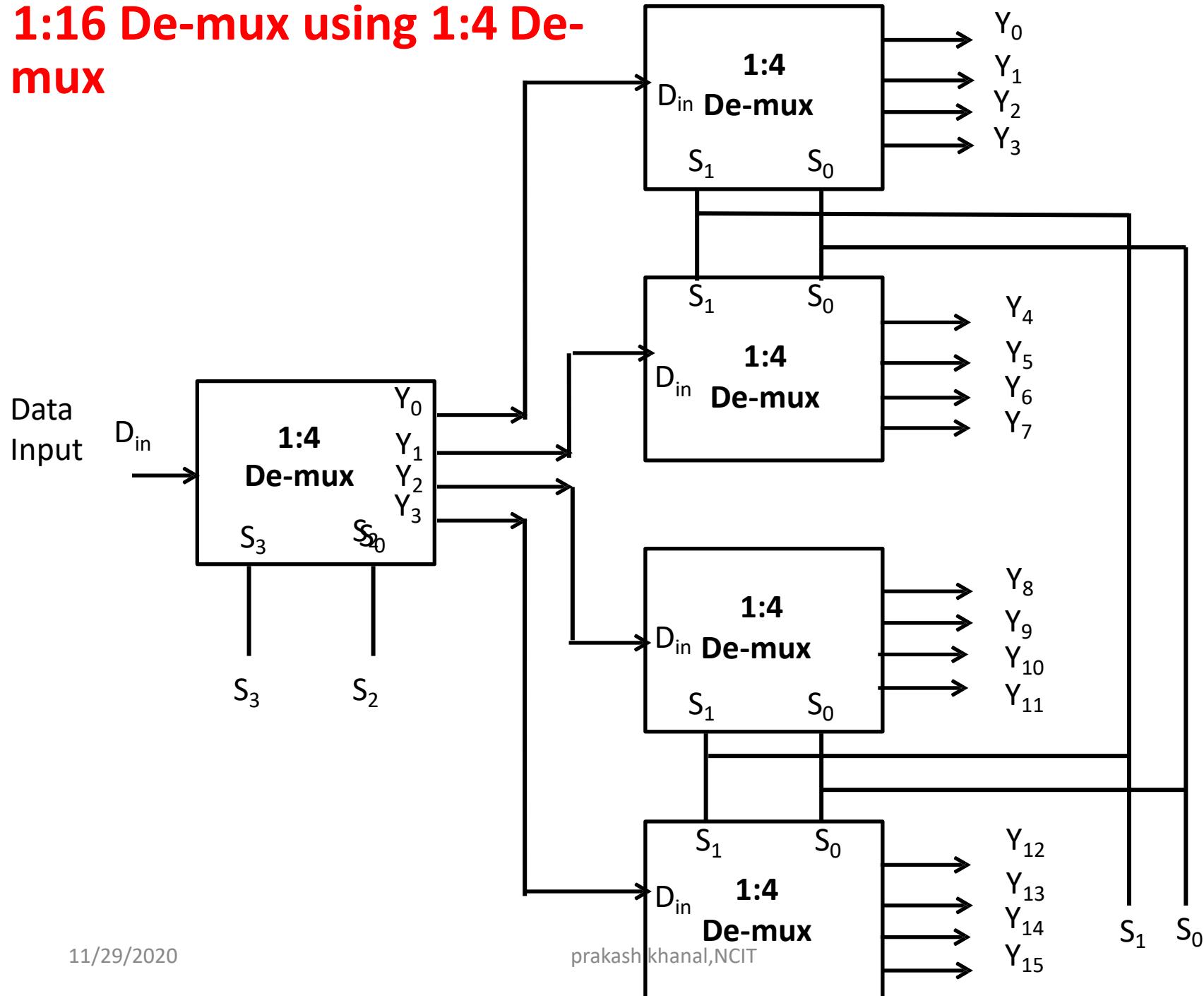
✓ Similar to multiplexer we can construct the de-multiplexer with more number of lines using de-multiplexer having less number of lines. This is called as “De-mux Tree”.

# 1:4 De-mux using 1:2 De-mux

---



# 1:16 De-mux using 1:4 De-mux

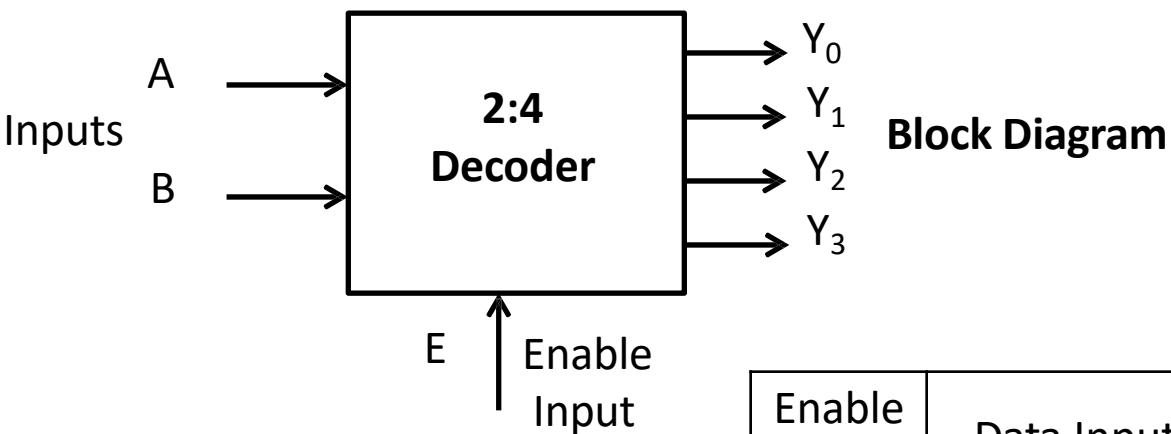


# Decoder

---

- ✓ Decoder is a combinational circuit.
- ✓ It converts  $n$  bit binary information at its input into a maximum of  $2^n$  output lines.
- ✓ For example, if  $n=2$  then we can design upto 2:4 decoder

# 2:4 Decoder



Truth Table

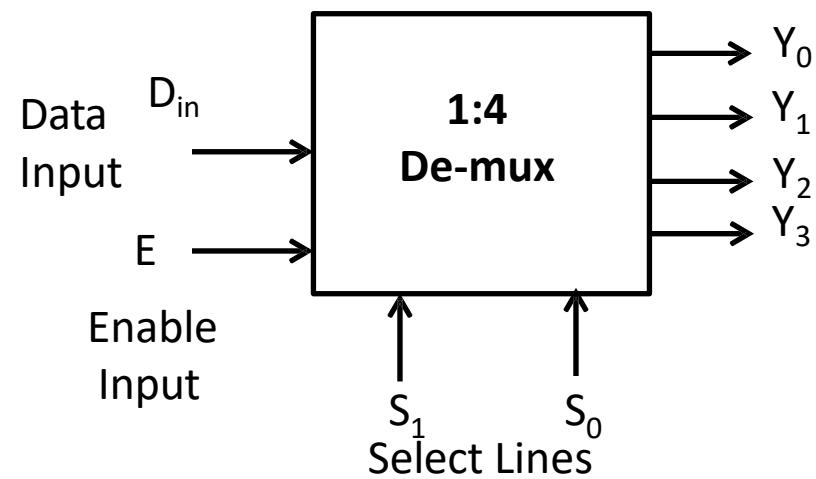
Enable i/p	Data Inputs		Outputs				
	E	A	B	Y <sub>0</sub>	Y <sub>1</sub>	Y <sub>2</sub>	Y <sub>3</sub>
0	X	X		0	0	0	0
1	0	0		1	0	0	0
1	0	1		0	1	0	0
1	1	0		0	0	1	0
1	1	1		0	0	0	1

## De-multiplexer as Decoder

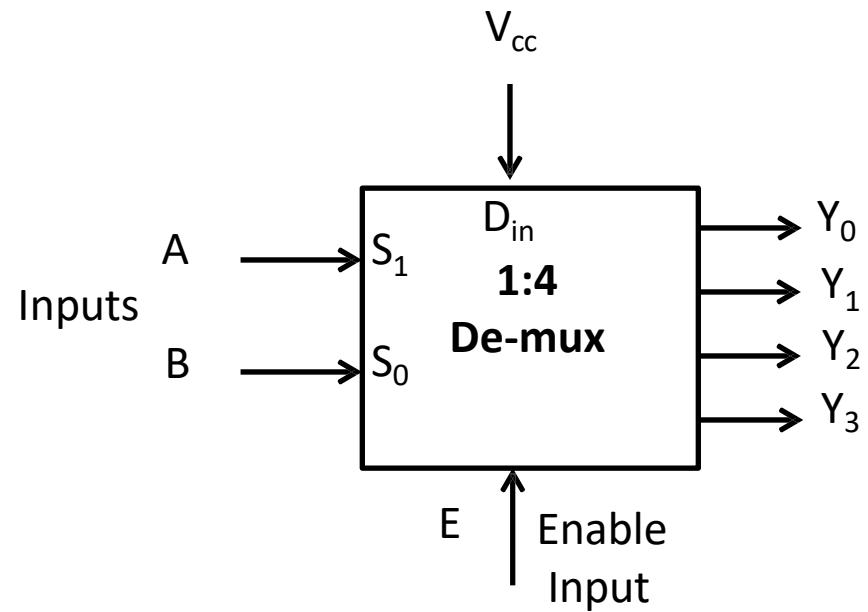
---

- ✓ It is possible to operate a de-multiplexer as a decoder.
- ✓ Let us consider an example of 1:4 de-mux can be used as 2:4 decoder

# 1:4 De-multiplexer as 2:4 Decoder



**1: 4 De-multiplexer**



**1: 4 De-multiplexer as 2:4 Decoder**

# Realization of Boolean expression using De-mux

---

- ✓ We can implement any Boolean expression using de-multiplexers.
- ✓ It reduces circuit complexity.
- ✓ It does not require any simplification

## Example 1

---

Implement following Boolean expression using de-multiplexer

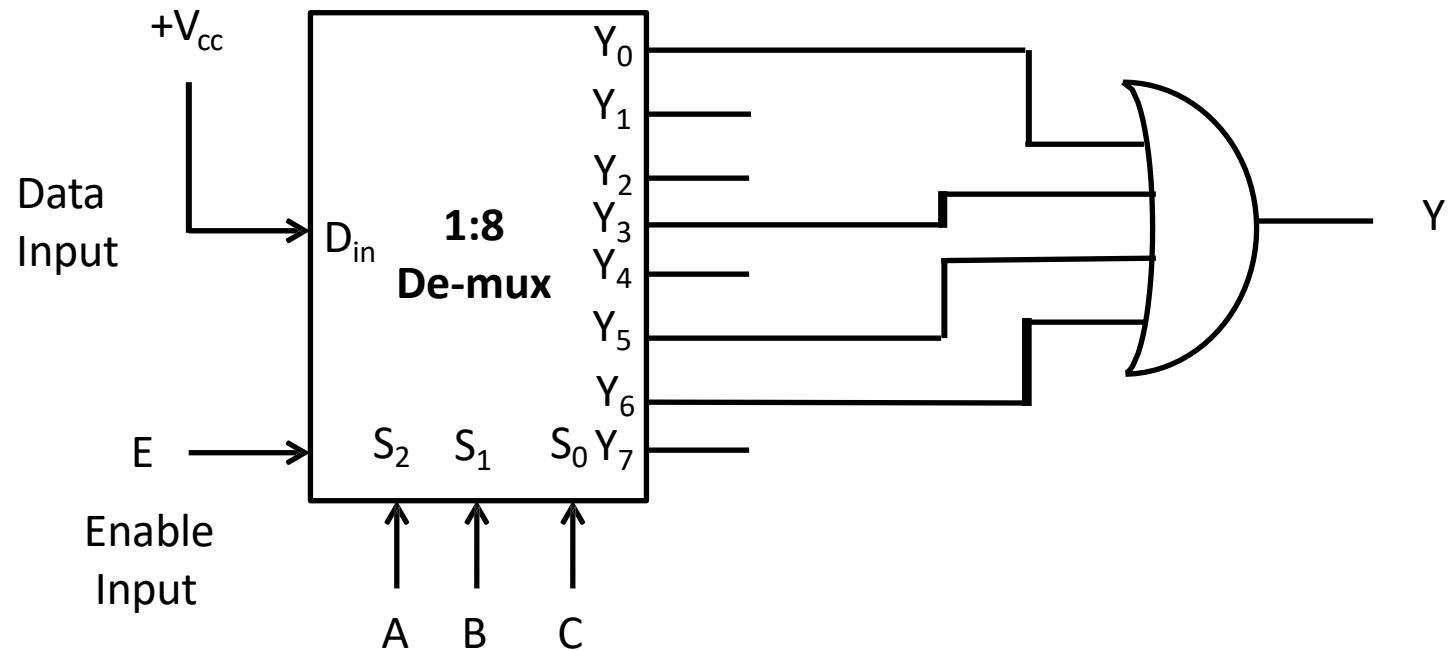
$$f(A, B, C) = \sum m(0, 3, 5, 6)$$

- ✓ Since there are three variables, therefore a de-multiplexer with three select input is required i.e. 1:8 de-multiplexer is required
- ✓ The 1:8 de-multiplexer is configured as below to implement given Boolean expression

# Example 1

continue.....

$$f(A, B, C) = \sum m(0, 3, 5, 6)$$



## Example 2

---

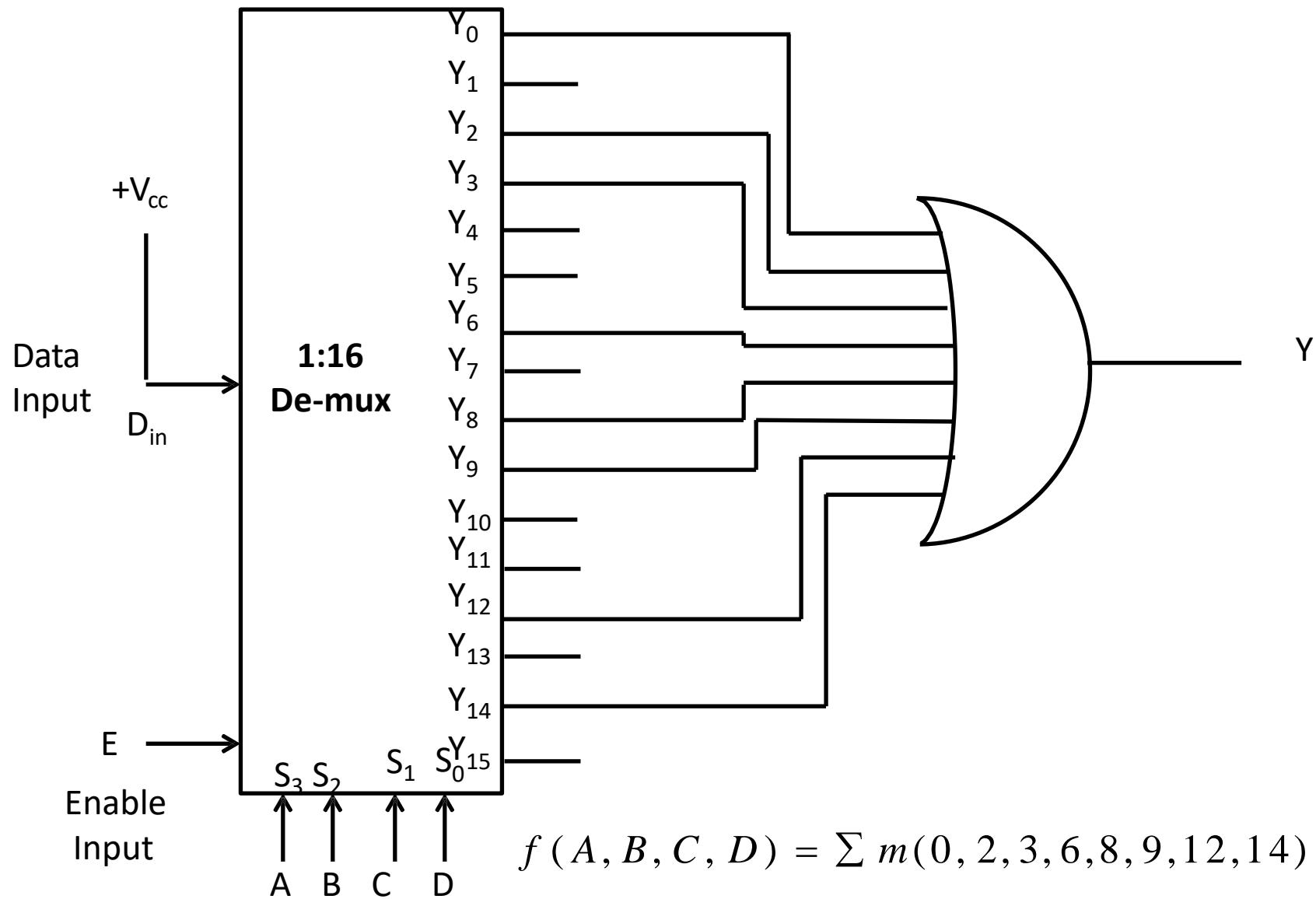
Implement following Boolean expression using de-multiplexer

$$f(A, B, C, D) = \sum m(0, 2, 3, 6, 8, 9, 12, 14)$$

- ✓ Since there are four variables, therefore a de-multiplexer with four select input is required i.e. 1:16 de-multiplexer is required
- ✓ The 1:16 de-multiplexer is configured as below to implement given Boolean expression

## Example 2

continue.....



## Tristate Logic

---

✓ In digital electronics three-state, tri-state, or 3-state logic allows an output port to assume a high impedance state in addition to the 0 and 1 logic levels, effectively removing the output from the circuit.

# **TYPES OF ROM:**

- Mask ROM.
  - Programmable ROM(PROM).
  - Erasable ROM(EROM).
  - Ultraviolet EPROM(UV PROM).
  - Electrically Erasable PROM(EEPROM)
- 
- Assignment: Discuss each type of ROM in brief.

# Programmable Array Logic(PAL):

- PAL is programmable array of logic gates on a single chip.
- The basic PAL consists of a programmable AND array and fixed OR array with output logic as shown in block diagram below.
- PAL is most common one time programmable(OTP) Logic device and is implemented with bipolar technology(TTL or ECL)

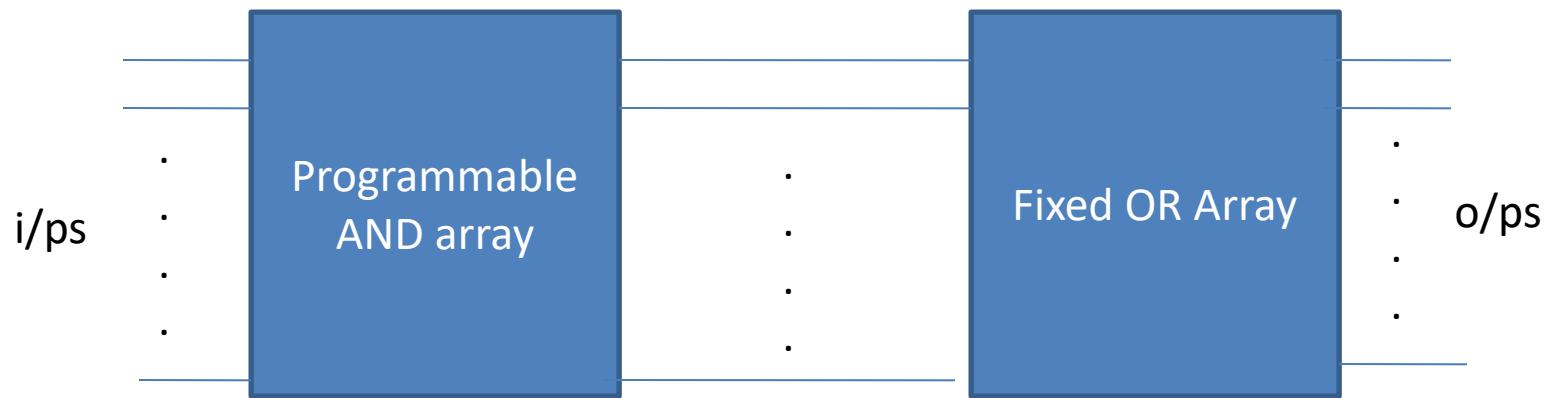


Fig: Block diagram of PAL

# Programmable Logic Array:

- A PLA consists of a programmable AND array and a programmable OR array.
- The PLA is also called a field programmable Logic array (FPLA) because the user in the field programs it, not the manufacturer.
- The use of PLA must be considered for combinational circuits that have a large number of input and outputs. It is superior to a ROM for circuits that have a large

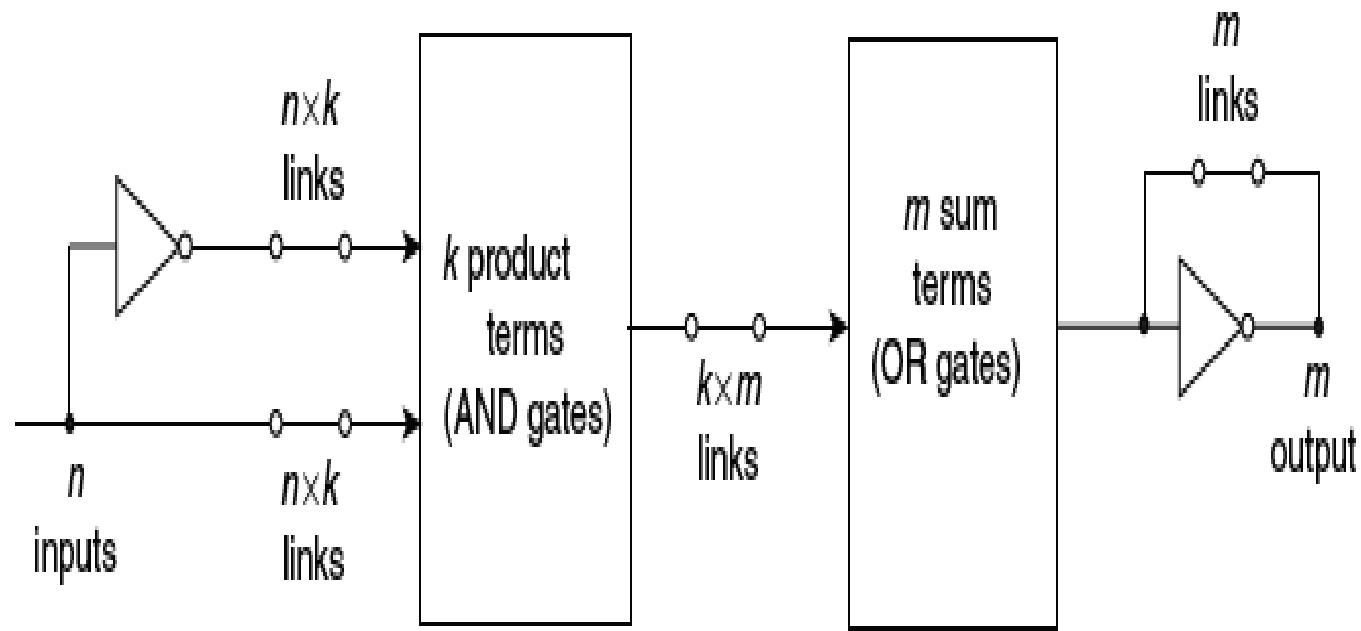
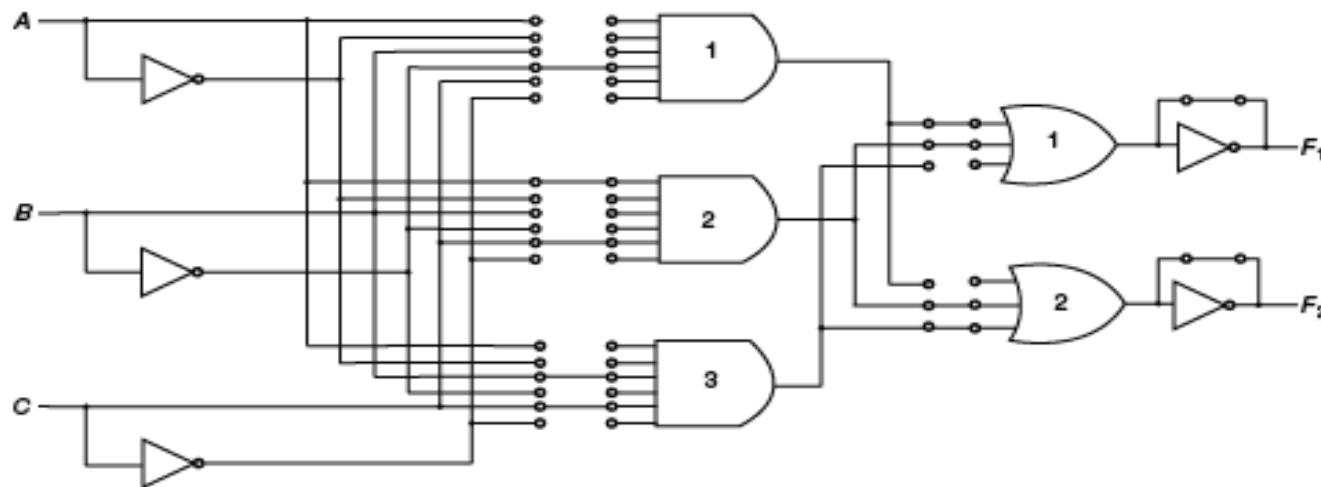


Figure 5-25 PLA block diagram

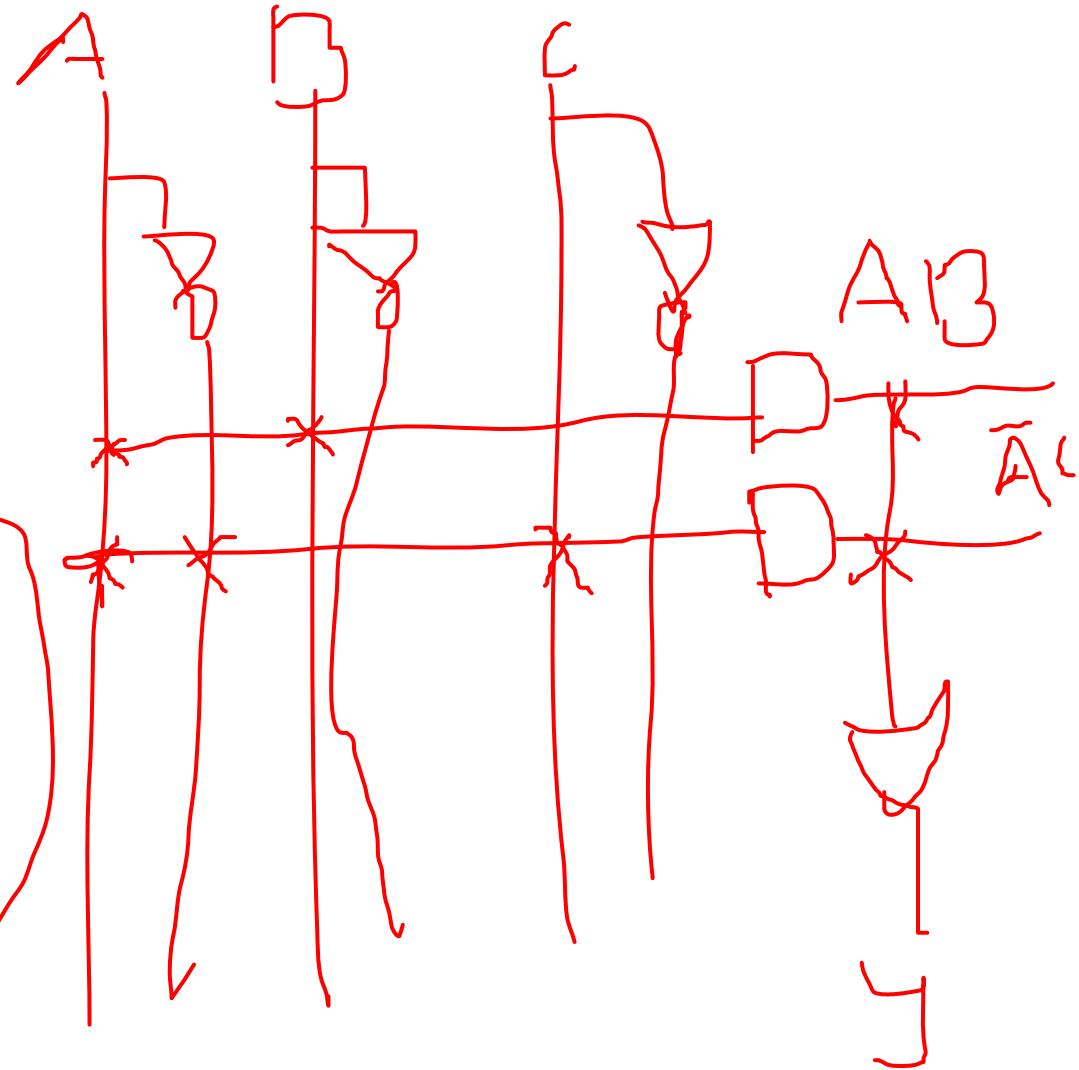


**Figure 5-26** PLA with 3 inputs, 3 product terms, and 2 outputs; it implements the combinational circuit specified in Fig. 5-27

# Implement function using PLA

- $F_0 = \text{sm}(0,1,4,6)$
- $F_1 = \text{sm}(2,3,4,6,7)$
- $F_2 = \text{sm}(0,1,2,6)$

$$Y = \overline{AB} + \overline{AC}$$



**EXAMPLE 5-6:** A combinational circuit is defined by the functions:

$$F_1(A, B, C) = \Sigma(3, 5, 6, 7)$$

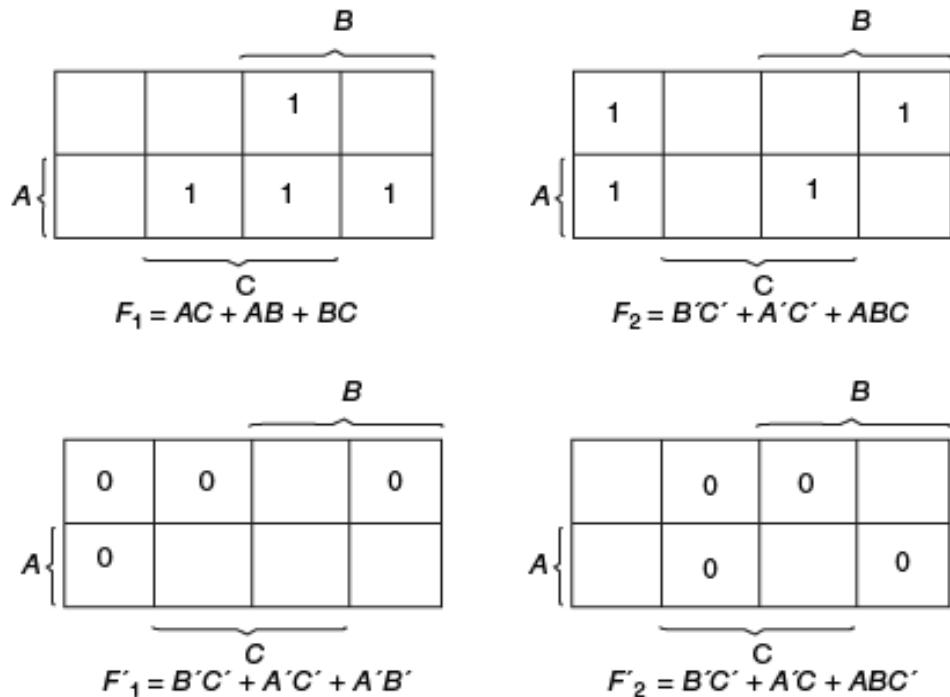
$$F_2(A, B, C) = \Sigma(0, 2, 4, 7)$$

Implement the circuit with a PLA having three inputs, four product terms, and two outputs. The two functions are simplified in the maps of Fig. 5-28. Both the true values and the complements of the functions are simplified. The combinations that gives a minimum number of product terms are:

$$F_1 = (B'C + A'C' + A'B')'$$

$$F_2 = B'C + A'C' + ABC$$

This gives only four distinct product terms:  $B'C$ ,  $A'C'$ ,  $A'B'$ , and  $ABC$ . The PLA program table for this combination is shown in Fig. 5-28. Note that output  $F_1$  is the normal (or true) output even though a  $C$  is marked under it. This is because  $F_1'$  is generated *prior to* the output inverter. The inverter complements the function to produce  $F_1$  in the output.



PLA program table

Product term	Inputs			Outputs	
	$A$	$B$	$C$	$F_1$	$F_2$
$B'C'$	1	-	0 0	1	1
$A'C'$	2	0	- 0	1	1
$A'B'$	3	0 0	-	1	-
$ABC$	4	1 1	1	-	1
				$C$	$T$
				$T/C$	

Figure 5-28 Solution to Example 5-6

# Implement a full adder using PLA:

- Output= sum and carry
- Sum=  $sm(1,2,4,7) = A'B'C + A'BC' + ABC + AB'C'$
- Carry= $sm(3,5,6,7) = A'BC + AB'C + ABC' + ABC$

## **Serial form of Data Vs Parallel Form of Data**

---

- ✓ Data may be available in Parallel form or Serial form.
- ✓ Multi bit data is said to be in parallel form when all the bits are available (accessible) simultaneously.
- ✓ The data is said to be in serial form when data bits appear sequentially (one after another in time) at a single terminal.
- ✓ Data may also be transferred in parallel form or in serial form.

## Data Transmission Serial/Parallel

---

- ✓ Parallel data transfer is the simultaneous transmission of all bits of data from one device to another.
- ✓ Serial data transfer is the transmission of one bit of data at time from one device to another.
- ✓ Serial data must be transmitted under the synchronization of a clock, since clock provides the means to specify the time at which each new bit is sampled

# Register

---

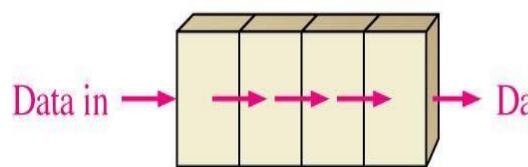
- ✓ As a flip flop can store only one bit of data, a 0 or a 1, it is referred as a single bit register.
- ✓ When more bits of data are to be stored, a number of FFs used.
- ✓ A register is a set of FFs used to store binary data.
- ✓ The storage capacity of a register is the number of bits (1s and 0s) of digital data it can retain.
- ✓ A register may have output data either in serial form or in parallel form.

# Shift Register

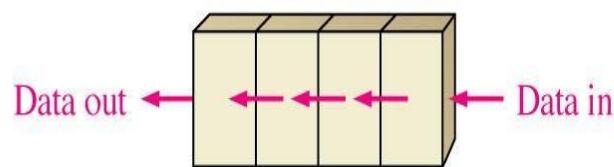
---

- ✓ A shift register is a very important digital building blocks. It has innumerable applications.
- ✓ Shift registers are a type of logic circuits closely related to counters.
- ✓ They are used basically for storage and transfer of digital data.
- ✓ The basic difference between a shift register and a counter is that, a shift register has no specified sequence of states whereas a counter has a specified sequence of states.

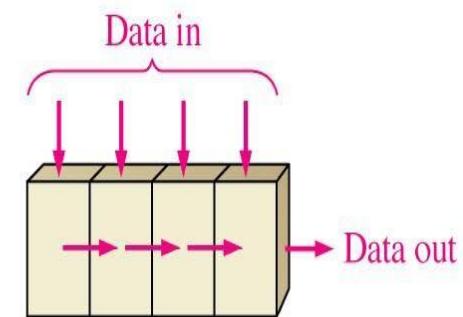
# Basic Data Movements in Shift Registers



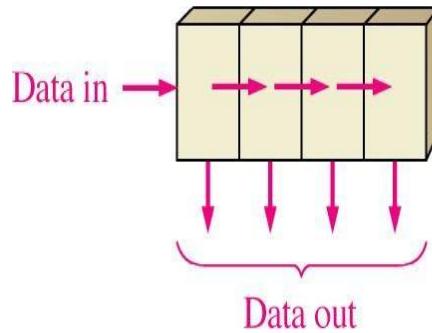
(a) Serial in/shift right/serial out



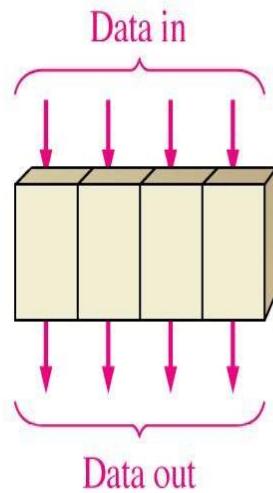
(b) Serial in/shift left/serial out



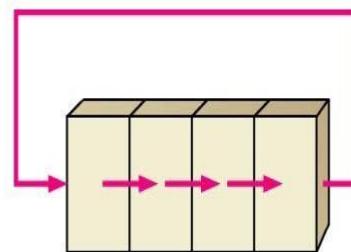
(c) Parallel in/serial out



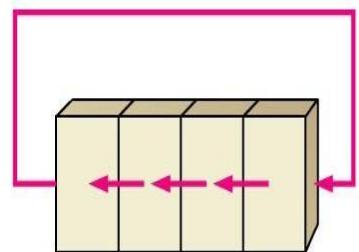
(d) Serial in/parallel out



(e) Parallel in/parallel out



(f) Rotate right



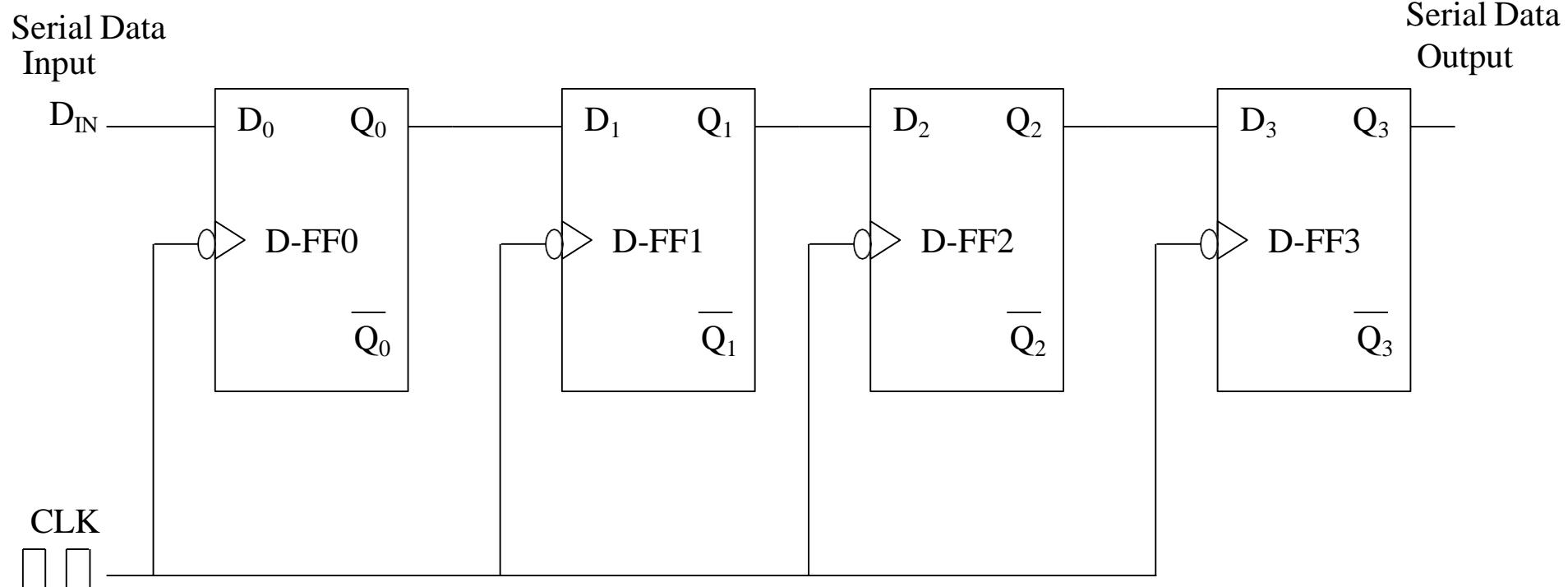
(g) Rotate left

# Types of Shift Registers

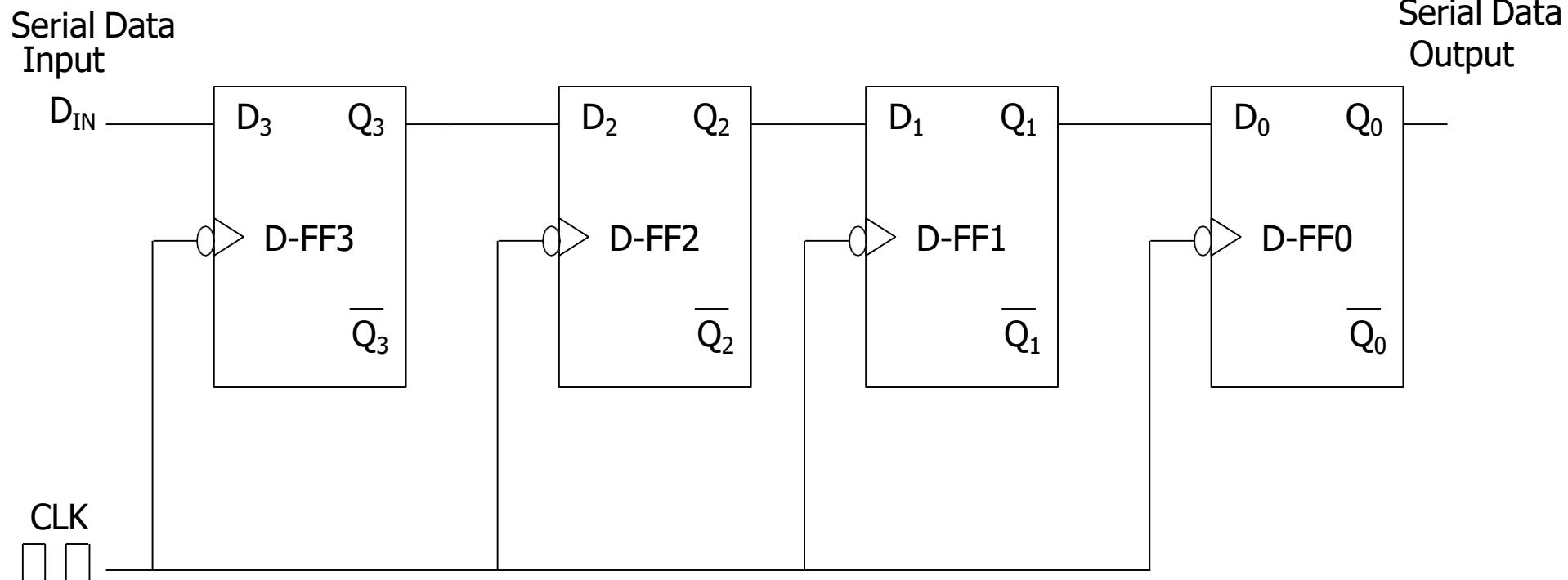
---

- ✓ SISO – Serial In Serial Out Shift Register
- ✓ SIPO – Serial In Parallel Out Shift Register
- ✓ PISO – Parallel In Serial Out Shift Register
- ✓ PIPO – Parallel In Parallel Out Shift Register
- ✓ Bi-directional Shift Register
- ✓ Universal Shift Register

# SISO – Serial In Serial Out Shift Register (Shift Left)

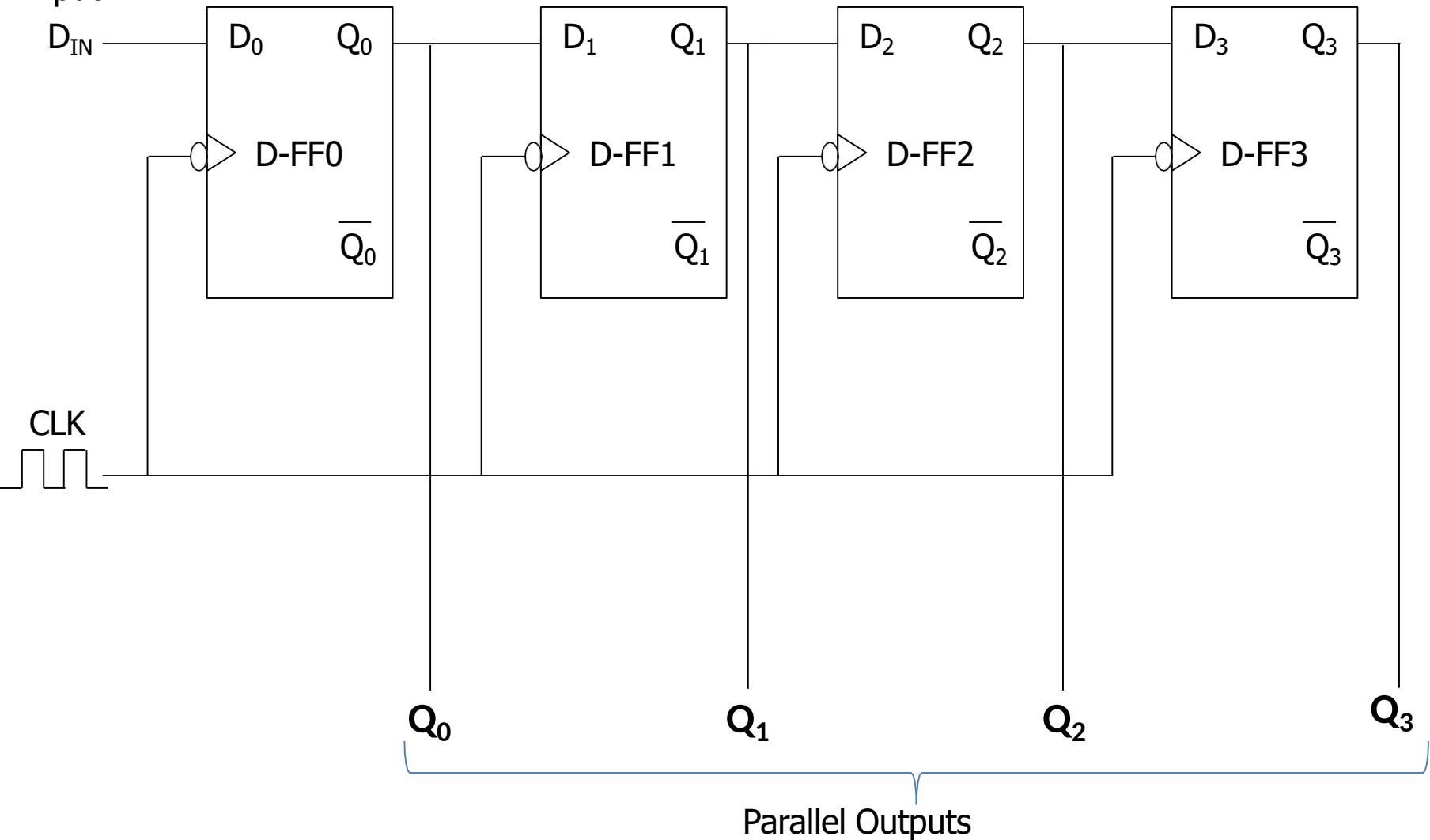


# SISO – Serial In Serial Out Shift Register (Shift Right)

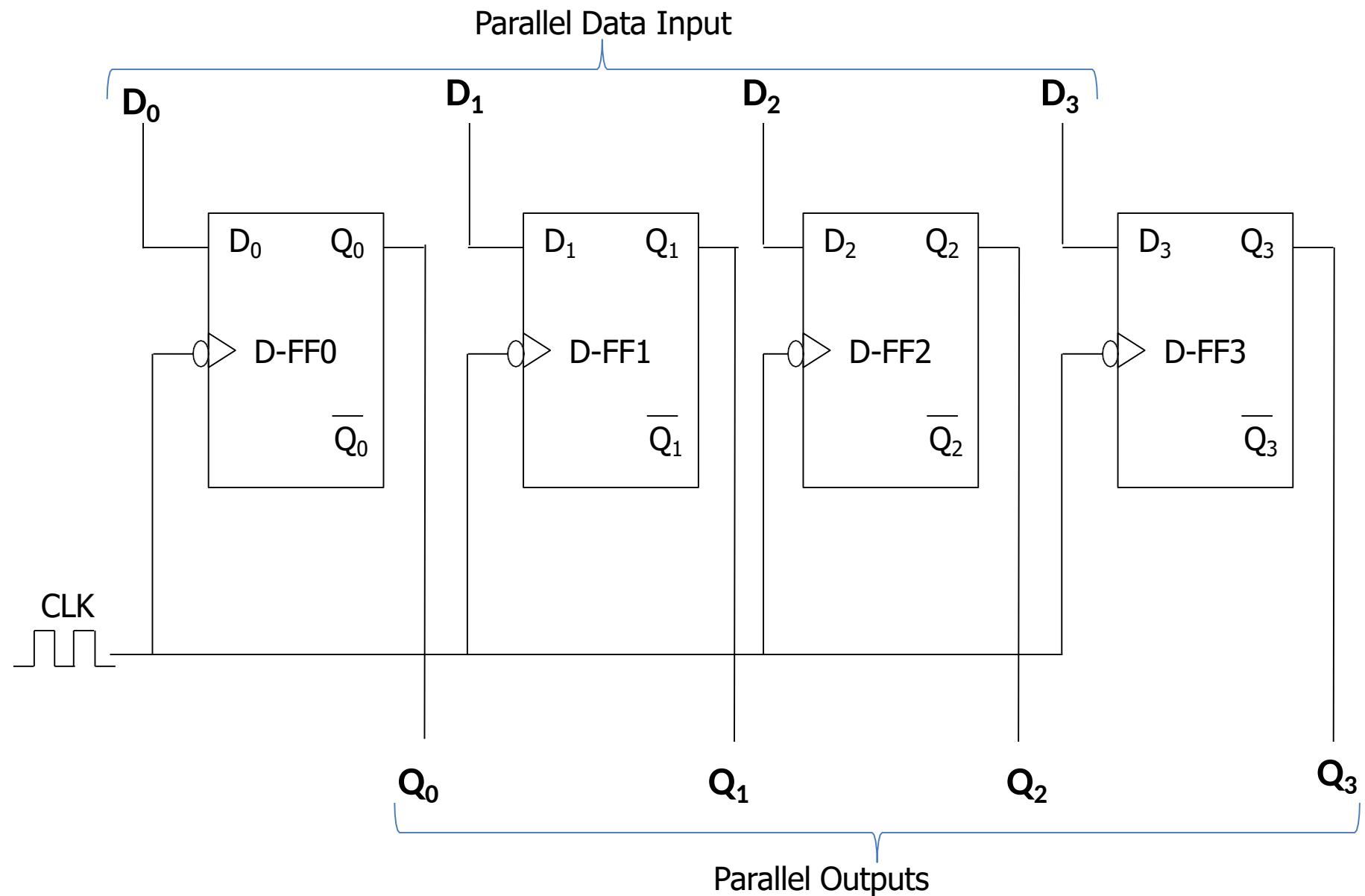


# SIFO – Serial In Parallel Out Shift Register

Serial Data  
Input

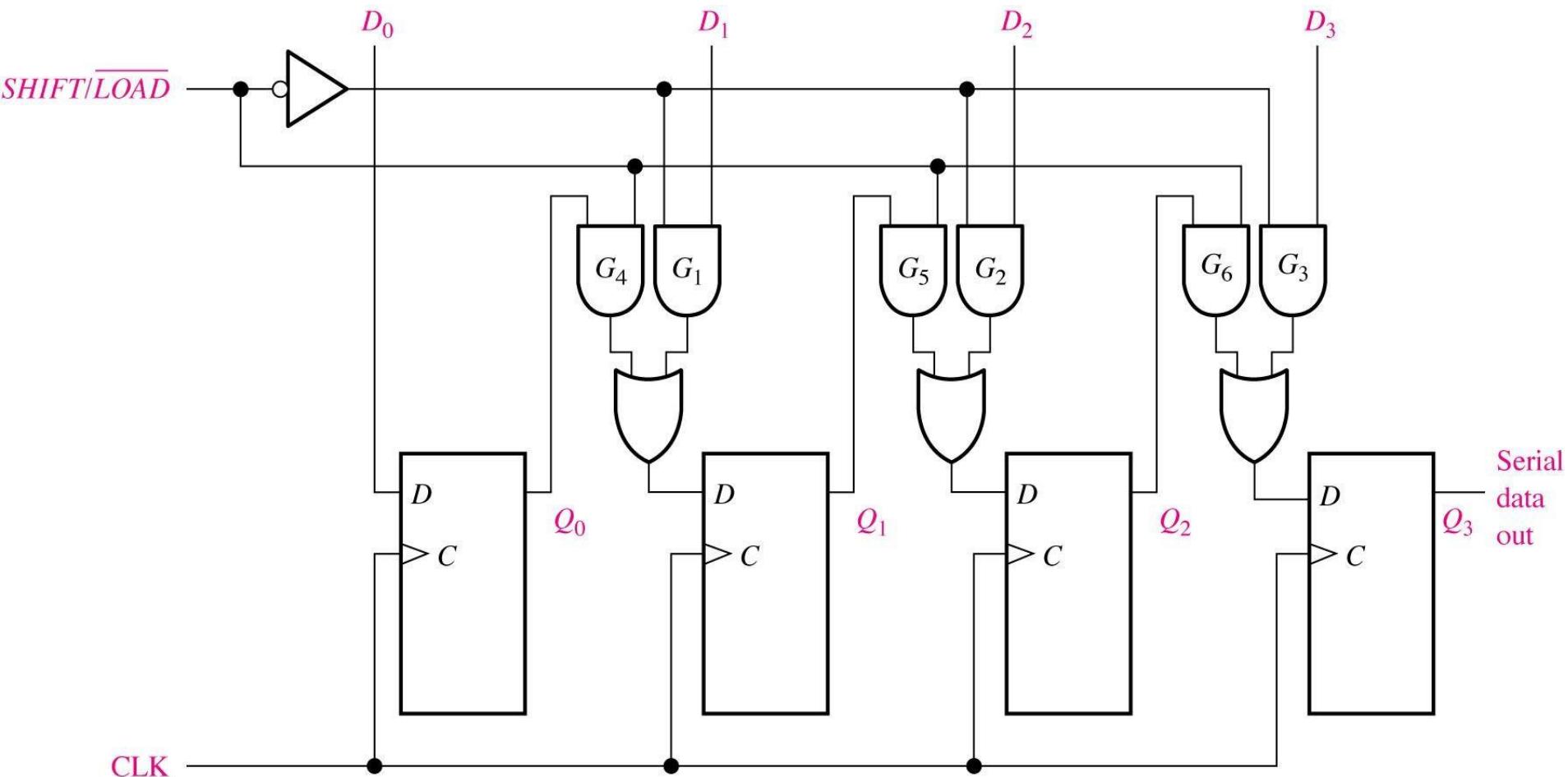


# PIPO – Parallel In Parallel Out Shift Register



# PISO – Parallel In Serial Out Shift Register

---



# PISO – Parallel In Serial Out Shift Register

---

## Load Mode:

- ✓ When the *Shift / Load* line is Low, the AND gates G1, G2 and G3 become active. They will pass D1, D2, and D3 bits to the corresponding Flip Flops.
- ✓ On the low going edge of clock, the binary inputs D0, D1, D2 and D3 will get loaded into corresponding flip flops. Thus parallel loading takes place.

# PISO – Parallel In Serial Out Shift Register

---

## Shift Mode:

- ✓ When the *Shift / Load* line is High, the AND gates G1, G2 and G3 become inactive. Hence parallel loading of data becomes impossible.
- ✓ But AND gates G4, G5 and G6 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses
- ✓ Thus the parallel in serial out operation takes place

## Shift register:

- A register capable of shifting its binary information either to the right or to the left.
- The logical configuration of shift register consists of a chain of flipflop connected in cascade.
- A flipflop receives a common clock pulse which causes the shift from one stage to another.

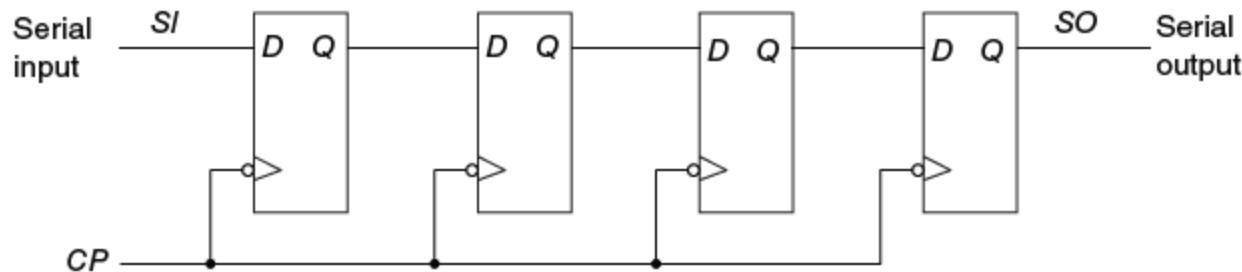
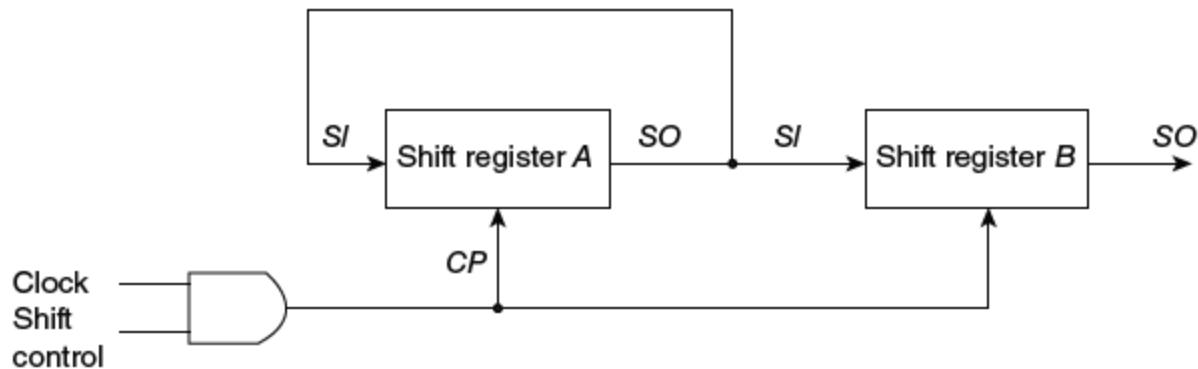


Figure 7-7 Shift register

## Serial transfer:

- A digital system is said to operate in a serial mode when information is transferred one bit at a time.
- The content of one register is transferred to another by shifting the bit from one register to another.
- The serial transfer of information from register A to register B is done with shift register as shown in block diagram.
- The serial output of register A (SO) goes to serial input of register A (Si) of register B
- To prevent the loss of information stored in the source register, the A register is made to circulate the information by connecting the serial output to the input terminal.



(a) Block diagram

- Assume the binary content of A before the shift is 1011 and that of B is 0010.
- The serial transfer from A to B will occur in 4 steps as shown in table.
- After the first pulse  $T_1$  the rightmost bit of A is shifted into the leftmost bit of B and at the same time , this bit is circulated into the leftmost position of A.
- Once the bit of A and B are shift to right, the previous serial output from B is lost.

Timing pulse	Shift register A	Shift register B	Serial output of B
Initial value	1 0 1 1	0 1 1 0	0
After $T_1$	1 1 0 1	1 0 0 1	1
After $T_2$	1 1 1 0	1 1 0 0	0
After $T_3$	0 1 1 1	0 1 1 0	0
After $T_4$	1 0 1 1	1 0 1 1	1

- Ring counter:
  - A type of counter in which the output of the last flipflop is connected as an input to the first flipflop is known as Ring Counter.
  - The input is shifted between the flipflops in ring shape so it is called as ring counter.
  - A ring counter is a synchronous counter.

- Johnson counter:
- Johnson counter as a reverse ring counter. In other words, feedback from the last flipflop is fed inversely to the data input of the first flipflop.
- For example for a D- flipflop shift register, the Q' output of the last flipflop is fed to the D input of the first flipflop.
- The Mod of the johnson counter is ‘ $2n$ ’, n is the bit size of the counter.Mod is the maximum number of states a counter can obtain.

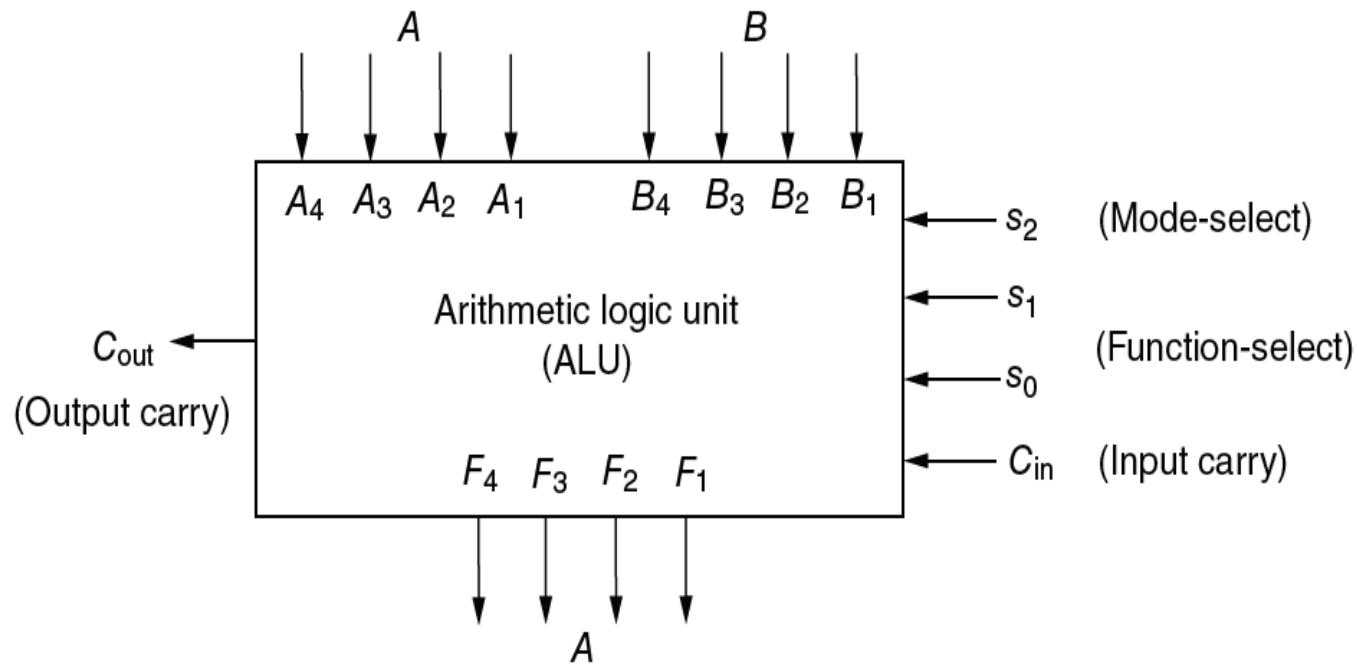
- Advantages:
- More output as compared to ring counter.
- It has the same number of flip flop but it can count twice the number of states the ring counter can count.
- It counts the data in the continuous loop.
- It only needs half the number of flip flops compared to the standard ring counter for the same MOD.

- Disadvantages:
- Only 8 of the 15 states are being used.
- It doesn't count in binary sequence.

## Arithmetic and Logic unit:

Q) Define ALU with block diagram.

- An arithmetic Logic unit (ALU) multi-operation, combinational logic digital function.
- It can perform a set of basic operation and a set of logic operation.
- Figure below shows the block diagram of 4 bit ALU. The four bit data input from 'A' are combined with 4 input from 'B' to generate 8 operation at the F output.



**Figure 9-5** Block diagram of a 4-bit ALU

- Selection line  $s_2$  is used for mode select to distinguish between arithmetic and logic operation.
- Selection line  $s_1$  and  $s_0$  specify the particular arithmetic or logic operation to be generated.
- With three selection variable, it is possible to specify four arithmetic operation and four logic operation.
- The input and output carries have meaning only during an arithmetic operation.
- The fourth selection variable  $c$ , can

## Design of Arithmetic circuit:

Q)Draw an arithmetic circuit logic diagram with function table that perform 8 different operation.

- The basic component of arithmetic section of ALU is a parallel adder.
- A parallel adder is constructed with number of flipflop. By controlling the data input to the parallel adder, it is possible to obtain 8 different types of arithmetic operation.

**1. Addition:** Arithematic addition is

2. **Addition with carry**: with  $c_{in} = 1$  in addition.

3. **A plus 1's complement of B**: By complementing all the bit of input B with  $c_{in} = 0$ .

4. **Subtraction**: By making  $cin = 1$ , it is possible to add 1 to the A plus 1's complement of B which produces the sum of A plus 2's complement of B. This operation is similar to subtraction if input carry is discarded.

5. **Transfer A**: if we force all the bit of B terminal to 0 then it transfers input A into output F.

6. **Increament A**: By making  $c_{in} = 1$  and all bit of B terminal to 0, we obtain  $F = A + 1$  which is increament of A.

7. **Decreament A**: If we insert all the bit of B terminal to with  $c_{in} = 0$  then this produce a

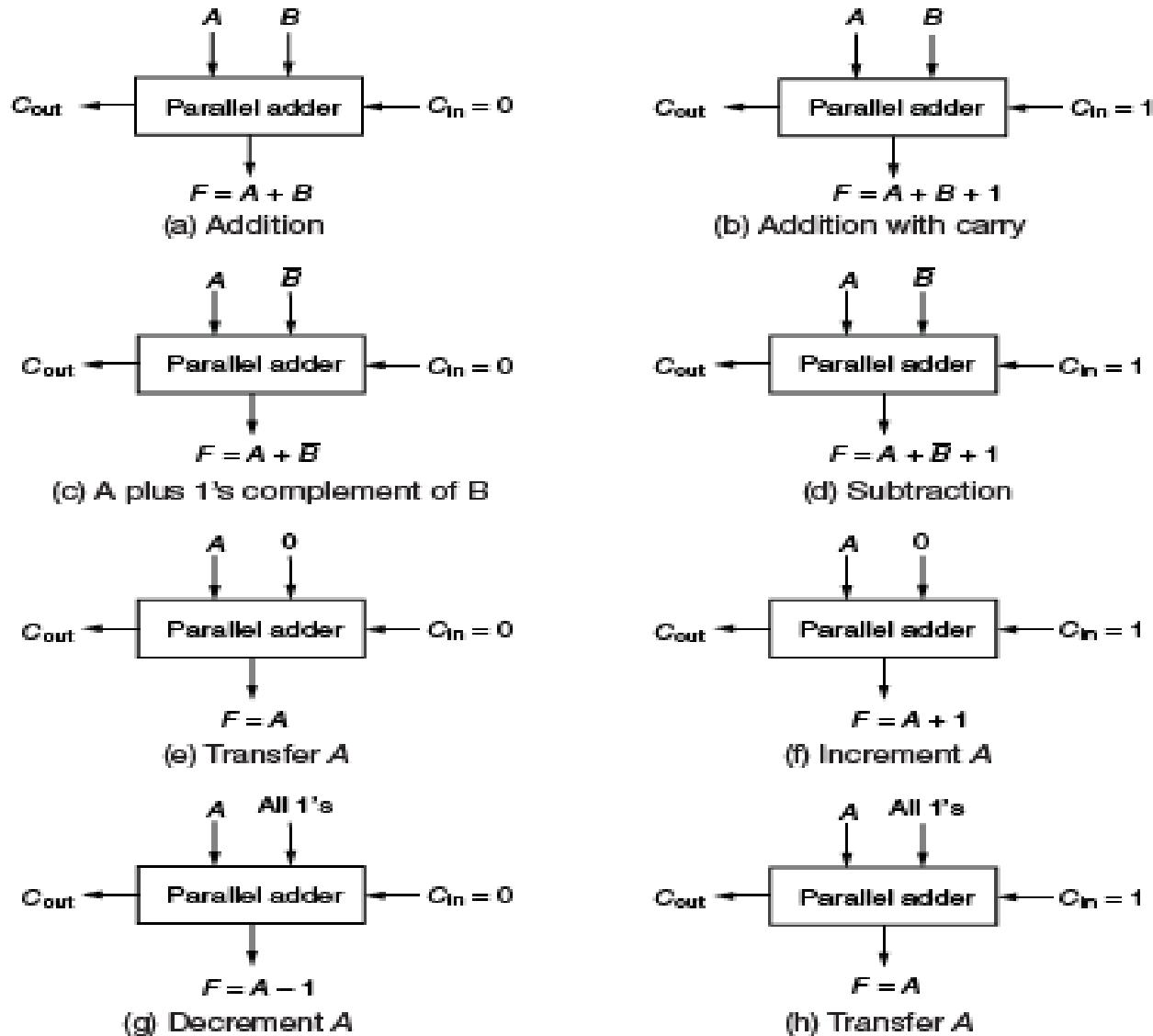


Figure 9-6 Operations obtained by controlling one set of inputs to a parallel adder

# Function table for the arithmetic circuit:

Function select			Y equals	Output equals	Function
$s_1$	$s_0$	$C_{in}$			
0	0	0	0	$F = A$	Transfer $A$
0	0	1	0	$F = A + 1$	Increment $A$
0	1	0	$B$	$F = A + B$	Add $B$ to $A$
0	1	1	$B$	$F = A + B + 1$	Add $B$ to $A$ plus 1
1	0	0	$\bar{B}$	$F = A + \bar{B}$	Add 1's complement of $B$ to $A$
1	0	1	$\bar{B}$	$F = A + \bar{B} + 1$	Add 2's complement of $B$ to $A$
1	1	0	All 1's	$F = A - 1$	Decrement $A$
1	1	1	All 1's	$F = A$	Transfer $A$

When  $s_1s_0 = 00$

$$y_i = 0$$

When  $s_1s_0 = 01$

$$y_i = B_i$$

When  $s_1s_0 = 10$

$$y_i = B'_i$$

When  $s_1s_0 = 11$

$$y_i = \text{all } 1\text{'s}$$

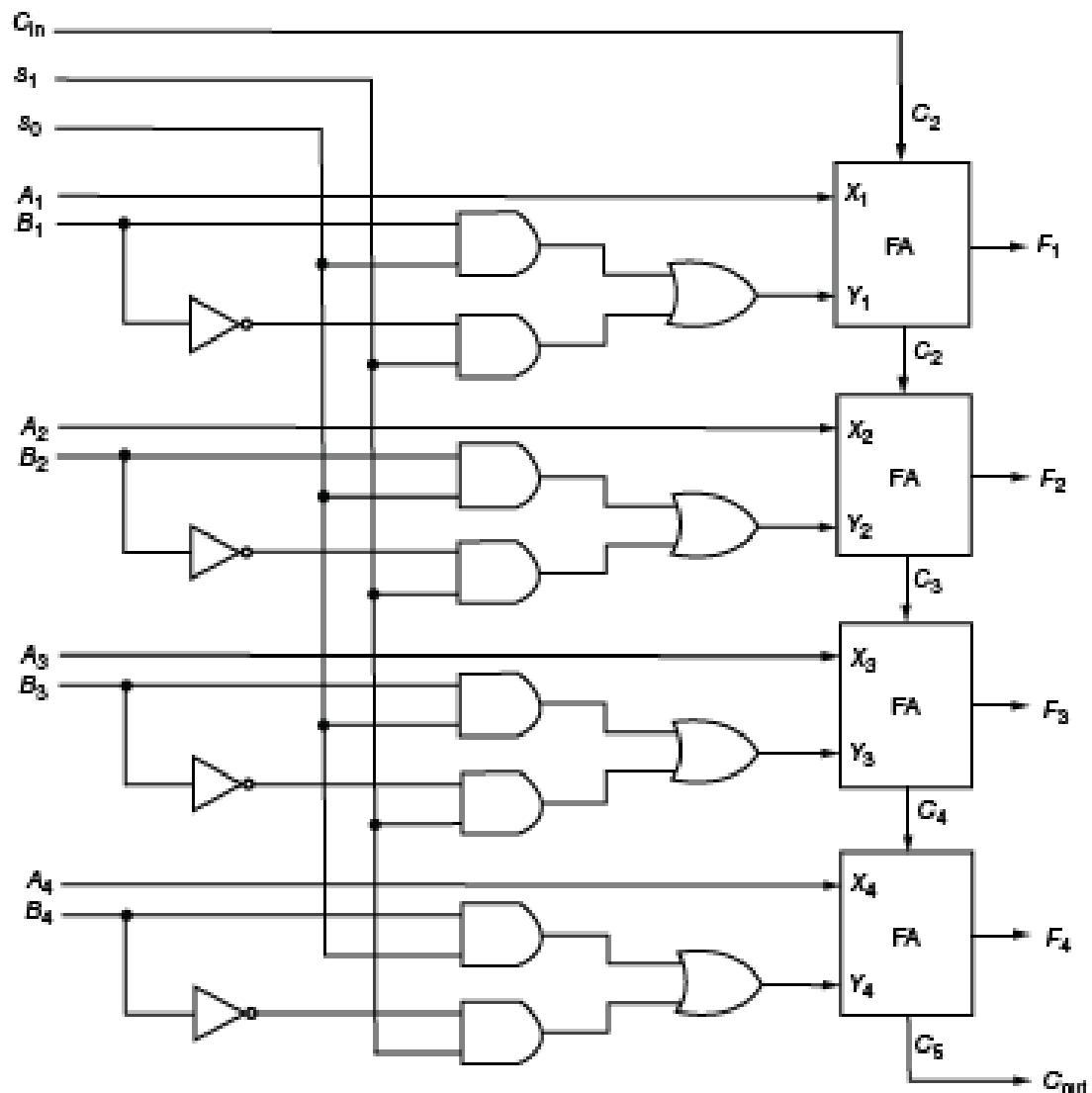
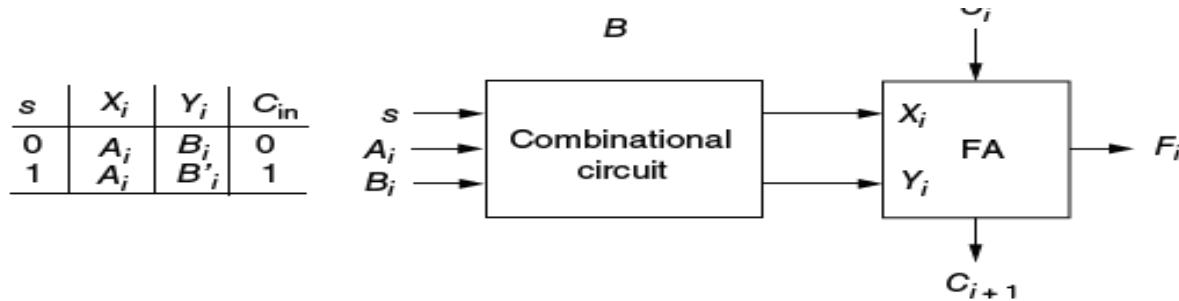


Figure 9-8 Logic diagram of arithmetic circuit

Q) Design an adder/ subtractor circuit with one selection variable and two input A and B, when  $s=0$  the circuit performs  $A+B$ . when  $S=1$  the circuit performs  $A-B$  by taking the 2's complement of B

The arithmetic circuit and truth table for given condition will be as:



(b) Specifying combinational circuit

$s$	$A_i$	$B_i$	$X_i$	$Y_i$	
0	0	0	0	0	
0	0	1	0	1	$X_i = A_i$
0	1	0	1	0	$Y_i = B_i \oplus s$
0	1	1	1	1	$C_{in} = s$
1	0	0	0	1	
1	0	1	0	0	
1	1	0	1	1	
1	1	1	1	0	

(c) Truth table and simplified equations

Figure 9.9 Derivation of an adder/subtractor circuit

When  $s=0$   $x_i$  and  $y_i$  for each full adder must be equal to the external input  $A_i$  and  $B_i$  Respectively. When  $s = 1$ ,  $x_i = A_i$  and  $y_i = B_i'$ . The input carry must be equal to value of  $s$ .

The logic diagram will be as:

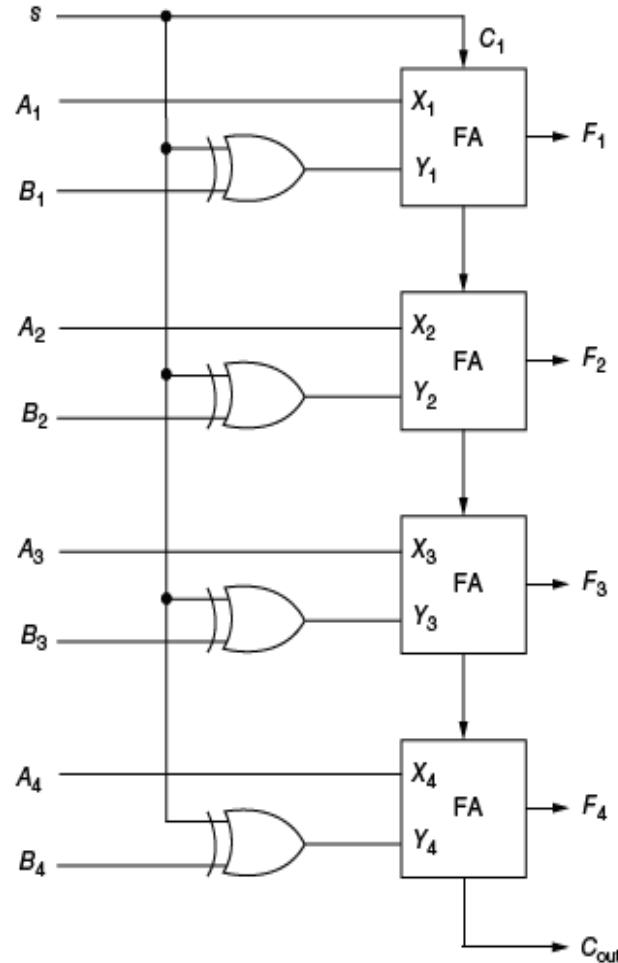


Figure 9-10 4-bit adder/subtractor circuit

## Shift register:

- A register capable of shifting its binary information either to the right or to the left.
- The logical configuration of shift register consists of a chain of flipflop connected in cascade.
- A flipflop receives a common clock pulse which causes the shift from one stage to another.

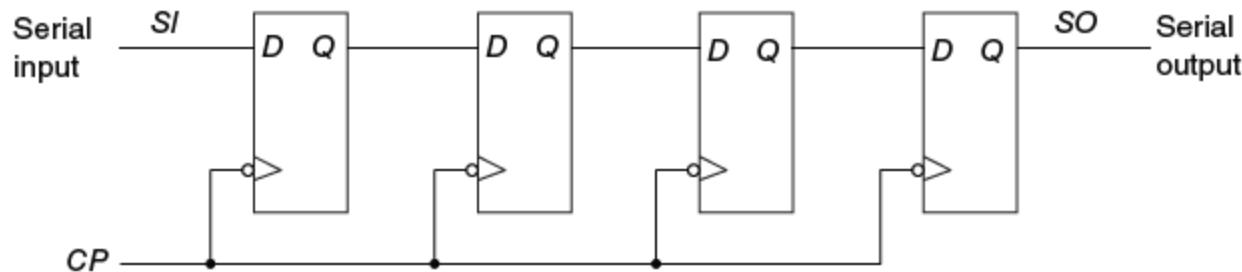
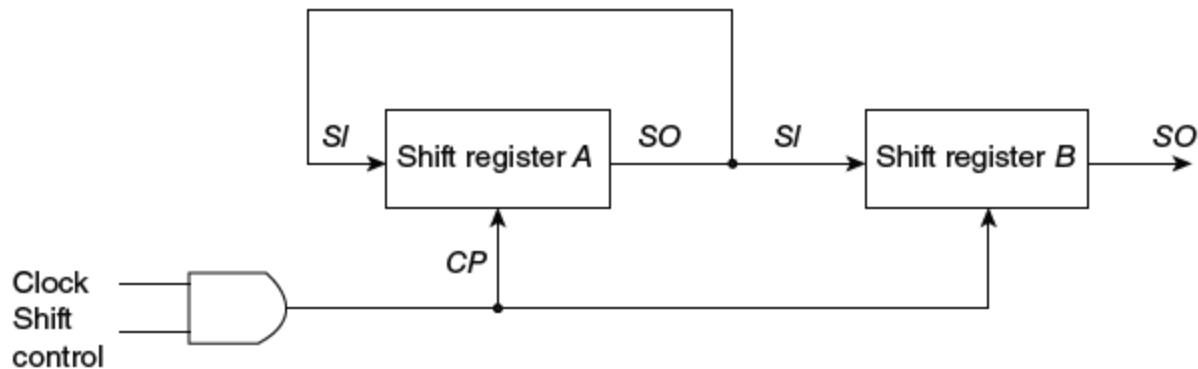


Figure 7-7 Shift register

## Serial transfer:

- A digital system is said to operate in a serial mode when information is transferred one bit at a time.
- The content of one register is transferred to another by shifting the bit from one register to another.
- The serial transfer of information from register A to register B is done with shift register as shown in block diagram.
- The serial output of register A (SO) goes to serial input of register A (Si) of register B
- To prevent the loss of information stored in the source register, the A register is made to circulate the information by connecting the serial output to the input terminal.



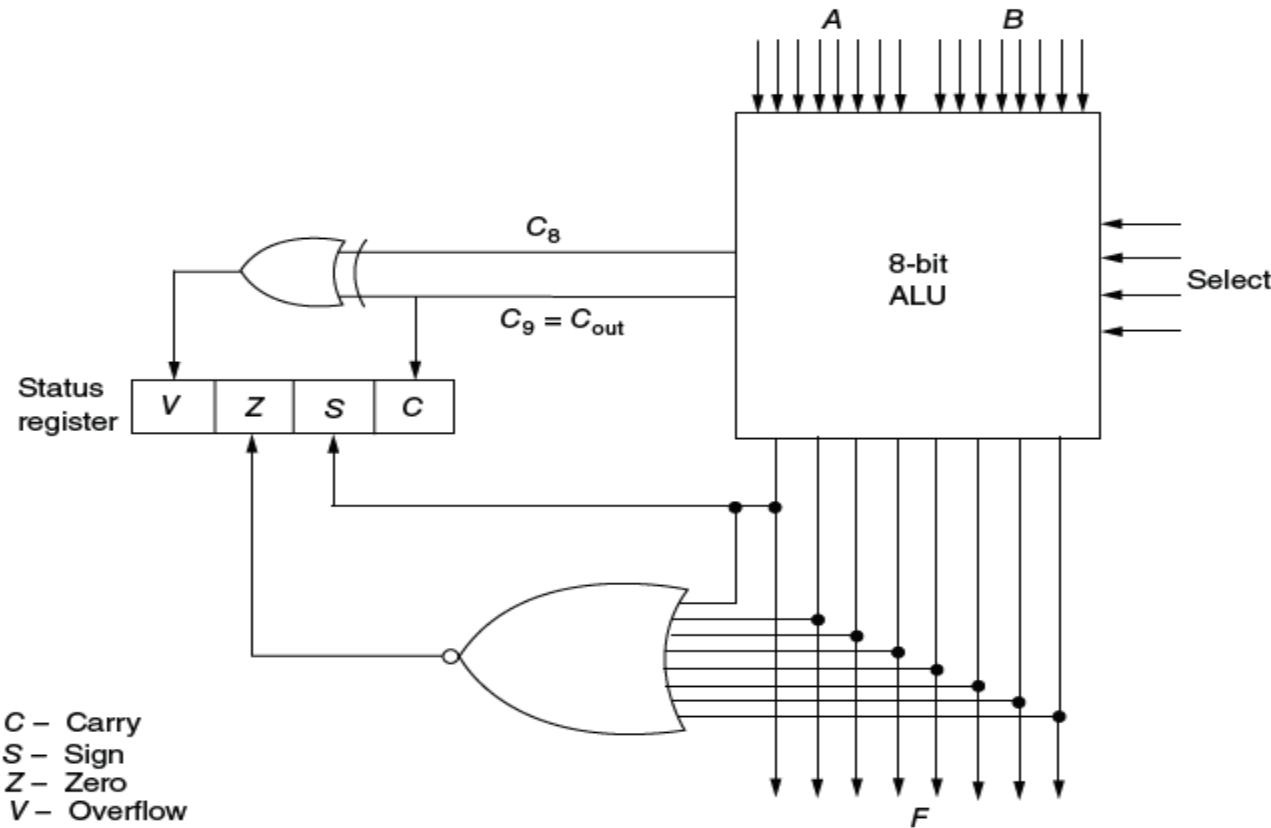
(a) Block diagram

- Assume the binary content of A before the shift is 1011 and that of B is 0010.
- The serial transfer from A to B will occur in 4 steps as shown in table.
- After the first pulse  $T_1$  the rightmost bit of A is shifted into the leftmost bit of B and at the same time , this bit is circulated into the leftmost position of A.
- Once the bit of A and B are shift to right, the previous serial output from B is lost.

Timing pulse	Shift register A	Shift register B	Serial output of B
Initial value	1 0 1 1	0 1 1 0	0
After $T_1$	1 1 0 1	1 0 0 1	1
After $T_2$	1 1 1 0	1 1 0 0	0
After $T_3$	0 1 1 1	0 1 1 0	0
After $T_4$	1 0 1 1	1 0 1 1	1

## Status register:

Figure below shows the block diagram of an 8 bit Alu with a 4 bit status register.



- The four status bits or flag bits are symbolized by C , S , Z and V.
- The bit are set or cleared as a result of an operation performed in the ALU.

1. Bit  $C$  is set if the output carry of the ALU is 1. It is cleared if the output carry is 0.
2. Bit  $S$  is set if the highest-order bit of the result in the output of the ALU (the sign bit) is 1. It is cleared if the highest-order bit is 0.
3. Bit  $Z$  is set if the output of the ALU contains all 0's, and cleared otherwise.  $Z = 1$  if the result is zero, and  $Z = 0$  if the result is nonzero.
4. Bit  $V$  is set if the exclusive-OR of carries  $C_8$  and  $C_9$  is 1, and cleared otherwise. This is the condition for overflow when the numbers are in sign-2's-complement representation (see Section 8-6). For the 8-bit ALU,  $V$  is set if the result is greater than 127 or less than - 128.