CARDAMOM PLANTERS' ASSOCIATION COLLEGE
(Re-Accredited With 'A' Grade By NAAC)
Pankajam Nagar, Bodinayakanur - 625 582.

DEPARTMENT OF CS & IT

Unit -3

Combinational Logic: Multiplexers – Demultiplexers – Decoders – Encoders – Code Converters–Parity Generators and Checkers

3.1 Introduction

Digital logic circuits are the basic building blocks of modern electronic systems. They process **binary data (0 and 1)** using logic gates such as:

- **AND**
- **OR**
- **NOT**
- (also, NAND, NOR, XOR, XNOR)

These circuits are widely used in devices like computers, mobile phones, calculators, and digital control systems.

Functions of Digital Logic Circuits

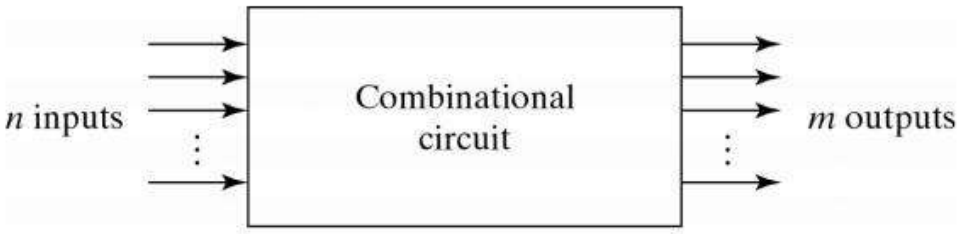
- Data processing
- Data storage
- Decision making
- Control operations

3.1.1. Classification of Digital Logic Circuits

Digital logic circuits are mainly classified into two types:

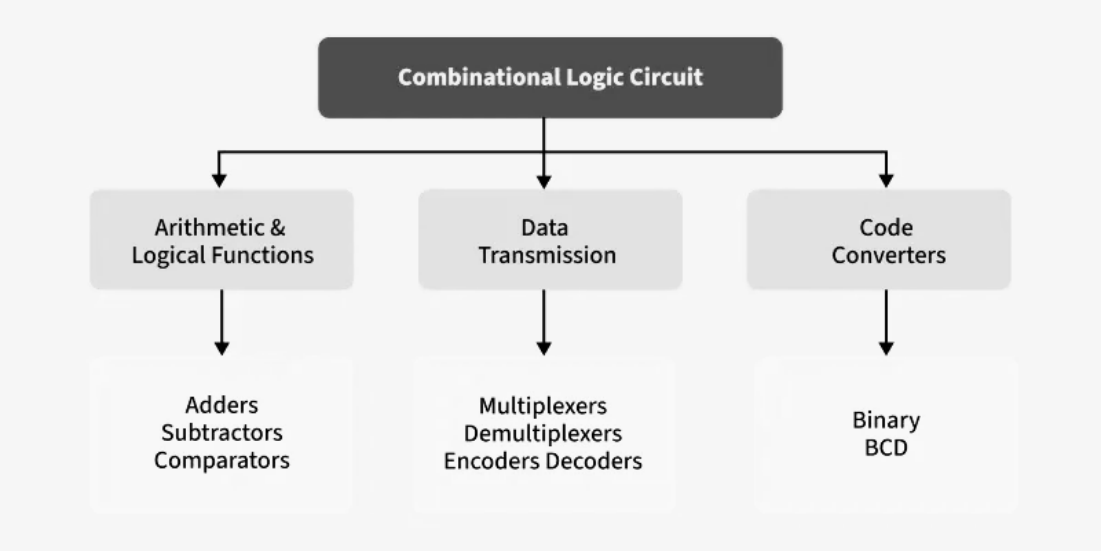
1. Combinational Circuits

- Output depends **only on present input**.
- No memory element.
- No feedback.
- Faster operation.



Examples:

- Multiplexer (MUX)
- Demultiplexer (DEMUX)
- Encoder
- Decoder
- Adder and Subtractor



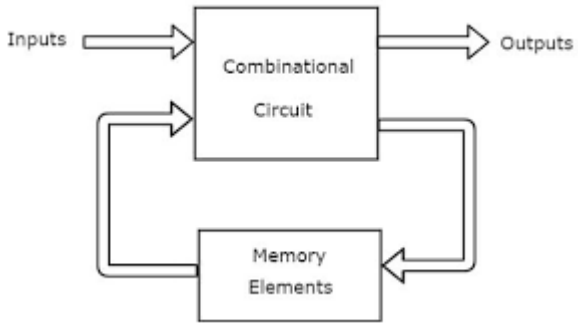
👉 **Example**: In a full adder, output (Sum and Carry) depends only on current input bits.

2. Sequential Circuits

- Output depends on **present input + previous state (past input)**.
- Contains **memory elements** like flip-flops.
- Uses feedback.

Sequential circuits are built using:

- Combinational logic
- Memory elements (Flip-Flops)



Examples:

- Flip-flops (SR, JK, D, T)
- Registers
- Counters

👉 **Example:** A counter remembers the previous count value before moving to the next count.

3.1.2. Difference Between Combinational and Sequential Circuits

Parameters	**Combinational Circuit**	**Sequential Circuit**
Meaning & Definition	A circuit that generates output based only on the **present input**. It is independent of time.	A circuit whose output depends on **present input as well as previous input (past state)**.
Feedback	No feedback is required. Output does not depend on previous state.	Feedback is required. Previous output is fed back as input for next operation.
Performance	Faster operation because it depends only on current input.	Slower compared to combinational circuits because it depends on past and present inputs.
Complexity	Less complex. No memory or clock required.	More complex due to memory elements, feedback, and clock signals.
Elementary Blocks	Built using **logic gates** (AND, OR, NOT, etc.).	Built using **flip-flops** along with logic gates.
Operation	Used for **Boolean and arithmetic operations**.	Mainly used for **data storage and state-based operations**.
Memory	No memory element.	Contains memory elements.
Clock Signal	No clock required.	Usually requires a clock signal.

Examples	Adder, Subtractor, MUX, DEMUX, Encoder, Decoder	Flip-flops, Registers, Counters

3.2. Multiplexer

A **Multiplexer (MUX)** is a **combinational circuit** that has **multiple data input lines and a single output line**.
The output depends on the **select (control) inputs**.

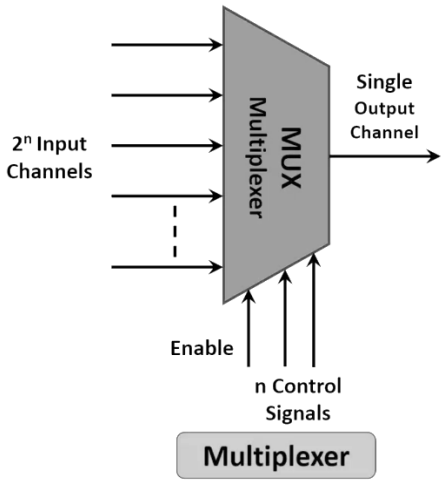
For:

- **N input lines → $\log_2(N)$ select lines required**
- **2^n input lines → n select lines required**

Alternative Names of Multiplexer

A multiplexer is also known as:

- **N-to-1 Selector**
- **Many-to-One Circuit**
- **Parallel-to-Serial Converter**
- **Universal Logic Circuit**



Working Principle

The select lines determine which input is connected to the output at a particular time.

Example:
In a **4-to-1 MUX**:

- Inputs = 4
- Select lines = 2

- Output = 1

Since $2^2 = 4$, two select lines are enough to choose one of the four inputs.

Types of Multiplexers

- 2:1 MUX → 2^1 → 1 select line
- 4:1 MUX → 2^2→ 2 select lines
- 8:1 MUX → 2^3→ 3 select lines
- 16:1 MUX → 2^4→ 4 select lines

Applications of Multiplexer

- In communication systems to transmit multiple signals over a single channel
- In computer systems for data routing
- In implementing Boolean functions
- In time-division multiplexing
- In bandwidth utilization improvement

3.2.1. 2 × 1 Multiplexer (2-to-1 MUX)

A **2×1 Multiplexer** is the most basic and fundamental type of multiplexer.
It selects **one signal from two input lines** and transmits it to a **single output line**.

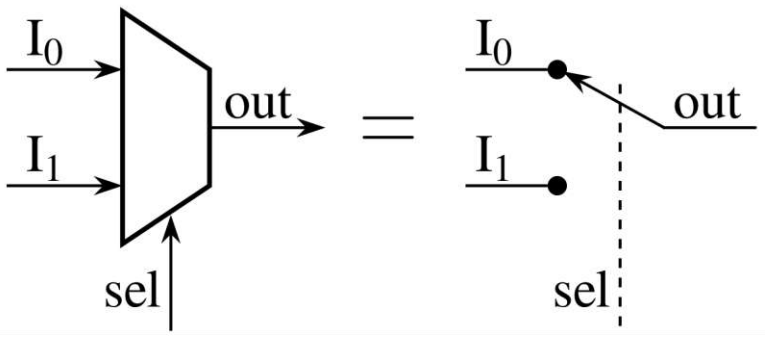
It is also called a **2-to-1 Multiplexer**.

Structure of 2×1 MUX

- **Inputs**: I_0, I_1
- **Select Line**: S_0
- **Output**: Y

Since there are 2 inputs, only **1 select line** is required because:

$\log_2(2) = 1$ *or* 2^1



Working Principle

The output depends on the select line S_0:

- When **S_0 = 0 (Low)** → Input **I_0** is selected
- When **S_0 = 1 (High)** → Input **I_1** is selected

Truth Table

S_0	**Output (Y)**
0	I_0
1	I_1

Logical Expression of 2×1 MUX

From the truth table, the Boolean expression is:

$$Y = \bar{S_0} \cdot I_0 + S_0 \cdot I_1$$

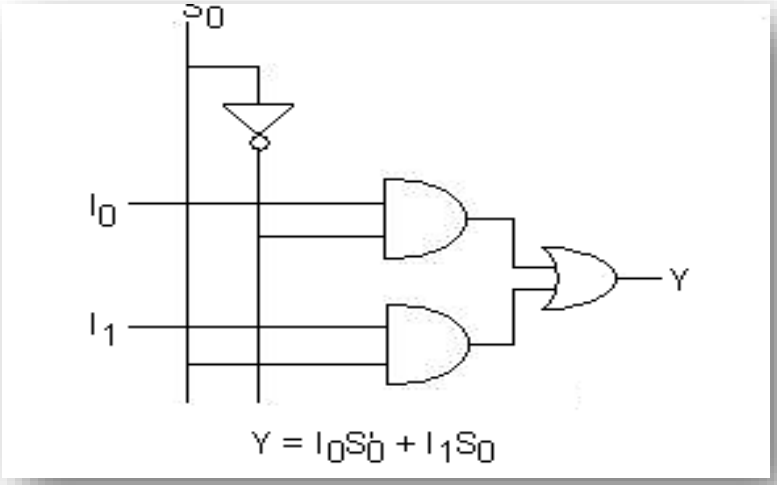
Where:

- $\bar{S_0}$= Complement of select line
- ·= AND operation
- += OR operation

Logic Gate Implementation

The 2×1 MUX can be implemented using:

- 2 AND gates
- 1 NOT gate
- 1 OR gate



Applications

- In **microprocessors** to select between two data sources
- Selecting between two instructions
- Data routing in digital systems
- Used as a building block for larger multiplexers

3.2.2. 4 × 1 Multiplexer (4-to-1 MUX)

A **4×1 Multiplexer** (4-to-1 MUX) is a combinational circuit that has:

- **4 input lines** → I_0, I_1, I_2, I_3
- **1 output line** → Y
- **2 select lines** → S_1, S_0

It selects **one of the four inputs** and sends it to the output based on the select lines.

Number of Selection Lines

The number of select lines is determined by:

$$\log_2(n)$$

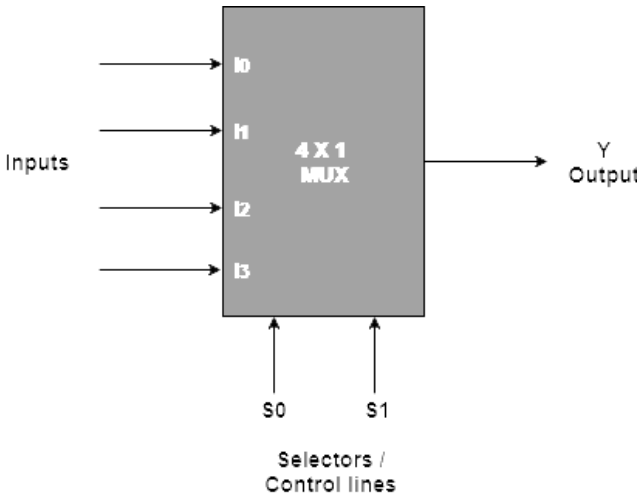
Where:

- n = number of inputs

For 4 inputs:

$$\log_2(4) = 2$$

So, **2 select lines are required**.



Working Principle

The output depends on the binary value of the select lines (S_1 S_0):

- When S_1 S_0=00, the input I0 is selected.
- When S_1 S_0=01, the input I1 is selected.
- When S_1 S_0=10, the input I2 is selected.
- When S_1 S_0=11, the input I3 is selected.

S_1	S_0	Selected Input	Output (Y)
0	0	I_0	I_0
0	1	I_1	I_1
1	0	I_2	I_2
1	1	I_3	I_3

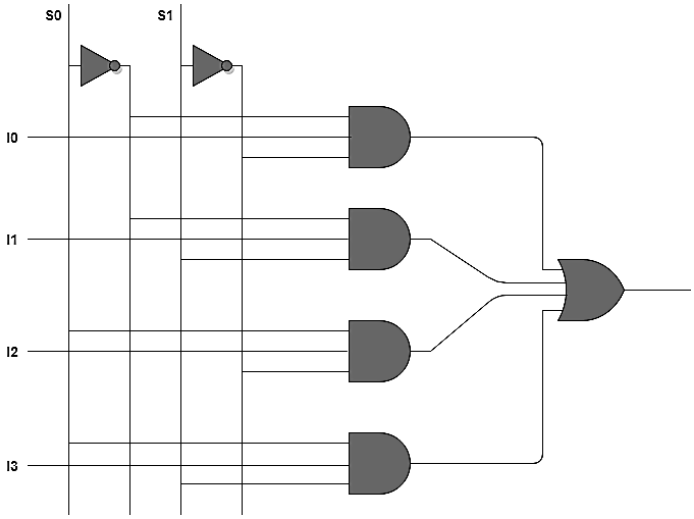
Logical Expression of 4×1 MUX

$$Y = I_0\bar{S_1}\bar{S_0} + I_1\bar{S_1}S_0 + I_2S_1\bar{S_0} + I_3S_1S_0$$

Logic Gate Implementation

A 4×1 MUX can be implemented using:

- 4 AND gates
- 2 NOT gates
- 1 OR gate



Applications

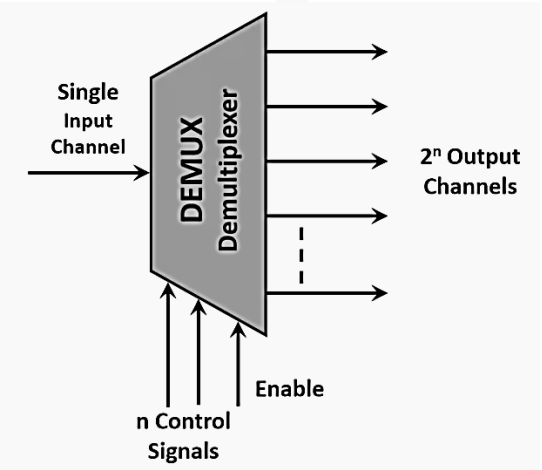
- Data selection in digital circuits
- CPU data routing
- Communication systems
- Used to implement Boolean functions
- Used as a building block for 8×1 and higher multiplexers

3.3. Demultiplexer

A **Demultiplexer (DEMUX)** is a **combinational circuit** that takes a **single input** and distributes it to one of many output lines based on the select lines.

It is also called a:

- **1-to-N Selector**
- **Data Distributor**
- **Serial-to-Parallel Converter**



Step-by-Step Working

1. The circuit has:
    - One input (D)
    - n Select lines (S_1, S_0, etc.)
    - Multiple outputs (Y_0, Y_1, Y_2, Y_3…)
2. The select lines determine **which output line will receive the input signal**.
3. At any time:
    - Only **one output line** becomes active.
    - All other output lines remain 0 (LOW).

Types of Demultiplexers

- 1:2 DEMUX → 2^1 → 1 select line
- 1:4 DEMUX → 2^2→ 2 select lines
- 1:8 DEMUX → 2^3→ 3 select lines
- 1:16 DEMUX → 2^4→ 4 select lines

Applications

- Communication systems
- Data routing
- Address decoding
- Memory systems
- Serial-to-parallel data conversion

3.3.1 1x2 Demultiplexer

A **1-to-2 Demultiplexer** has:

- **1 Input bit (I)**
- **1 Select line (S)** – also called control bit
- **2 Output lines (Y_0, Y_1)**

It distributes one input signal to one of the two output lines based on the value of the select line.

Working Principle

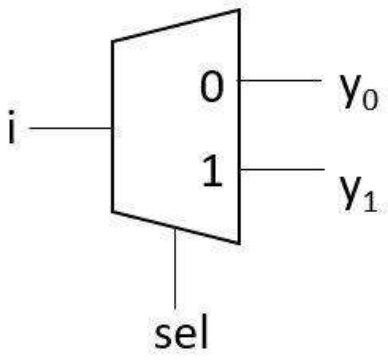
- The input bit **I** is transmitted to one of the output lines.

- The output depends on the value of select line **S**.

Operation:

- If **S = 0** → Input **I** is connected to **Y_0**
- If **S = 1** → Input **I** is connected to **Y_1**

Only one output is active at a time.



Truth Table

S	Y_0	Y_1
0	I	0
1	0	I

Boolean Expressions

$$Y_0 = I \cdot \bar{S}$$
$$Y_1 = I \cdot S$$

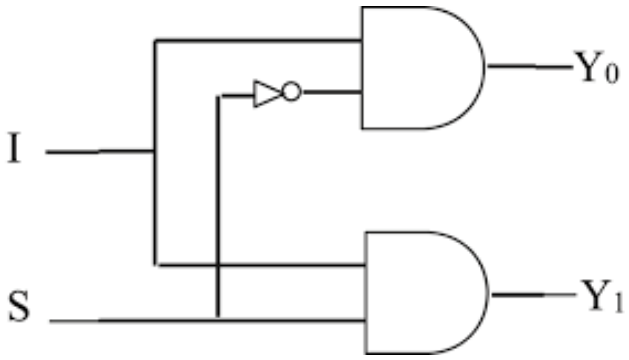
Where:

- $\bar{S}$= Complement of S
- ·= AND operation

Logic Gate Implementation

The circuit can be implemented using:

- 1 NOT gate
- 2 AND gates



3.3.2. 1x4 Demultiplexer

A **1×4 Demultiplexer** is a combinational circuit that has:

- **1 Input** → D
- **2 Select lines** → S_1, S_0
- **4 Outputs** → Y_0, Y_1, Y_2, Y_3

It distributes the single input to one of four outputs depending on the select lines.

Number of Select Lines

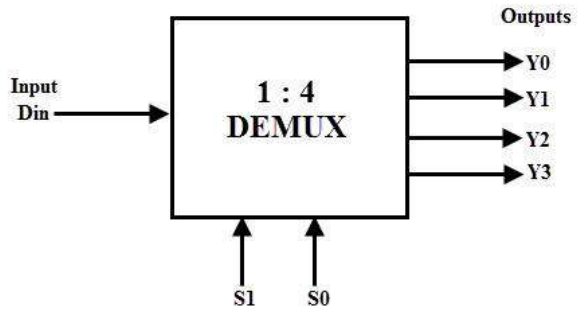
The number of select lines is determined by:

$$2^n = \text{Number of outputs}$$

For 4 outputs:

$$2^2 = 4$$

So, **2 select lines** are required.



Working Principle

The output depends on the binary value of **S_1 S_0**.

Case 1: S_1S_0 = 00

- Output **Y_0 = D**
- $Y_1 = 0$

- $Y_2 = 0$
- $Y_3 = 0$

Case 2: $S_1S_0 = 01$

- Output **$Y_1 = D$**
- $Y_0 = 0$
- $Y_2 = 0$
- $Y_3 = 0$

Case 3: $S_1S_0 = 10$

- Output **$Y_2 = D$**
- $Y_0 = 0$
- $Y_1 = 0$
- $Y_3 = 0$

Case 4: $S_1S_0 = 11$

- Output **$Y_3 = D$**
- $Y_0 = 0$
- $Y_1 = 0$
- $Y_2 = 0$

Operation Table:

S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

Only one output receives the input at a time.

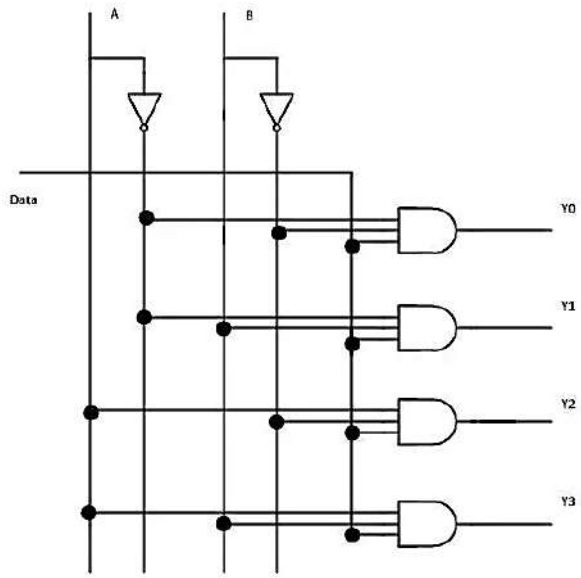
Boolean Expressions

$$Y_0 = D\bar{S_1}\bar{S_0}$$
$$Y_1 = D\bar{S_1}S_0$$
$$Y_2 = DS_1\bar{S_0}$$
$$Y_3 = DS_1S_0$$

Logic Gate Implementation

The 1×4 DEMUX can be implemented using:

- 2 NOT gates
- 4 AND gates



Applications

- Data distribution in communication systems
- Address decoding
- Memory selection
- Serial-to-parallel conversion

Difference Between Multiplexer and Demultiplexer

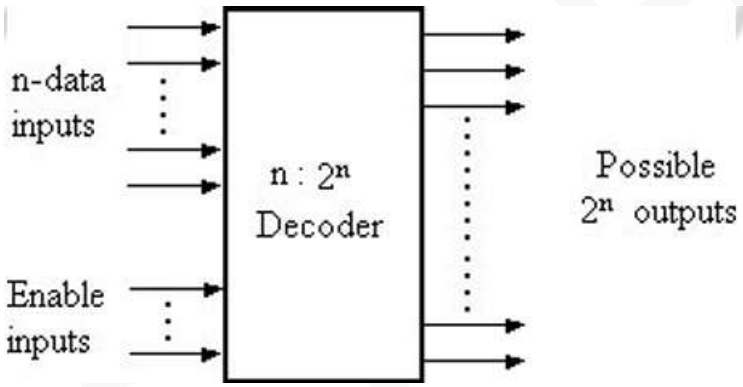
Multiplexer (MUX)	**Demultiplexer (DEMUX)**
A combinational circuit that uses multiple inputs to generate a single output.	A combinational circuit that uses a single input and directs it to multiple outputs.
Has **many inputs and one output**.	Has **one input and many outputs**.
Called a **Data Selector**.	Called a **Data Distributor**.
It is a digital switch.	It is a digital circuit.
Works on **many-to-one** principle.	Works on **one-to-many** principle.
Used for **parallel-to-serial conversion**.	Used for **serial-to-parallel conversion**.
Used at the **transmitter end** in TDM.	Used at the **receiver end** in TDM.

Also called **MUX**.	Also called **DEMUX**.
Select lines choose the input to be sent to output.	Select lines choose which output receives the input.
Improves communication efficiency by combining signals.	Separates the combined signal into original form at receiver.
Examples: 8:1 MUX, 16:1 MUX, 32:1 MUX.	Examples: 1:8 DEMUX, 1:16 DEMUX, 1:32 DEMUX.

3.4. Decoder

A **Decoder** is a **combinational logic circuit** that converts an **N-bit binary input code** into a maximum of **2^n output lines**, such that:

- For each input combination, **only one output line is active (HIGH)**.
- All other output lines remain **inactive (LOW)**.
- The process performed by a decoder is called **Decoding**.



Types of Decoders

Decoders are classified based on the number of input lines (N) and output lines (2^n).

Common types of decoders are:

- 2-to-4 Decoder (Binary decoder)
- 3-to-8 Decoder (Binary to Octal decoder)
- 4-to-16 Decoder
- BCD to Seven Segment decoder

Applications of Decoder

- Memory address decoding
- Instruction decoding in microprocessors

- Seven-segment display
- Data routing
- Code conversion

3.4.1. 2-to-4 Decoder (Binary Decoder)

The 2 to 4 decoder is one that has 2 input lines and 4 (2^2) output lines.

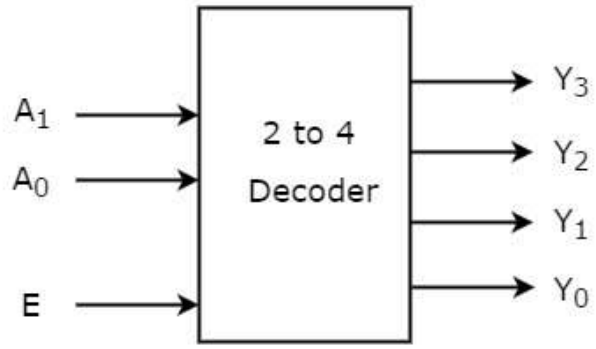
- Inputs: A, B (2 inputs)
- Outputs: Y_0, Y_1, Y_2, Y_3 (4 outputs)

Since:

$$2^2 = 4$$

Working Principle

- The 2-bit binary input selects one of the four outputs.
- For each input combination, only one output is HIGH.



Operation

When Enable (E) = 0 (Inactive)

- The decoder is **disabled**.
- All AND gates are OFF.
- All outputs remain **0**.

$$Y_0 = Y_1 = Y_2 = Y_3 = 0$$

No output is selected.

When Enable (E) = 1 (Active)

- The decoder is **enabled**.
- Based on inputs A and B, only one output becomes HIGH (1).

Operation Based on Inputs:

When A = 0 and B = 0

- AND gate 1 becomes active.
- Output **$Y_0 = 1$**
- Other outputs = 0

When A = 0 and B = 1

- AND gate 2 becomes active.
- Output **$Y_1 = 1$**
- Other outputs = 0

When A = 1 and B = 0

- AND gate 3 becomes active.
- Output **$Y_2 = 1$**
- Other outputs = 0

When A = 1 and B = 1

- AND gate 4 becomes active.
- Output **$Y_3 = 1$**
- Other outputs = 0

Truth Table with Enable

E	A	B	Y_0	Y_1	Y_2	Y_3
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

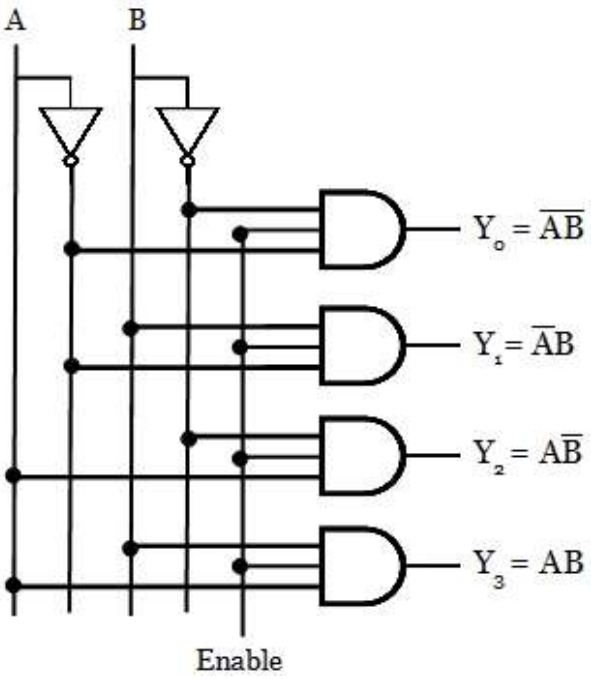
Boolean Expressions

$$Y_0 = E \cdot \bar{A} \cdot \bar{B}$$

$$Y_1 = E \cdot \bar{A} \cdot B$$

$$Y_2 = E \cdot A \cdot \bar{B}$$

$$Y_3 = E \cdot A \cdot B$$



3.4.2. 3-to-8 Line Decoder

A **3-to-8 decoder** is a combinational logic circuit that has:

- **3 input lines** → A, B, C
- **8 output lines** → Y_0 to Y_7

Based on the three inputs, **one of the eight outputs is selected (HIGH)** at a time.

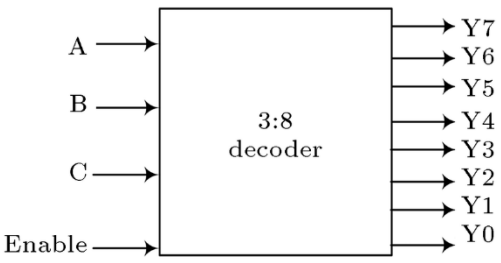
Since: $2^3 = 8$

Three inputs are decoded into eight outputs.

Working Principle

- The 3 input variables (A, B, C) represent a 3-bit binary number.
- Each output corresponds to one **minterm** of the three input variables.
- Only **one output becomes HIGH (1)** for any given input combination.
- All other outputs remain LOW (0).

👉 The outputs are **mutually exclusive** because only one output can be 1 at any time.



Truth Table

Inputs				Outputs							
EN	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Minterm Representation

Each output represents one minterm:

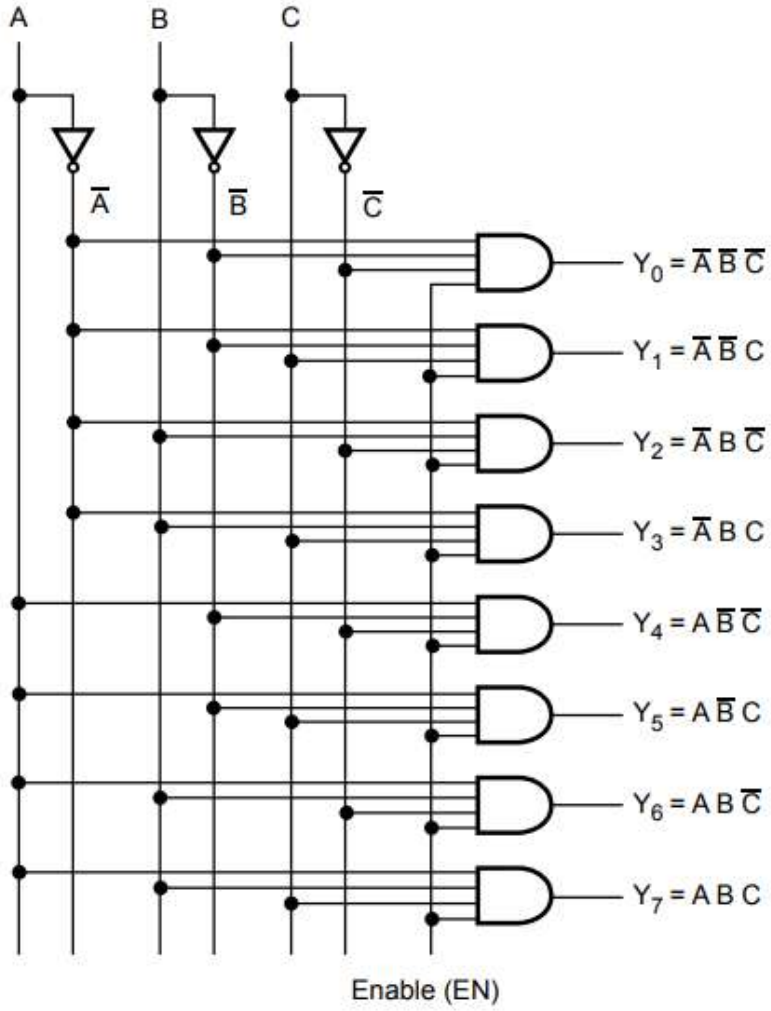
$$Y_0 = A'B'C'$$
$$Y_1 = A'B'C$$
$$Y_2 = A'BC'$$
$$Y_3 = A'BC$$
$$Y_4 = AB'C'$$
$$Y_5 = AB'C$$
$$Y_6 = ABC'$$
$$Y_7 = ABC$$

Binary-to-Octal Conversion

- The 3 input bits represent a **binary number (000–111)**.
- The outputs correspond to **octal digits (0–7)**.
- The output line that becomes HIGH represents the octal equivalent of the binary input.

Example:

- Input = 101 (binary 5)
- Output $Y_5 = 1$
- Corresponds to octal digit 5



Applications

- Binary-to-octal conversion
- Memory address decoding
- Instruction decoding
- Code conversion
- Data routing

3.4.3. BCD to 7-Segment Display Decoder

A **BCD to 7-Segment Decoder** is a combinational logic circuit that converts a **4-bit Binary Coded Decimal (BCD) input** into signals required to drive a **7-segment display**.

It converts decimal digits (0–9) into the corresponding 7-segment display format.

Inputs and Outputs

1. Inputs (4-bit BCD)

- A (MSB)
- B
- C
- D (LSB)

Since:

$$2^4 = 16$$

But only **10 combinations (0000–1001)** are valid BCD numbers (0–9).

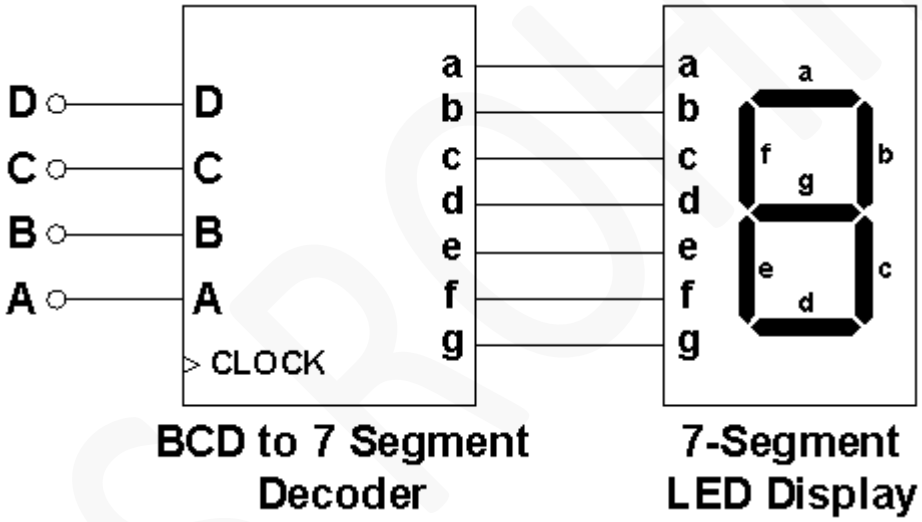
2. Outputs (7 segments)

- a, b, c, d, e, f, g

Each output controls one segment of the display.

7-Segment Display Structure

The display consists of 7 LED segments:



Working Principle

- The 4-bit BCD input represents a decimal number (0–9).
- The decoder activates the required segments to display that digit.
- Each digit requires a specific combination of segments to glow.

Example

For Decimal 0 (BCD = 0000)

Segments ON → a, b, c, d, e, f
Segment OFF → g

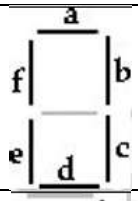
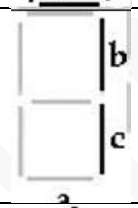
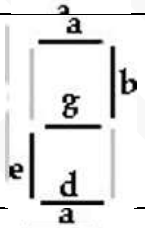
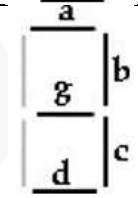
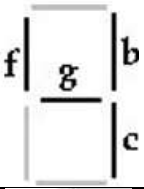
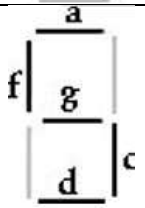
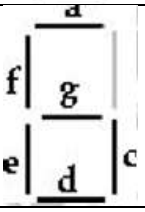
For Decimal 1 (BCD = 0001)

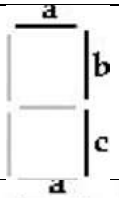
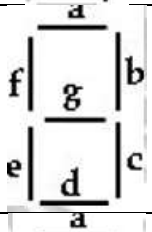
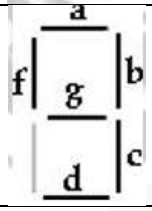
Segments ON → b, c
Other segments OFF

For Decimal 2 (BCD = 0010)

Segments ON → a, b, d, e, g

Truth Table (Partial)

Decimal	A B C D	Segments ON	
0	0000	a b c d e f	
1	0001	b c	
2	0010	a b d e g	
3	0011	a b c d g	
4	0100	b c f g	
5	0101	a c d f g	
6	0110	a c d e f g	

7	0111	a b c	
8	1000	a b c d e f g	
9	1001	a b c d f g	

Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

Types of 7-Segment Displays

A 7-segment display consists of 7 LED segments (a–g) used to display decimal digits (0–9). There are two main types:

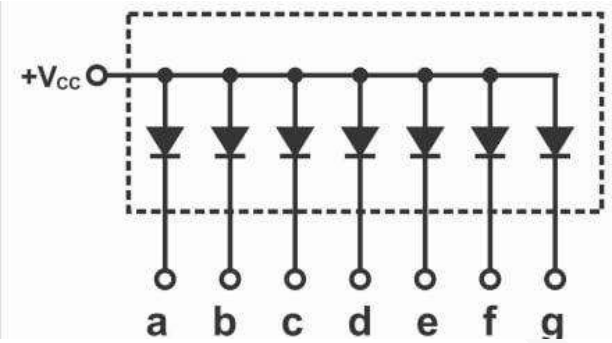
- **Common Anode**
- **Common Cathode**

Common Anode (CA)

In a **Common Anode display**, all the **anodes (positive terminals)** of the 7 LEDs are connected together and tied to **+V (power supply)**.

Working Principle

- The common terminal is connected to **HIGH (+V)**.
- A segment glows when its corresponding input is given **LOW (0)**.
- It works on **active LOW logic**.
    - To turn ON a segment → Apply **0 (LOW)**
    - To turn OFF a segment → Apply **1 (HIGH)**



Key Point

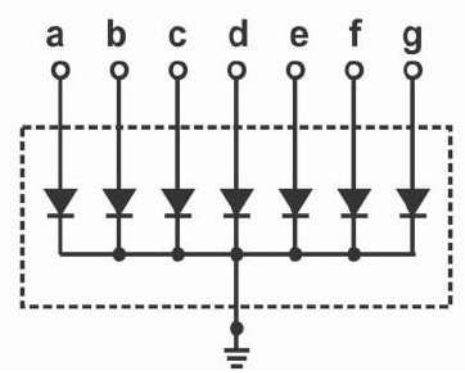
Segment lights when **current flows from +V through the LED to ground**.

Common Cathode (CC)

In a **Common Cathode display**, all the **cathodes (negative terminals)** of the 7 LEDs are connected together and tied to **Ground (0V)**.

Working Principle

- The common terminal is connected to **Ground**.
- A segment glows when its corresponding input is given **HIGH (1)**.
- It works on **active HIGH logic**.
    - To turn ON a segment → Apply **1 (HIGH)**
    - To turn OFF a segment → Apply **0 (LOW)**



Key Point

Segment lights when **current flows from input through LED to ground**.

Applications

- Digital clocks
- Calculators
- Counters
- Digital meters
- Elevator display systems

3.4.4. 4-to-16 Decoder

A **4-to-16 decoder** is a combinational logic circuit that converts **4 input lines** into **16 output lines**.
For every unique 4-bit input combination, **only one output becomes HIGH (1)** and all others remain LOW (0).

👉 It is also called a **4-line to 16-line decoder**.

Inputs and Outputs

- **Inputs**: A, B, C, D (4 inputs)
- **Outputs**: Y_0 to Y_{15} (16 outputs)
- Optional: Enable (E)

Working Principle

- The 4 input bits represent a **binary number (0000 to 1111)**.
- The decoder converts this binary number into **one active output line**.
- Only the output corresponding to the input combination becomes HIGH.
- All other outputs remain LOW.
- Outputs are **mutually exclusive** (only one output is active at a time).

Truth table

binary inputs				outputs														
A_3	A_2	A_1	A_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Example

If input = **1010 (10 in decimal)**

☞ Output $Y_{10} = 1$, all others = 0

Logic Expression

Each output represents a **minterm** of 4 variables.

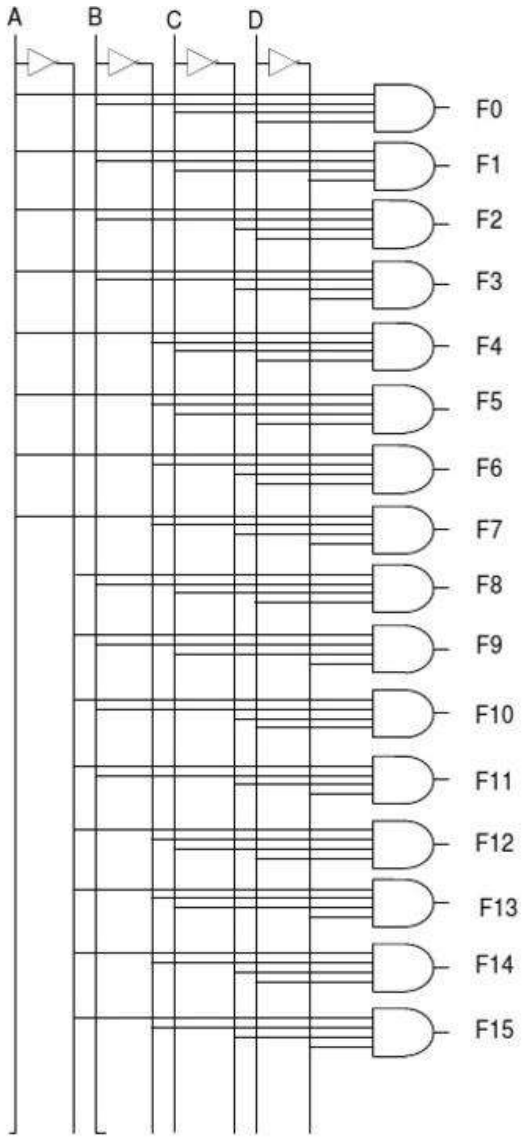
For example:

- Y_0 = A'B'C'D'
- Y_1 = A'B'C'D
- Y_2 = A'B'CD'
- ...
- Y_{15} = ABCD

So, the decoder generates all **16 minterms** of 4 variables.

Block Diagram Explanation

- 4 inputs enter the decoder.
- Internally it uses **AND gates** and **NOT gates**.
- Each AND gate generate one minterm.
- 16 AND gates produce 16 outputs.



Applications

- Memory address decoding
- Instruction decoding
- Data routing
- Demultiplexing
- Digital system design

3.5. Encoder

An **Encoder** is a **combinational logic circuit** that converts **multiple input lines into a coded binary output**.

In simple words, an encoder converts information from **normal (human-readable) form to binary (machine-readable) form**.
This process is called **Encoding**.

Basic Concept

- An encoder has **2^n input lines**
- And **n output lines**
- Only **one input is active at a time**
- The output gives the **binary code of the active input**

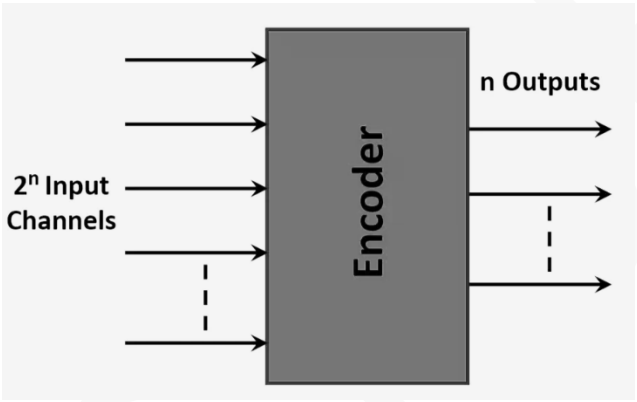
👉 It performs the **reverse operation of a Decoder**

Working Principle

- When one input line becomes HIGH (1),
- The encoder generates the **binary code** corresponding to that input position.
- Outputs represent the **binary number** of the active input.

Example:
If input line 5 is active → Output will be binary 101

Block Diagram Explanation



Example:
4 inputs → Encoder → 2 outputs

Types of Encoders

- 4 to 2 Encoder
- Octal to Binary Encoder (8 to 3 Encoder)
- Decimal to BCD Encoder
- Priority Encoder

Applications

- Data transmission
- Digital communication systems

- Keyboard encoding
- Interrupt handling
- Signal processing
- Automation systems

3.5.1. 4-to-2 Encoder

A **4-to-2 Encoder** is a combinational circuit that converts **4 input lines (D_3, D_2, D_1, D_0)** into **2 output lines (Y_1, Y_0)**.

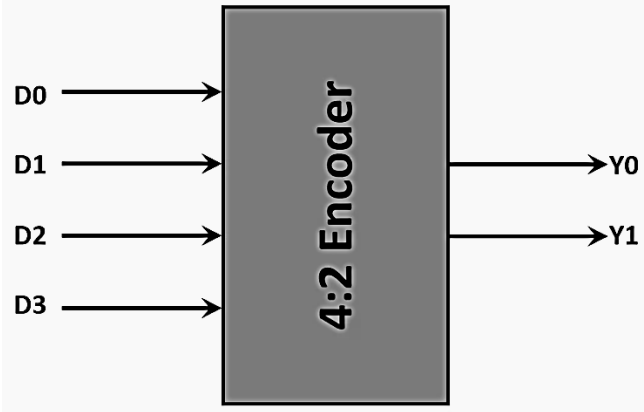
At any time, **only one input is HIGH (1)**, and the outputs give the **binary code** of the active input.

Number of inputs = $2^2 = 4$
Number of outputs = 2

Working Principle

- The encoder checks which input line is HIGH.
- It produces the **binary equivalent** of that input position at the output.
- Only one input should be active at a time.
- If more than one input is HIGH, output becomes undefined (in basic encoder).

Truth Table



Input				Output	
D0	D1	D2	D3	Y0	Y1
1	0	0	0	0	0
0	1	0	0	1	0
0	0	1	0	0	1
0	0	0	1	1	1

Example

If $D_2 = 1$
Output → $Y_0Y_1 = 01$

If **D_3 = 1**
Output → **Y_0Y_1= 11**

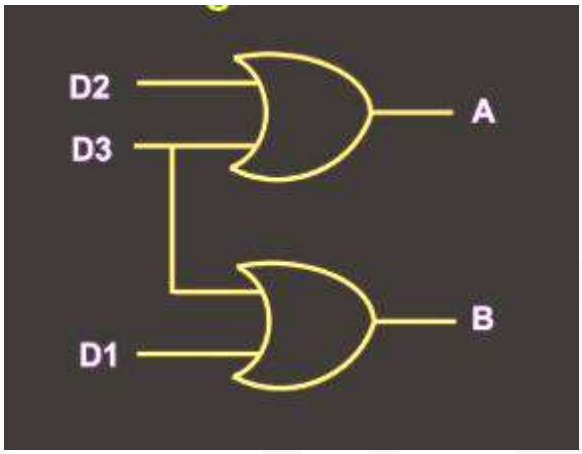
Boolean Expressions

From the truth table:

$$Y_1 = D_2 + D_3$$
$$Y_0 = D_1 + D_3$$

Logic Circuit Explanation

- Two OR gates are used.
- Y_1 is connected to D_2 and D_3.
- Y_0 is connected to D_1 and D_3.
- D_0 does not connect to any gate because its binary code is 00.



3.5.2. Octal to Binary Encoder (8-to-3 Encoder)

An **8-to-3 Encoder** (Octal to Binary Encoder) is a combinational circuit that converts **8 input lines (D_7 to D_0)** into **3 output lines (Y_2, Y_1, Y_0)**.

Each input represents an **octal digit (0–7)**, and the outputs generate the corresponding **3-bit binary code**.

👉 Only **one input should be HIGH at a time**.

Inputs and Outputs

- **Inputs:** D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0
- **Outputs:** Y_2(A), Y_1(B), Y_0(C)

Number of inputs = $2^3 = 8$
Number of outputs = 3

Working Principle

- When one input line becomes HIGH (1),
- The encoder generates its **binary equivalent** on the output lines.
- Outputs represent the **binary number of the active input**.
- Only one output combination is active at a time.

Truth Table

No	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	1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	1
2	0	0	0	0	0	1	0	0	0	1	0
3	0	0	0	0	1	0	0	0	0	1	1
4	0	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	0	0	0	0	1	0	1
6	0	1	0	0	0	0	0	0	1	1	0
7	1	0	0	0	0	0	0	0	1	1	1

Logical Expressions

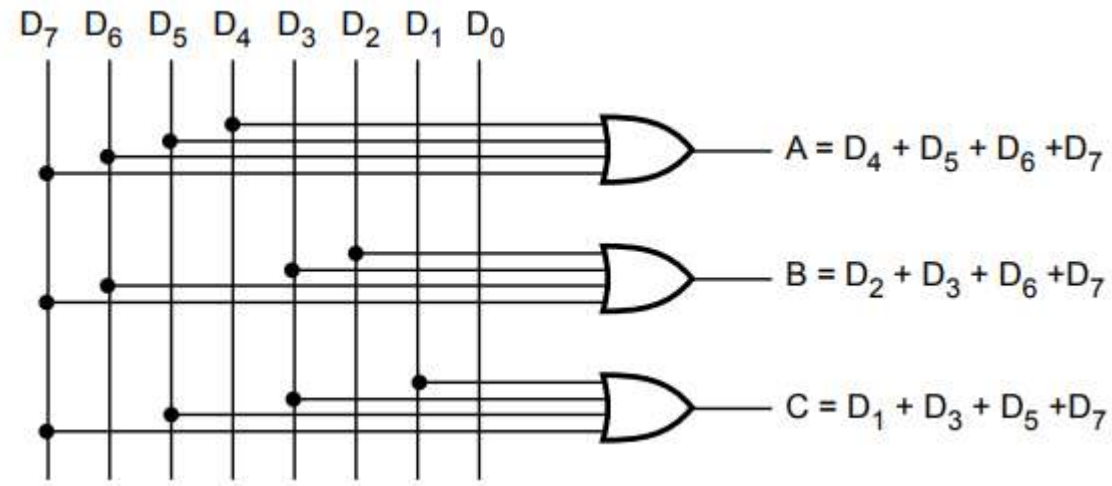
From the truth table:

$$Y_2 = D_7 + D_6 + D_5 + D_4$$
$$Y_1 = D_7 + D_6 + D_3 + D_2$$
$$Y_0 = D_7 + D_5 + D_3 + D_1$$

("+" represents OR operation)

Logic Circuit Explanation

- Three OR gates are used.
- Each output is connected to specific input lines based on binary weighting.
- The circuit generates the correct 3-bit binary code.



3.5.3. Decimal to BCD Encoder

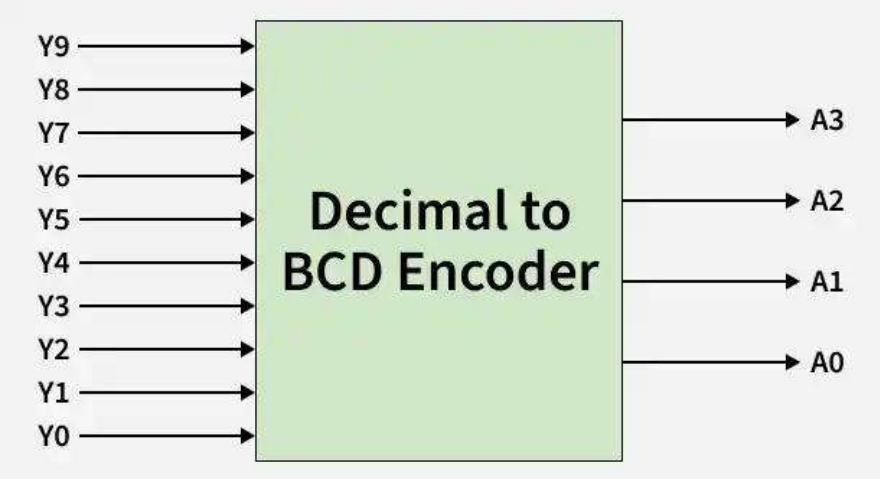
A **Decimal to BCD Encoder** is a combinational circuit that converts a **decimal digit (0–9)** into its equivalent **Binary Coded Decimal (BCD)** format.

In BCD, each decimal digit is represented using **4-bit binary code**.

Inputs and Outputs

- **Inputs:** D_0 to D_9 (10 input lines)
- **Outputs:** A_3, A_2, A_1, A_0 (4 output lines)

👉 Only **one decimal input** should be HIGH at a time.
👉 Output gives the **4-bit BCD equivalent**.



Working Principle

- When one decimal input line becomes HIGH,
- The encoder produces the corresponding **4-bit BCD code**.
- Each decimal digit (0–9) has a unique 4-bit representation.

BCD Code Table

Decimal Digit	A_3	A_2	A_1	A_0	BCD Code
0	0	0	0	0	0000
1	0	0	0	1	0001
2	0	0	1	0	0010
3	0	0	1	1	0011
4	0	1	0	0	0100
5	0	1	0	1	0101
6	0	1	1	0	0110
7	0	1	1	1	0111
8	1	0	0	0	1000
9	1	0	0	1	1001

Decimal Input										BCD Output			
D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0	1	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	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

Boolean Expressions

From the table:

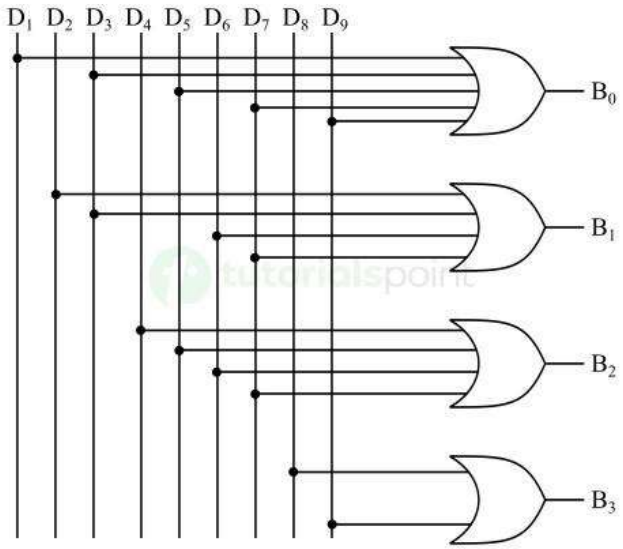
$$A_3 = D_8 + D_9$$
$$A_2 = D_4 + D_5 + D_6 + D_7$$
$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_0 = D_1 + D_3 + D_5 + D_7 + D_9$$

("+" represents OR operation)

Logic Circuit Explanation

- Four OR gates are used.
- Each output line is connected to specific decimal inputs based on BCD weighting.
- The circuit produces correct 4-bit BCD output.



Applications

- Digital calculators
- Digital clocks
- 7-segment display systems
- Digital counters
- Microprocessor systems

3.5.4. Priority Encoder

A **Priority Encoder** is a combinational circuit that encodes multiple inputs into binary output **with priority assigned to each input**.

If **two or more inputs are HIGH (1) at the same time**, the input with the **highest priority** will be encoded.

Need for Priority Encoder

In a normal encoder:

- Only one input should be HIGH.
- If multiple inputs are HIGH → output becomes undefined.

To solve this problem, a **priority encoder** is used.

4-to-2 Priority Encoder

Inputs:

D_3, D_2, D_1, D_0

Outputs:

x, y (binary output)
V (Valid bit)

👉 Higher subscript → Higher priority
👉 D_3 has highest priority
👉 D_0 has lowest priority

Working Principle

- If **$D_3 = 1$** → Output = 11 (regardless of other inputs)
- If **$D_2 = 1$** and $D_3 = 0$ → Output = 10
- If **$D_1 = 1$** and $D_3 = D_2 = 0$ → Output = 01
- If **$D_0 = 1$** and higher inputs are 0 → Output = 00
- If all inputs = 0 → V = 0 (no valid input)

Truth Table

D_3	D_2	D_1	D_0	x	y	V
0	0	0	0	-	-	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

(X = Don't care)

Valid Bit (V)

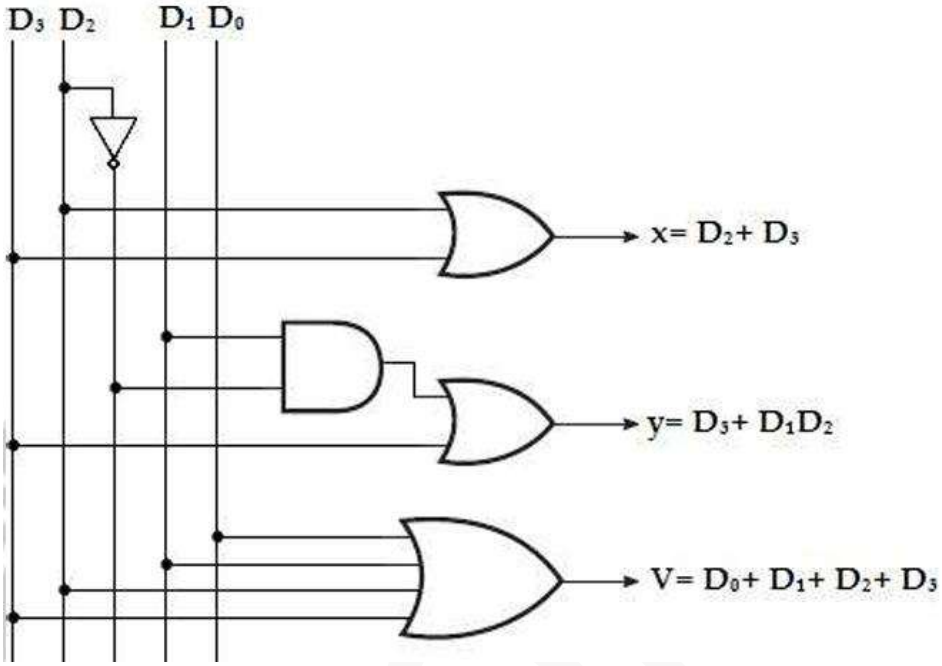
- **V = 1** → At least one input is HIGH
- **V = 0** → No input is active

Valid bit expression:

$$V = D_3 + D_2 + D_1 + D_0$$

Key Points

- Resolves multiple input problem
- Gives priority to highest input
- Provides valid output indicator
- Used in interrupt systems



Applications

- Interrupt controllers
- Microprocessor systems
- Bus arbitration
- Digital communication

3.6. Parity Generators and Checkers

Parity circuits are used for **error detection** in digital communication systems.

3.6.1. Parity Generator

A **Parity Generator** is a combinational circuit that adds an extra bit (parity bit) to data to make the total number of 1's either:

- **Even Parity** → Total number of 1's is even
- **Odd Parity** → Total number of 1's is odd

3.6.1.1. Even Parity Generator

Rule:

Total number of 1's (including parity bit) must be **even**.

Working:

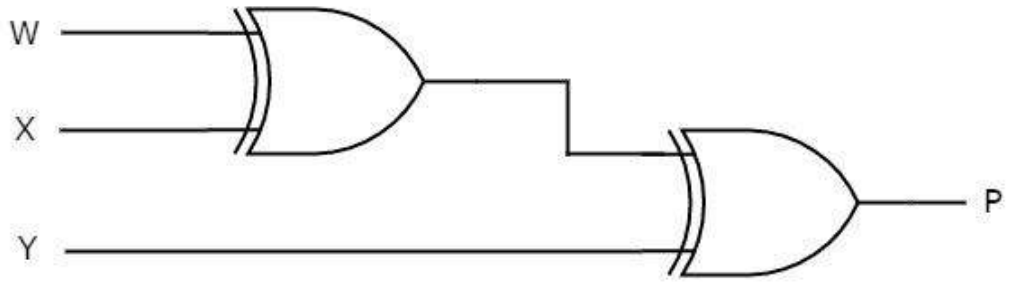
1. All data bits are applied to an **XOR gate**.
2. XOR output gives the **even parity bit**.
3. If number of 1's in data is:
    - Even → Parity bit = 0
    - Odd → Parity bit = 1

Expression (for 3-bit data A, B, C)

$$P_{even} = A \oplus B \oplus C$$

Example

Data = 101
Number of 1's = 2 (even)
Parity bit = 0
Transmitted data = 1010



3.6.1.2. Odd Parity Generator

Rule:

Total number of 1's (including parity bit) must be **odd**.

Working:

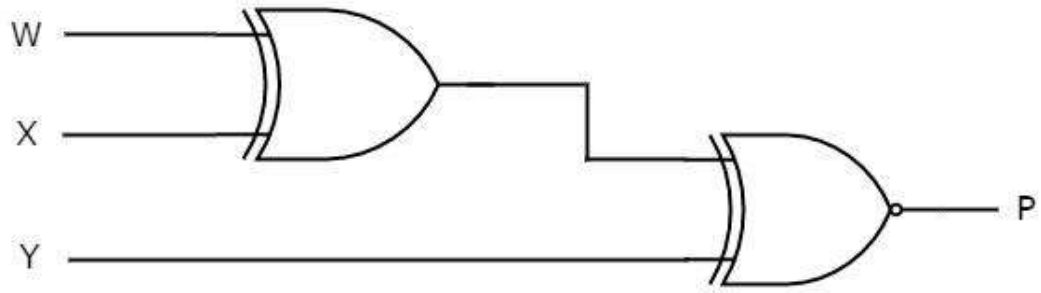
1. First generate even parity using XOR.
2. Then complement it.

Expression

$$P_{odd} = (A \oplus B \oplus C)'$$

Example

Data = 101
Number of 1's = 2 (even)
To make total odd → Add P = 1
Transmitted data = 1011



Example

Data = 101

Number of 1's = 2 (even)

- For **Even parity** → P = 0
- For **Odd parity** → P = 1

Truth Table (Even Parity)

A	B	C	Parity Bit (P)
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

3.6.2. Parity Checker

A **Parity Checker** checks whether the received data has error or not.

It uses XOR gates.

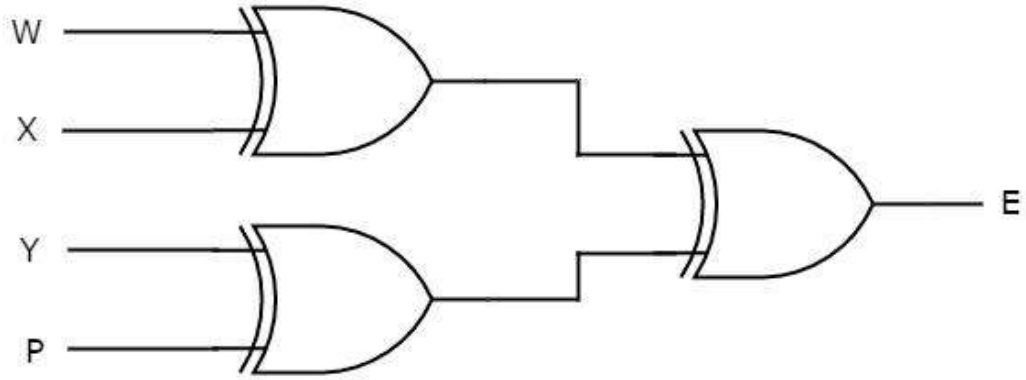
Working (Even Parity Checking)

Received bits: A, B, C, P

$$Output = A \oplus B \oplus C \oplus P$$

- Output = 0 → No error

- Output = 1 → Error detected



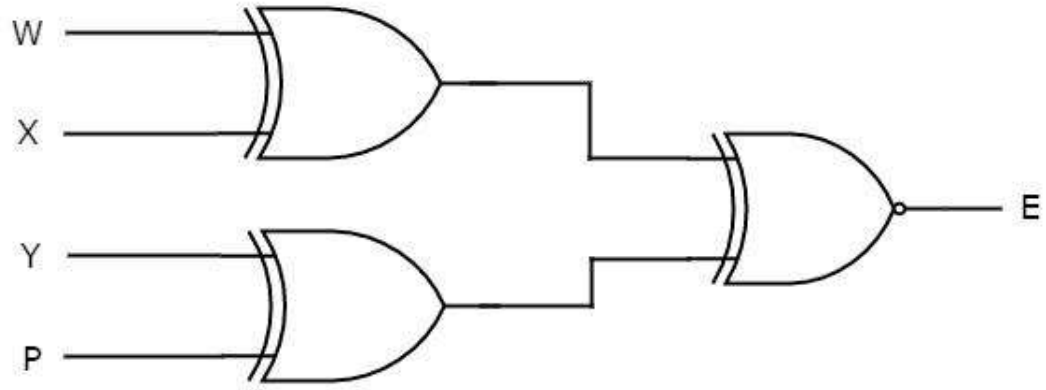
Working (odd Parity Checking)

For 3-bit data (A, B, C) and parity bit P:

$$Check = A \oplus B \oplus C \oplus P$$

For **Odd Parity System**:

- Output = 1 → Correct data
- Output = 0 → Error present



Example 1 (No Error Case)

Data = 101

Number of 1's = 2 (even)

To make total odd → Add P = 1

Transmitted Data = 1011

Receiver Check:

$$1 \oplus 0 \oplus 1 \oplus 1 = 1$$

Output = 1 → **No error**

Example 2 (Error Case)

Suppose during transmission:

Received = 1001

Now check:

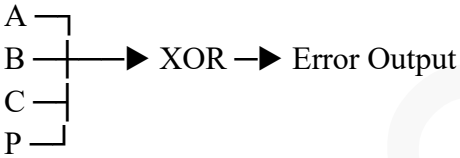
$$1 \oplus 0 \oplus 0 \oplus 1 = 0$$

Output = 0 → **Error detected**

Block Diagram Concept

Parity Generator: Data bits → XOR gates → Parity bit output

Parity Checker: Data bits + Parity bit → XOR gates → Error output



Error Detection Capability

- Detects **single-bit errors**
- Cannot detect:
    - Two-bit errors
    - Multiple even-number bit errors

Applications

- Data transmission systems
- Computer memory systems
- Communication systems
- Microprocessor data transfer