@Sree Vishnu Varthini

21 Days of Embedded Systems Programming

@Sree Vishnu Varthini

Day -1

Embedded Systems Programming

OHM'S LAW

$$V = I R$$

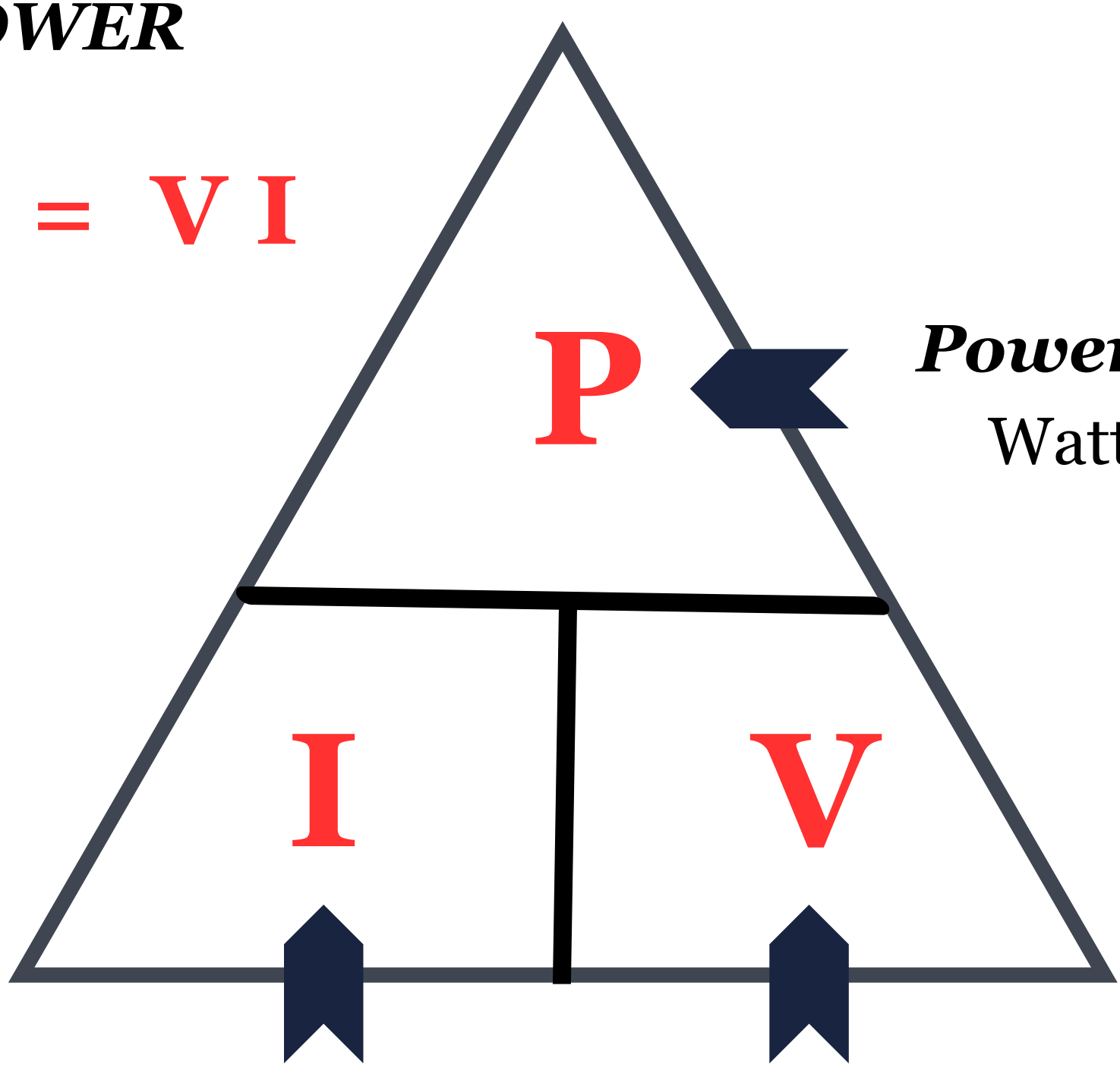
V --> *Voltage* in Volts (**Generation**)

I --> *Current* in Amperes
(**Transmission**)

R --> *Resistance* in Ohms (**Utilization**)

POWER

$$P = V I$$



Power in Watts

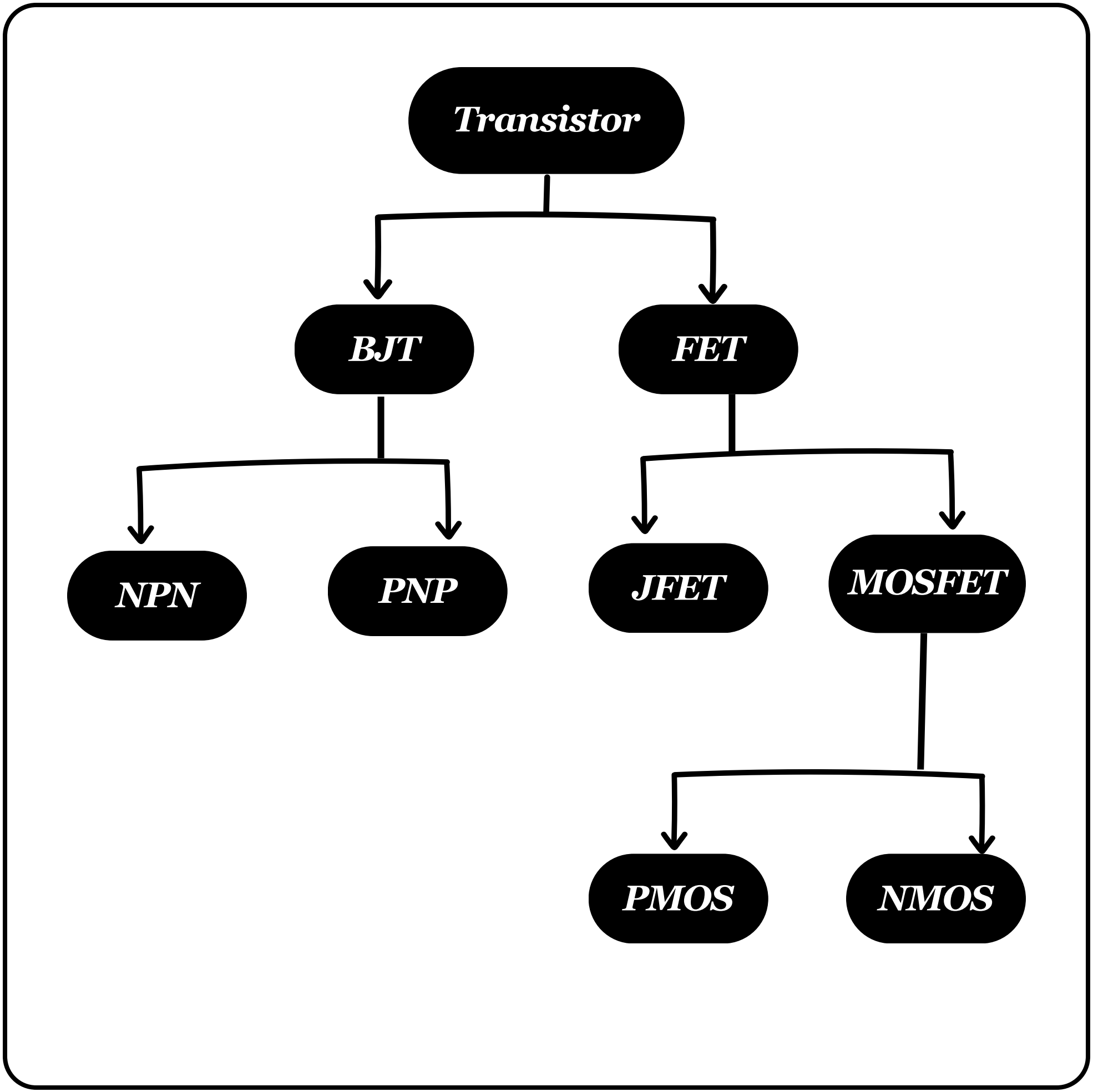
Current in Amperes

Voltage in Volts

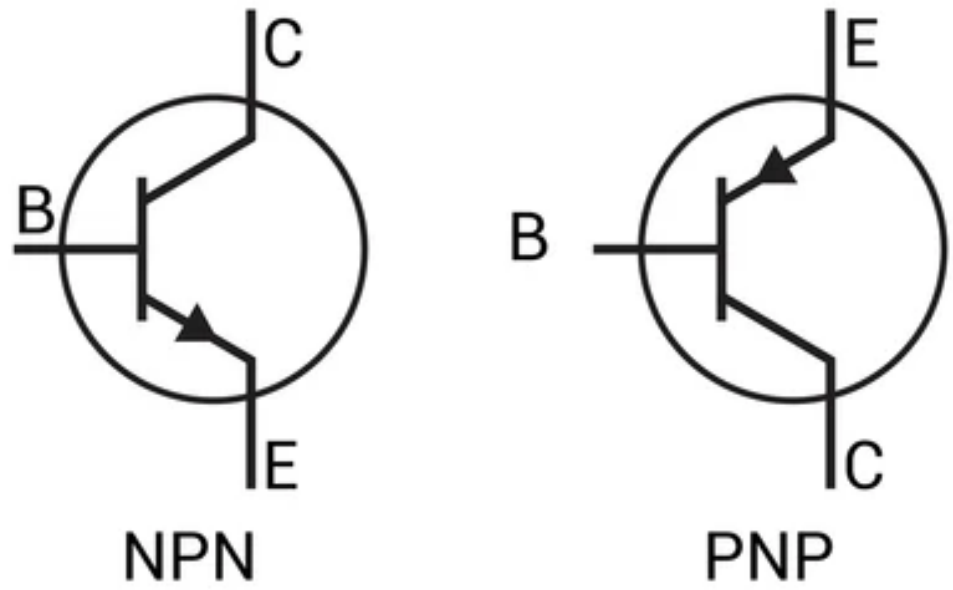
@Sree Vishnu Varthini

Day - 2

Embedded Systems Programming


Transistor
BJT
FET
NPN
PNP
JFET
MOSFET
PMOS
NMOS

BJT - Bipolar Junction Transistor



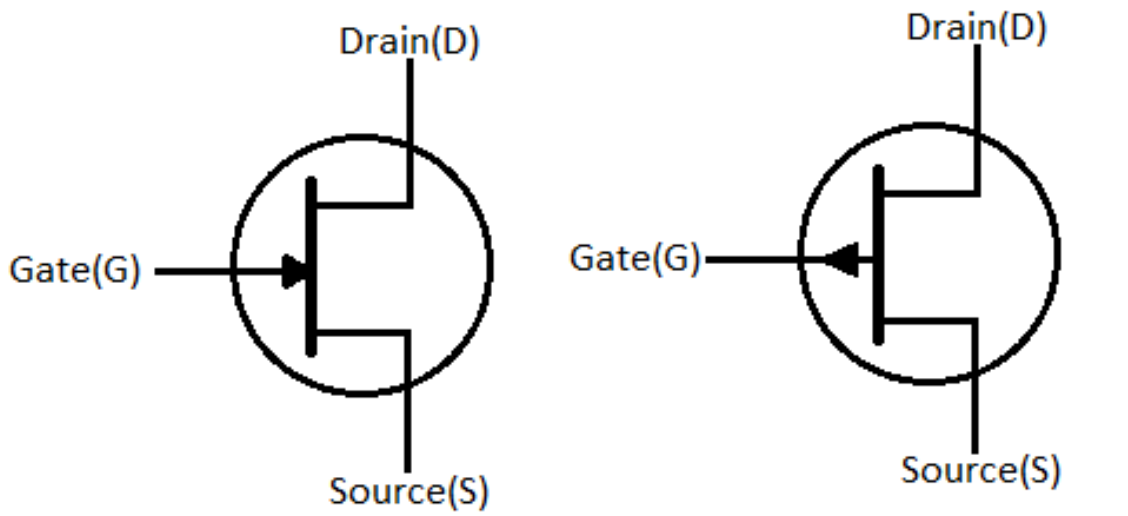
Base Current, $i_b = (V_b - V_{BE})/R_b$

Collector Current, $i_c = \beta\, i_b$

Collector Voltage, $V_c = i_c \times R_c$

Output Voltage, $V_o = V - V_c$

JFET - Junction Field Effect Transistor



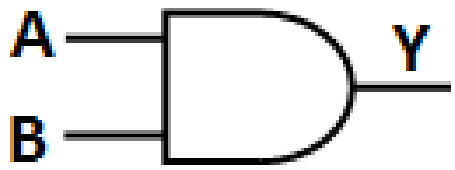
MOSFET - Metal Oxide Semiconductor Field Effect Transistor

@Sree Vishnu Varthini

Day - 3

Embedded Systems Programming

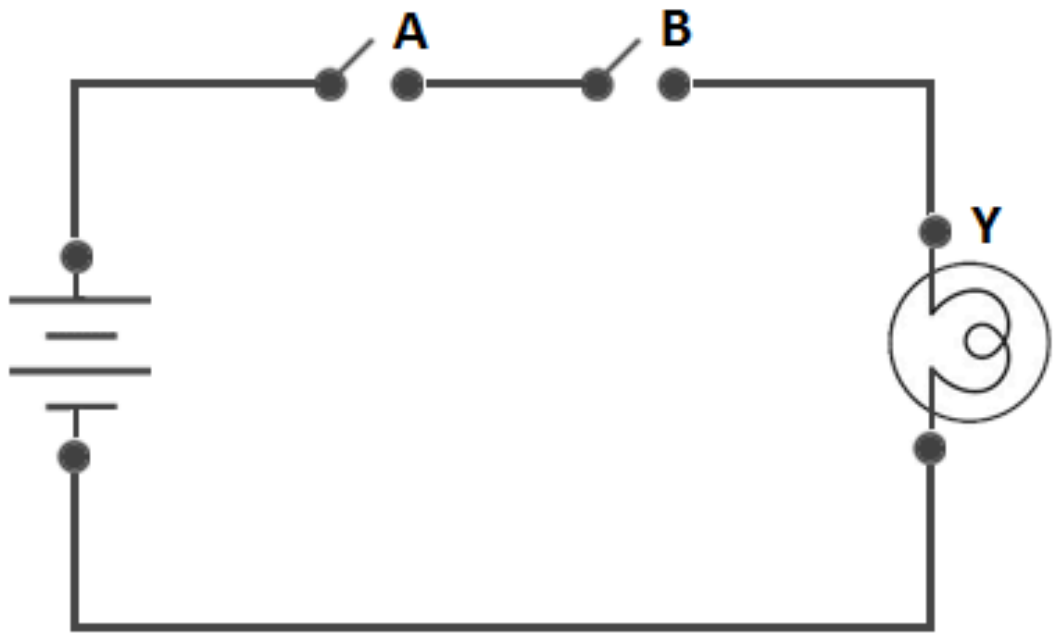
AND GATE



Symbol

A	B	Y = A * B
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table



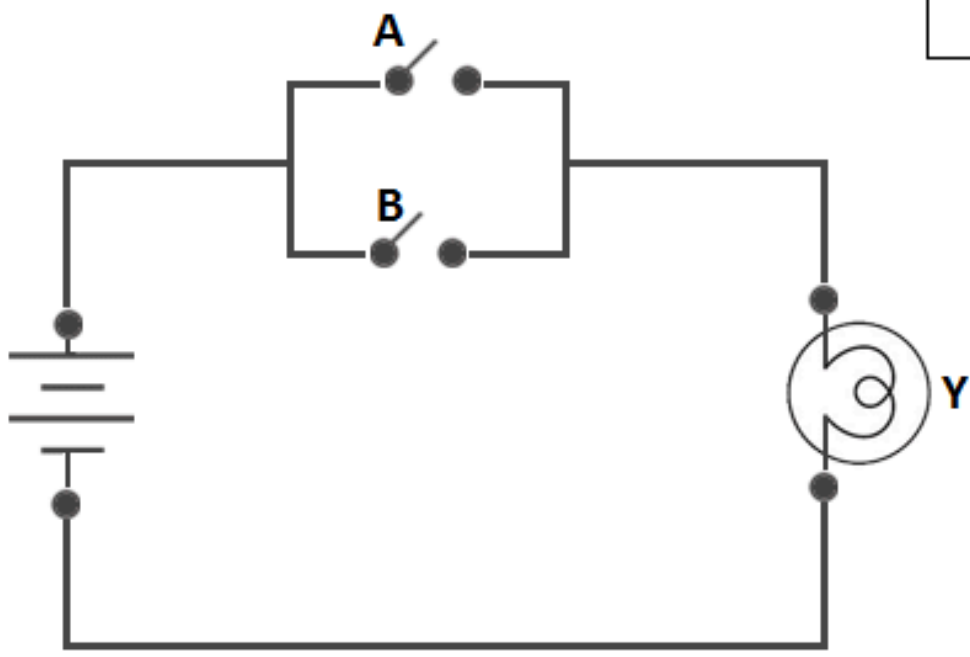
Electrical Circuit

OR GATE

Symbol

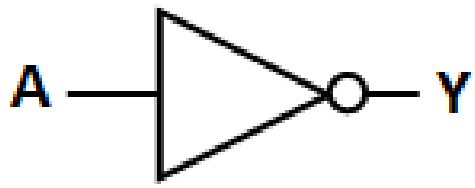
A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table



Electrical Circuit

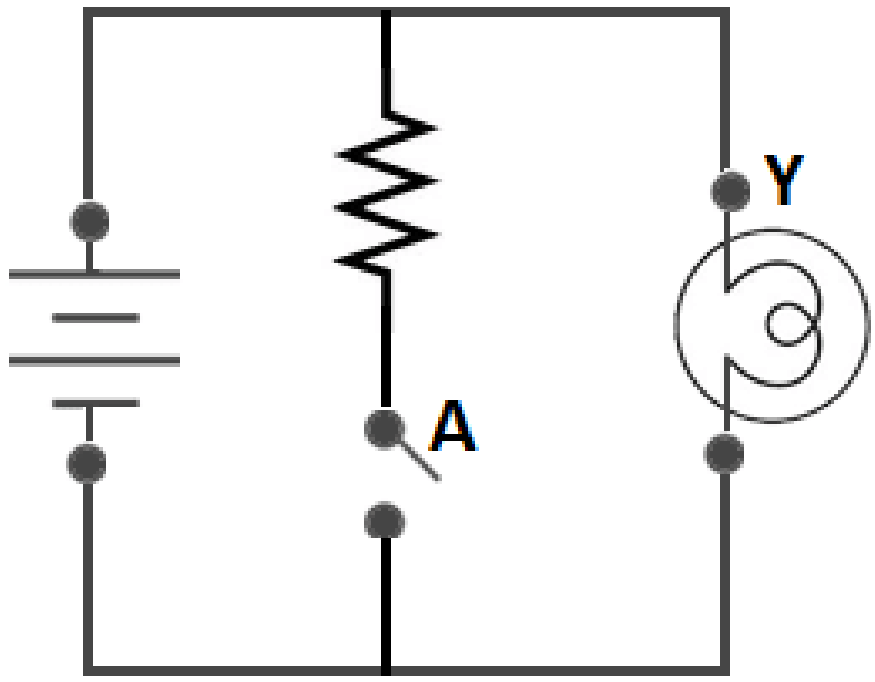
NOT GATE



Symbol

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

Truth Table



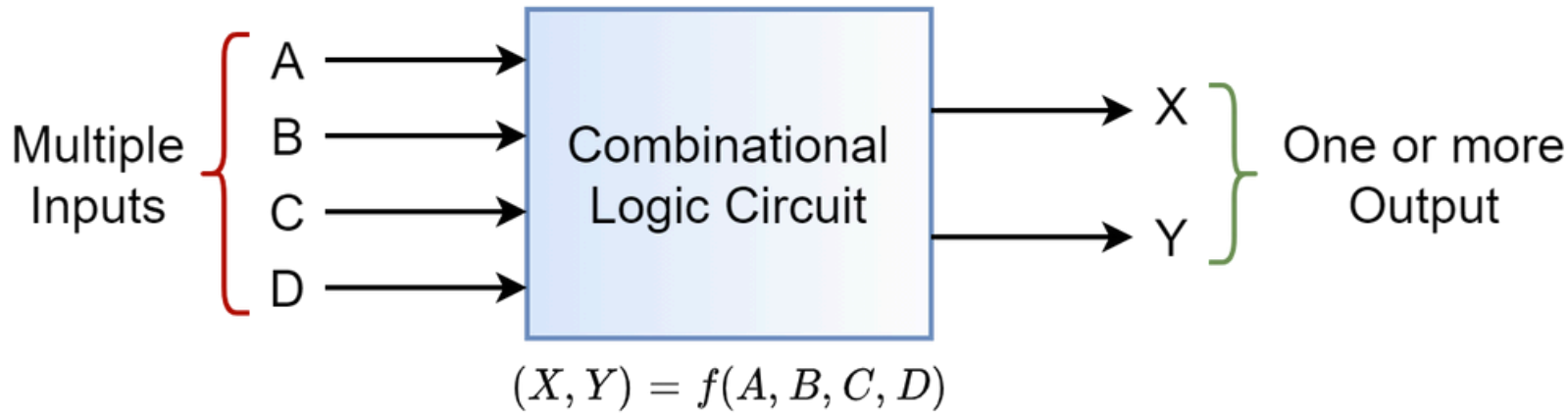
Electrical Circuit

@Sree Vishnu Varthini

Day - 4

Embedded Systems Programming

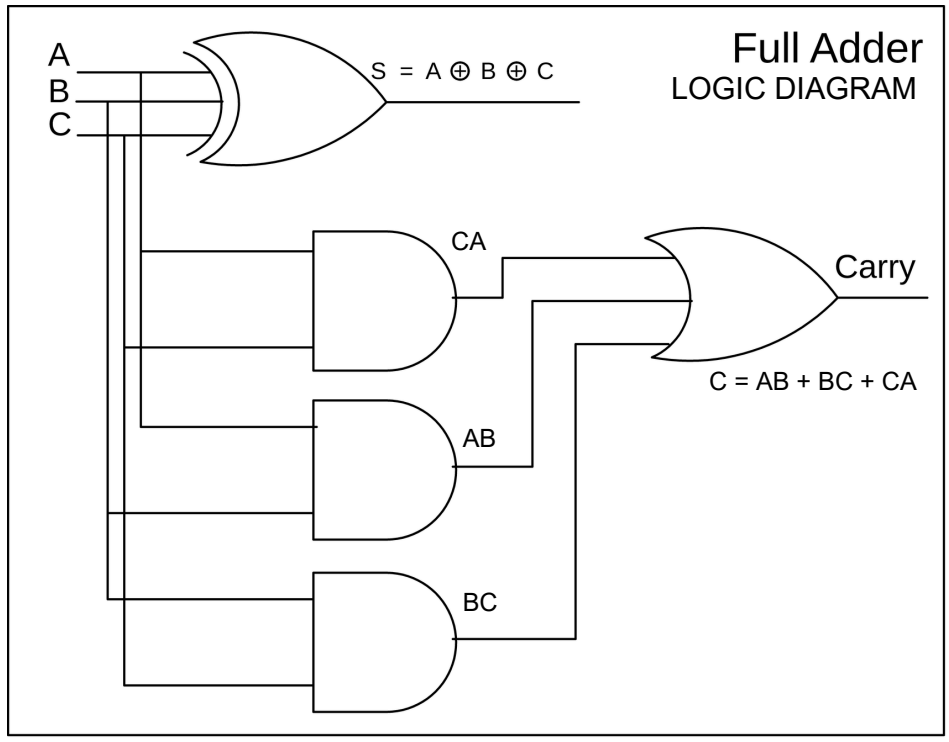
COMBINATIONAL LOGIC



Applications:

- Adder
- Subtractor
- Encoder
- Decoder
- Multiplexer
- Demultiplexer

FULL ADDER



- A full adder adds three 1-bit binary inputs: **A, B and C** (Carry-in).
- It has two outputs: **Sum** and **Carry** (Carry-out).

Sum:

$(A \oplus B) \oplus C$

Carry:

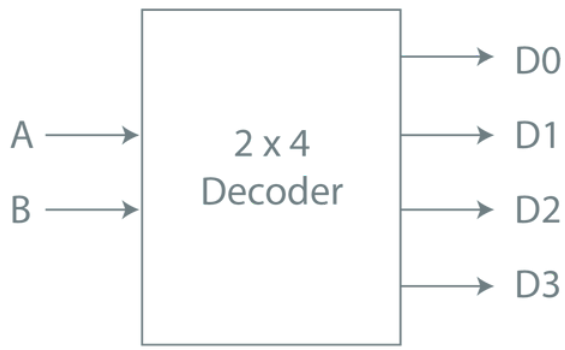
AB + BC + CA

Truth Table

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
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

DECODER

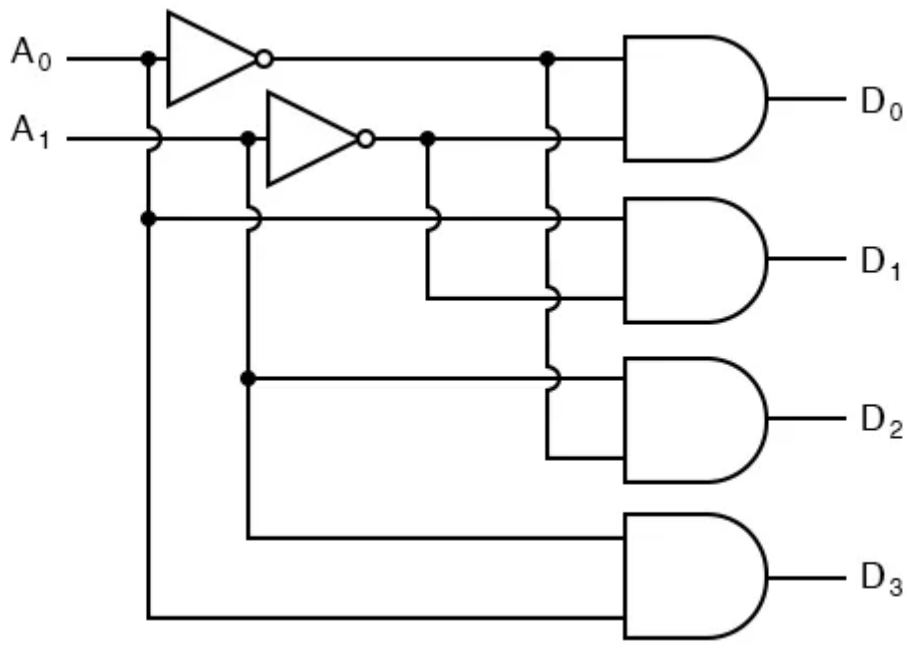
Block Diagram



Truth Table

Enable	INPUTS		OUTPUTS			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Logic Diagram



A decoder is a combinational circuit that converts binary data from N input lines into 2^N output lines.

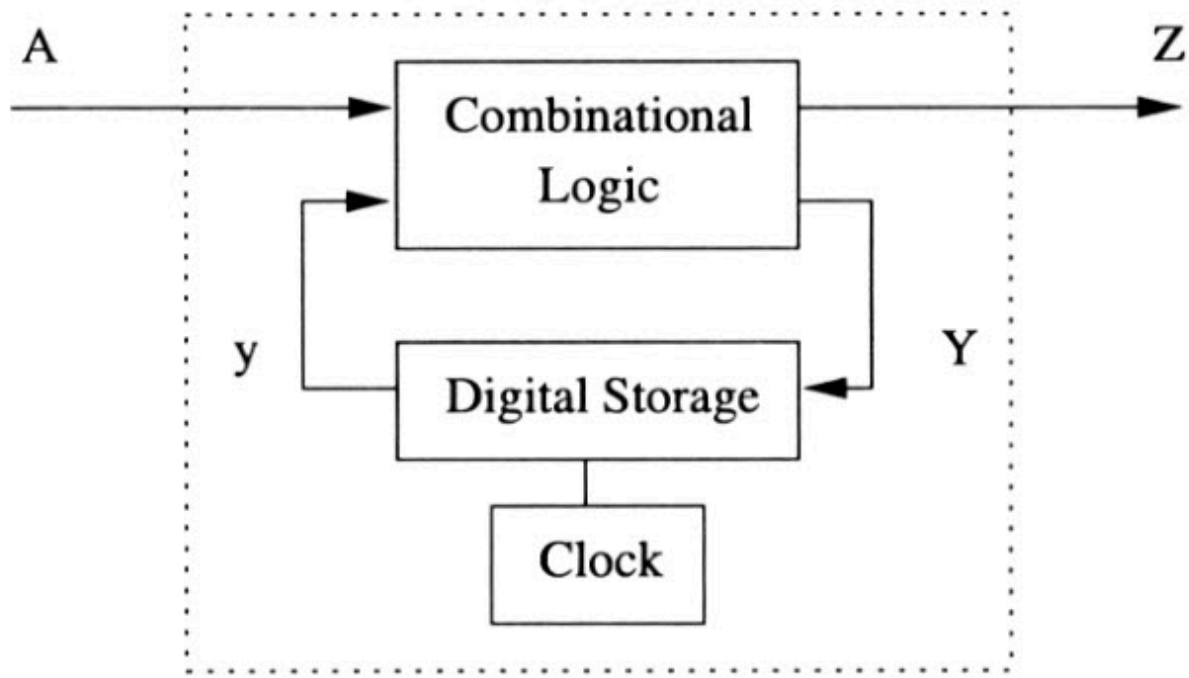
In the 2 to 4 line decoder, there is a total of two inputs, i.e., A0, and A1 and four outputs, i.e., Y0, Y1, Y2, and Y3.

@Sree Vishnu Varthini

Day - 5

Embedded Systems Programming

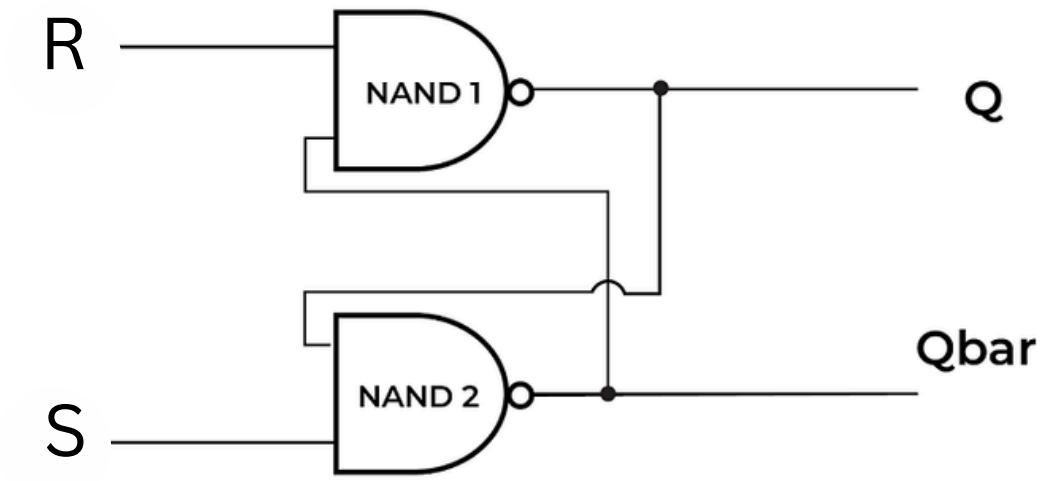
SEQUENTIAL LOGIC



Applications:

- Shift Registers
- Flipflops
- Counters
- Registers

RS LATCH



The SR Latch is one of the simplest types of flip-flops used for storing a single bit of data (either 0 or 1). The name "**SR**" stands for **Set** and **Reset**.

Truth Table:

R	S	Qn+1
0	0	Forbidden
0	1	1
1	0	0
1	1	Hold

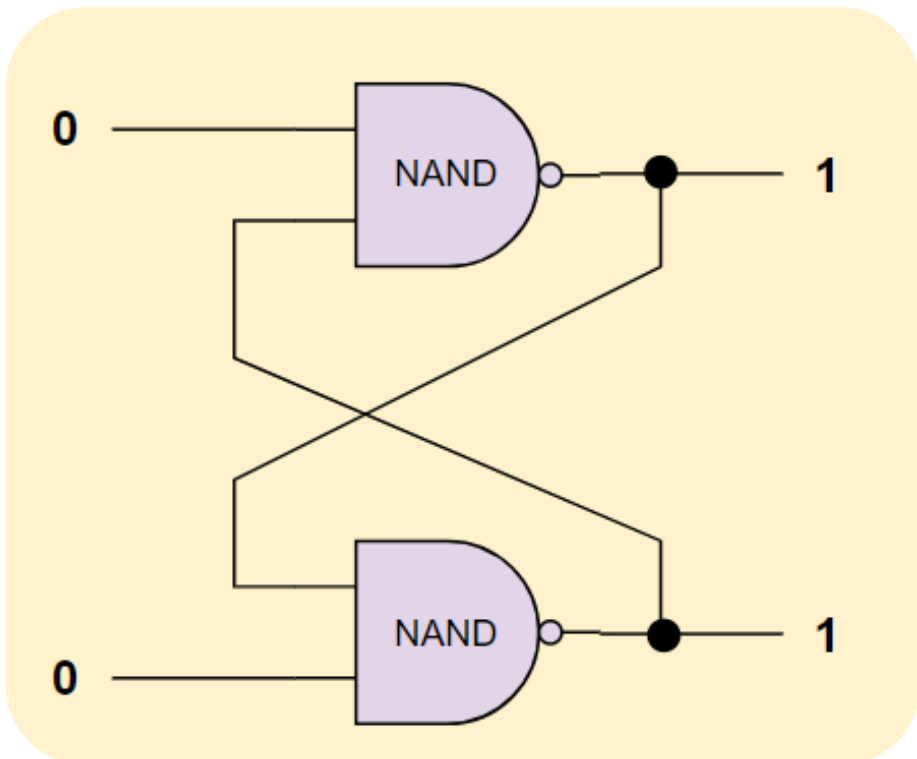
Working:

Case 1: Invalid State (R = 0, S = 0)

R = 0 $\rightarrow$ Output of the first NAND gate (Q) becomes 1.

S = 0 $\rightarrow$ Output of the second NAND gate (Q') becomes 1.

Both outputs, Q and Q', become 1, which is invalid since Q' should be the inverse of Q. This input should be avoided in practical designs.

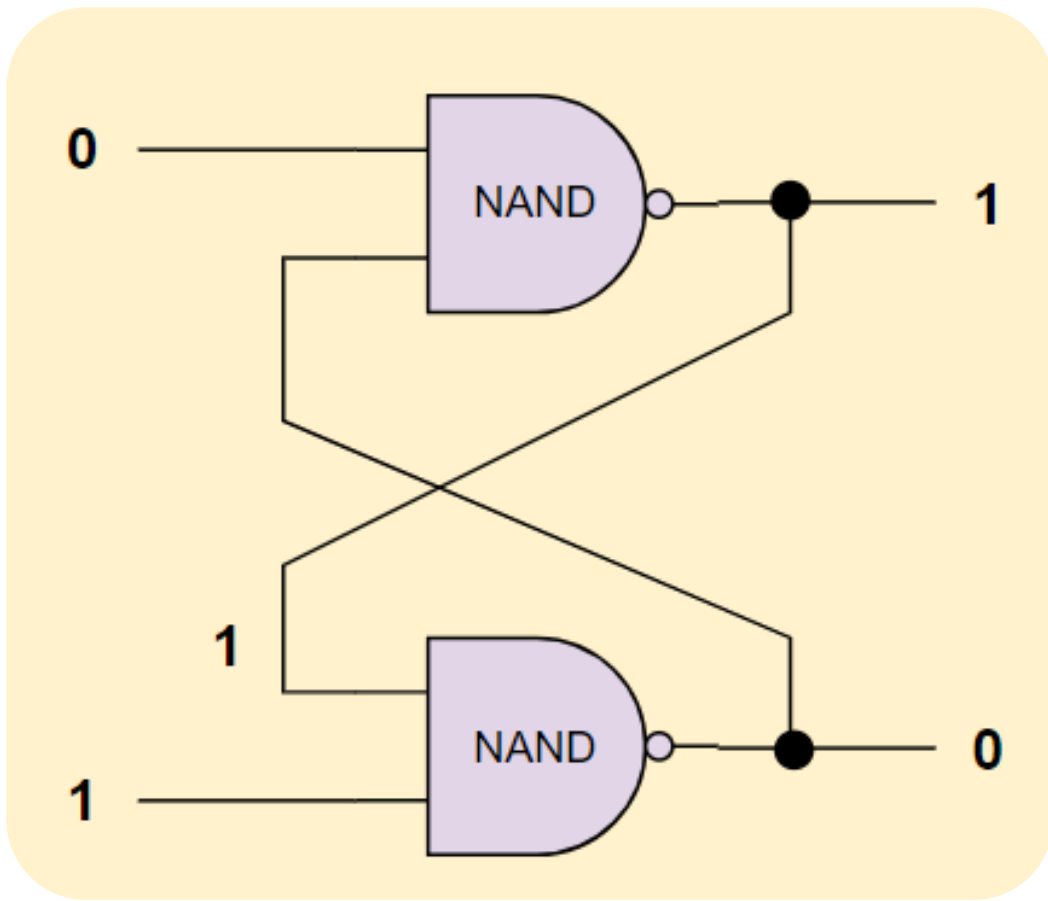


Working:

Case 2: Set State (R = 0, S = 1)

R = 0 $\rightarrow$ Output of the first NAND gate (Q) becomes 1. Due to this, Output of the second NAND gate (Q') becomes 0.

Q is set to 1, and so Q’ is set to 0.



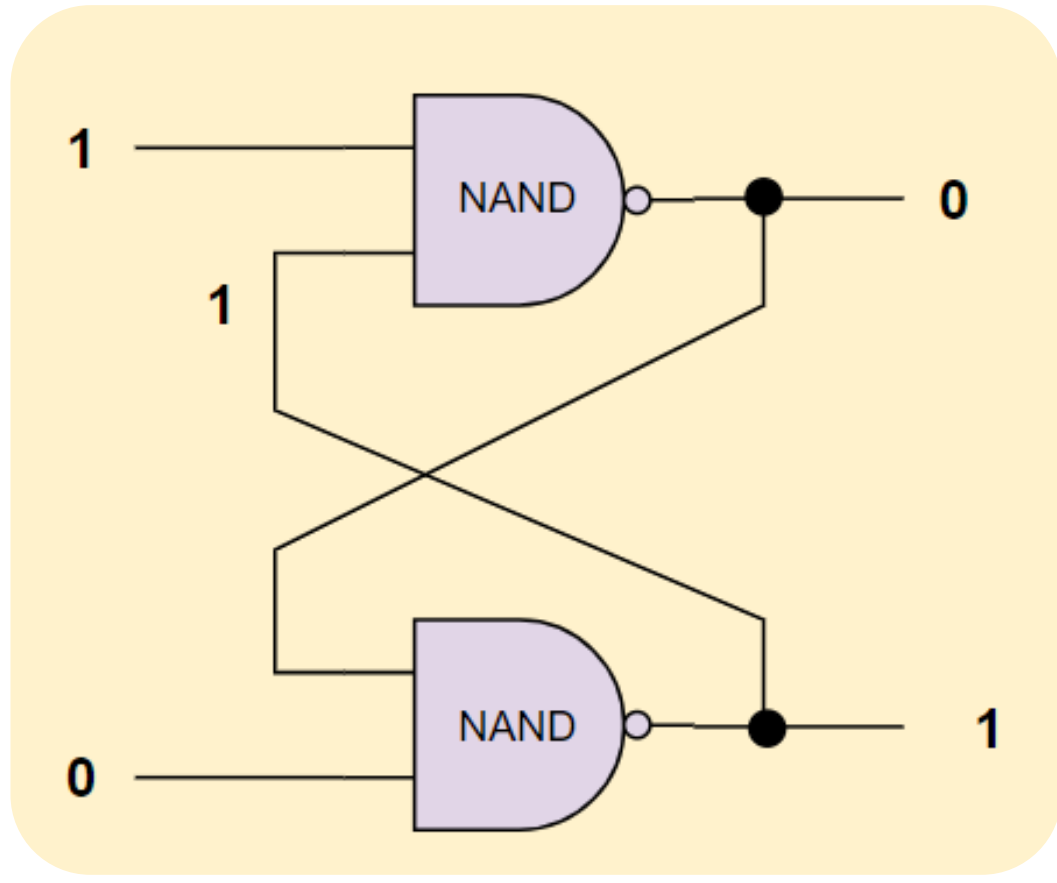
Working:

Case 3: Reset State (R = 1, S = 0)

S = 0 $\rightarrow$ Output of the second NAND gate (Q') becomes 1.

Due to this, Output of the first NAND gate (Q) becomes 0.

Q’ is set to 1, and so Q is set to 0.

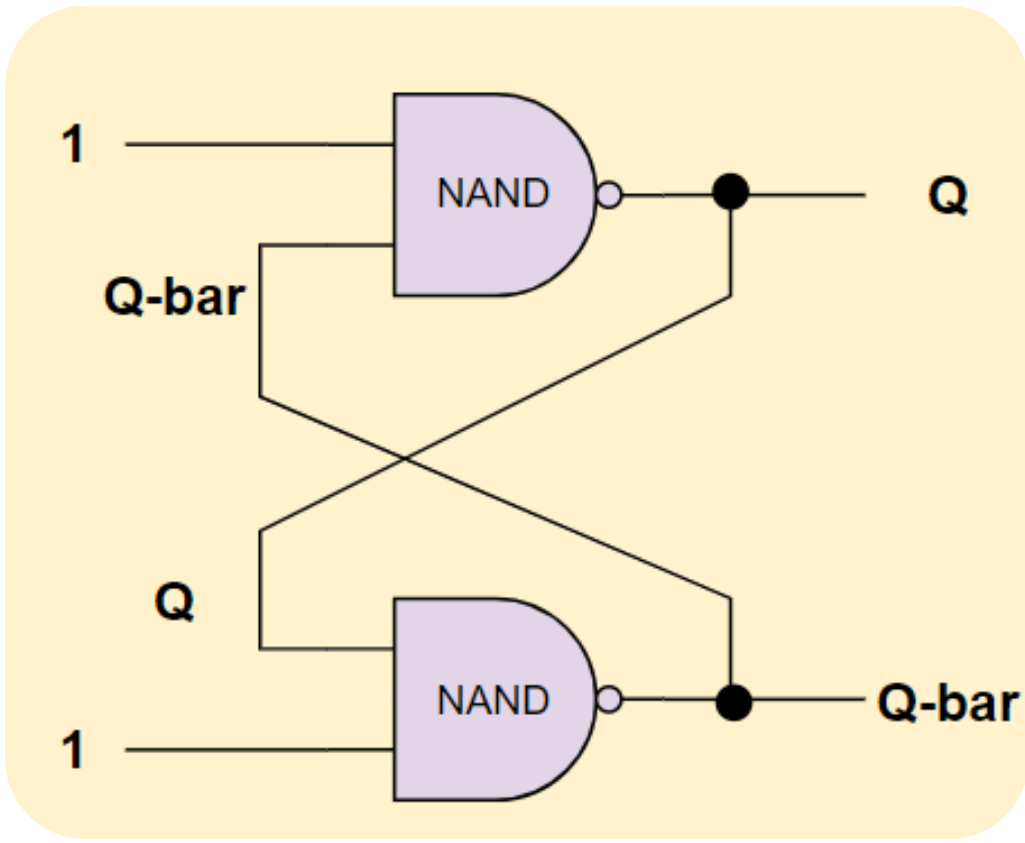


Working:

Case 4: Hold State (R = 1, S = 1)

Inputs of both NAND gates (S and R) are 1, which results in depending on the previous Q values for output.

The flip-flop remains in its current state, with Q and Q' retaining their previous values.

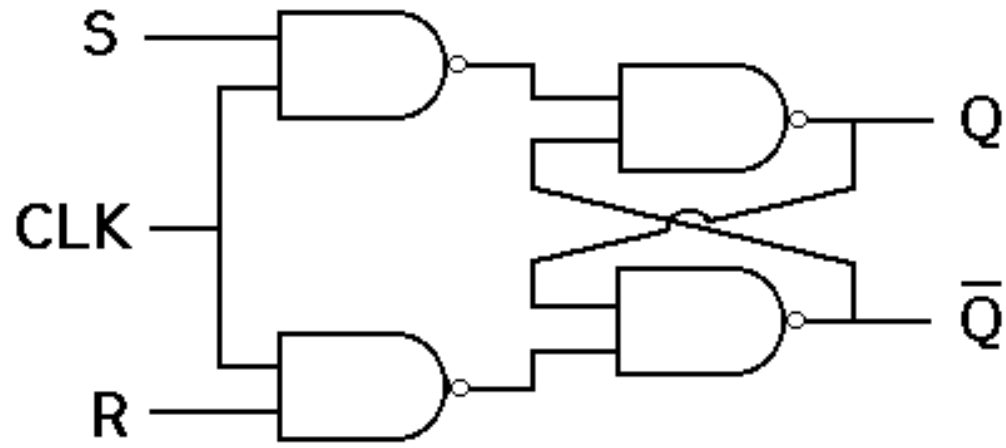


@Sree Vishnu Varthini

Day - 6

Embedded Systems Programming

SR FLIPFLOP



An SR flip-flop, or Set-Reset flip-flop, is a basic memory element in digital electronics. It has two inputs: S (Set) and R (Reset), and two outputs: Q and Q' (the inverse of Q).

Truth Table

S	R	Q_{n+1}
0	0	Q_n(No Change)
0	1	0
1	0	1
1	1	x

Components of SR Flipflop:

Inputs:

- **S (Set):** Sets the output (Q) to 1.
- **R (Reset):** Resets the output (Q) to 0.

Outputs:

- **Q:** Main output of the flip-flop.
- **Q':** Inverse of Q (if Q = 1, then Q' = 0 and vice versa).

Analogy:

Imagine the SR flip-flop as a light switch:

- The Set input (S) turns the light **on** (Q = 1).
- The Reset input (R) turns the light **off** (Q = 0).
- When neither Set nor Reset is pressed (S = 0, R = 0), the light stays in whatever **state it was previously in.**
- If both buttons are pressed at the same time (S = 1, R = 1), the system is confused and enters an **invalid state.**

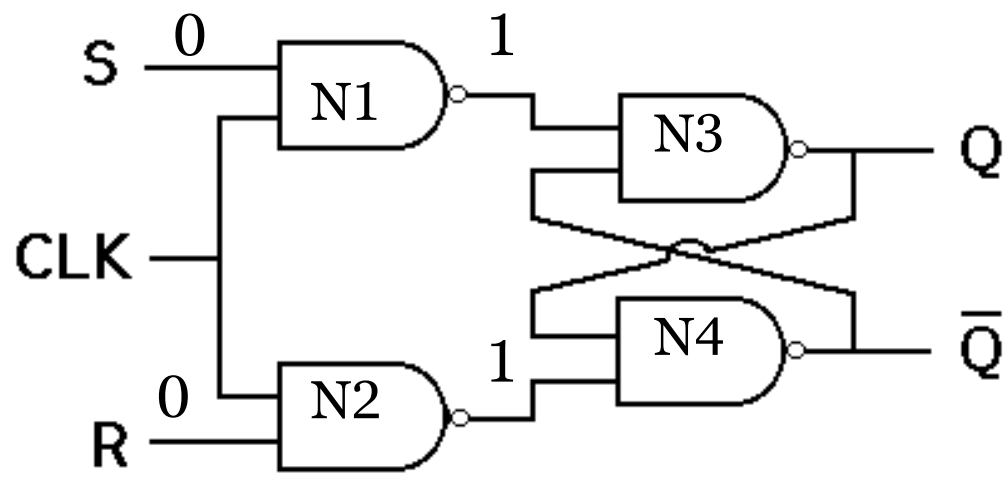
Working:

Case 1: Hold State (S = 0, R = 0)

Outputs of both NAND gates (N1 and N2) become 1.

Inputs of both NAND gates (N3 and N4) are 1, which results in depending on the previous Q values for output.

The flip-flop remains in its current state, with Q and Q' retaining their previous values.



Working:

Case 2: Reset State (S = 0, R = 1)

S = 0 → Output of the first NAND gate (N1) becomes 1.

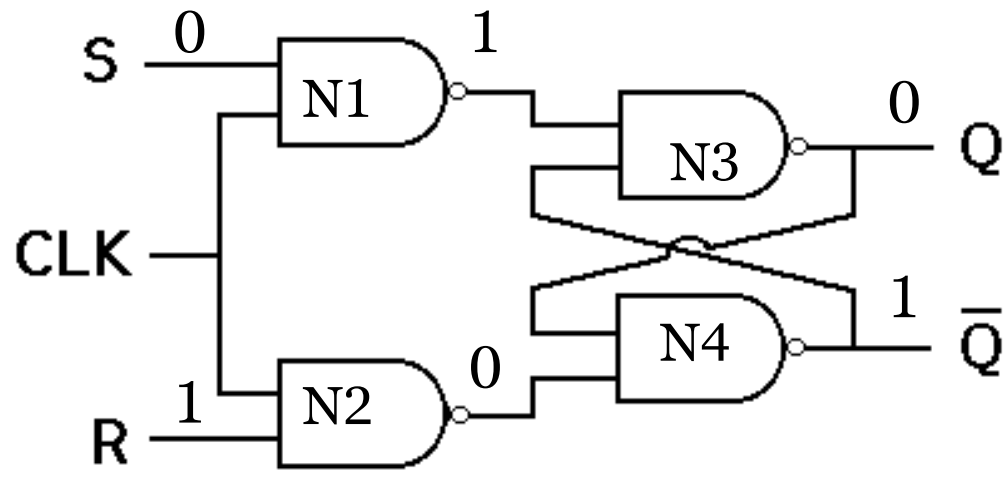
R = 1 → Output of the second NAND gate (N2) becomes 0.

Input to fourth NAND gate (N4) is 0 and so Q' becomes 1.

Due to this, input to third NAND (N3) gate becomes 1 and 1.

So Q becomes 0.

Q' is set to 1, and so Q is set to 0.



Working:

Case 3: Set State (S = 1, R = 0)

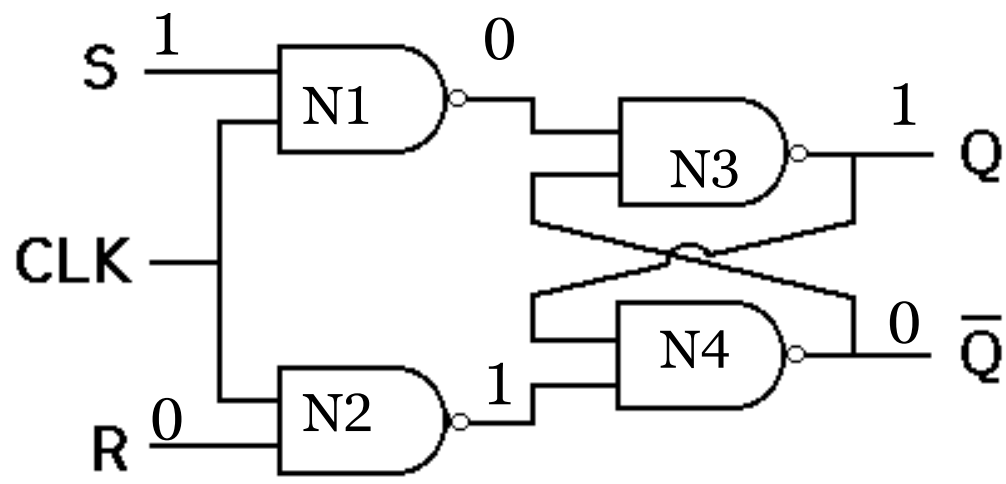
S = 1→ Output of the first NAND gate (N1) becomes 0.

R = 0→ Output of the second NAND gate (N2) becomes 1.

Input to third NAND gate (N3) is 0 and so Q becomes 1.

Due to this, input to fourth NAND (N4) gate becomes 1 and 1. So Q' becomes 0.

Q is set to 1, and so Q' is set to 0.



Working:

Case 4: Invalid State (S = 1, R = 1)

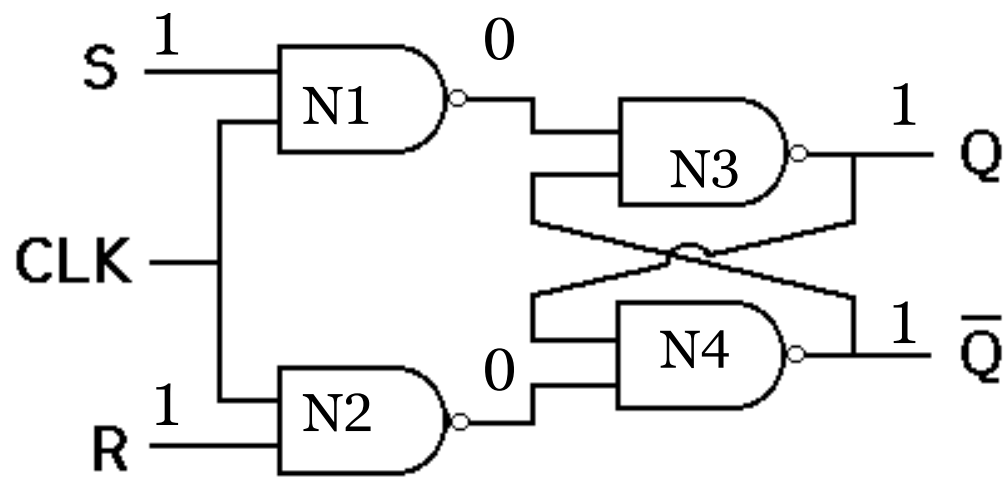
S = 1 → Output of the first NAND gate (N1) becomes 0.

R = 1 → Output of the second NAND gate (N2) becomes 0.

Inputs to both NAND gates (N3 and N4) is 0.

So both outputs, Q and Q', become 1, which is invalid since Q' should be the inverse of Q.

This input should be avoided in practical designs.

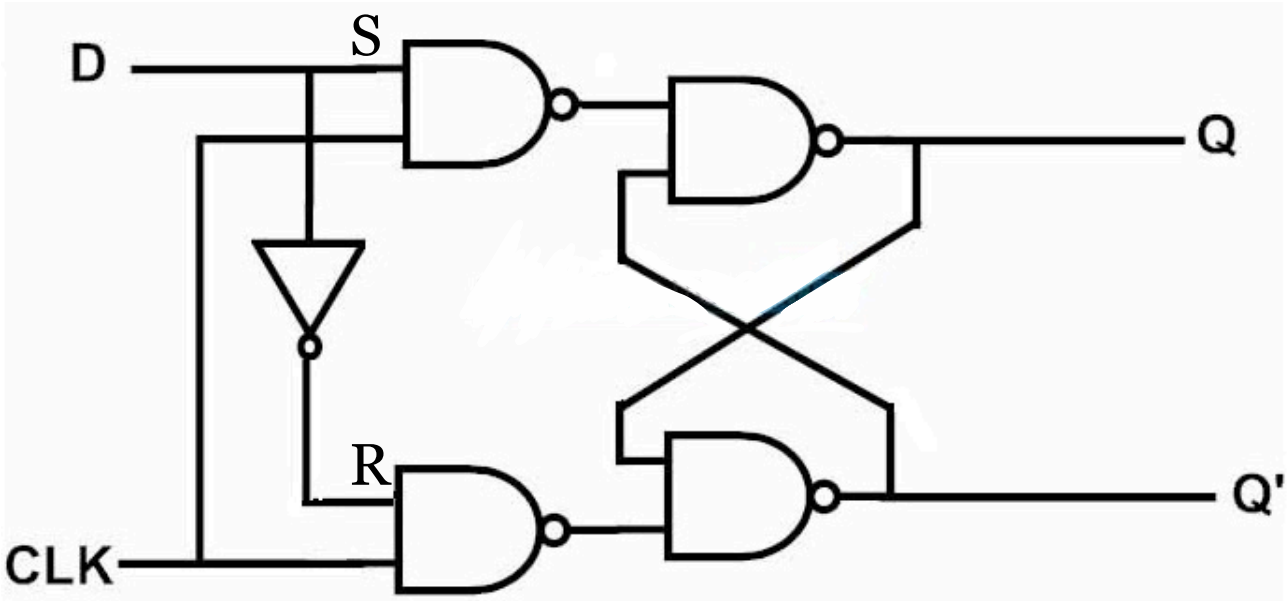


@Sree Vishnu Varthini

Day - 7

Embedded Systems Programming

D FLIPFLOP



A D Flip-Flop (also called a Data or Delay Flip-Flop) is a type of digital storage device that stores one bit of data and is widely used in digital circuits to maintain memory.

Truth Table

D Flip Flop		
Input	Output	
D	Q	Q'
0	0	1
1	1	0

REASON FOR CREATION OF D FLIPFLOP

The D flip-flop was designed to simplify the operation of the SR flip-flop and avoid the problem of invalid states. Instead of two inputs (S and R), the D flip-flop uses only one input, called D (Data).

To prevent the invalid state (S = R = 1), the D flip-flop was derived by connecting S and R logically. This is done by:

- Setting S = D
- Setting R = NOT(D) (or D')

This ensures that S and R are never both 1 at the same time.

By connecting D to S and its inverse to R, the D flip-flop operates with just one input (D), making it simpler and avoiding any unpredictable behavior.

Working:

Case 1: D = 0

D = 0 → S remains 0 and R becomes 1

S = 0 → Output of the first NAND gate (N1) becomes 1.

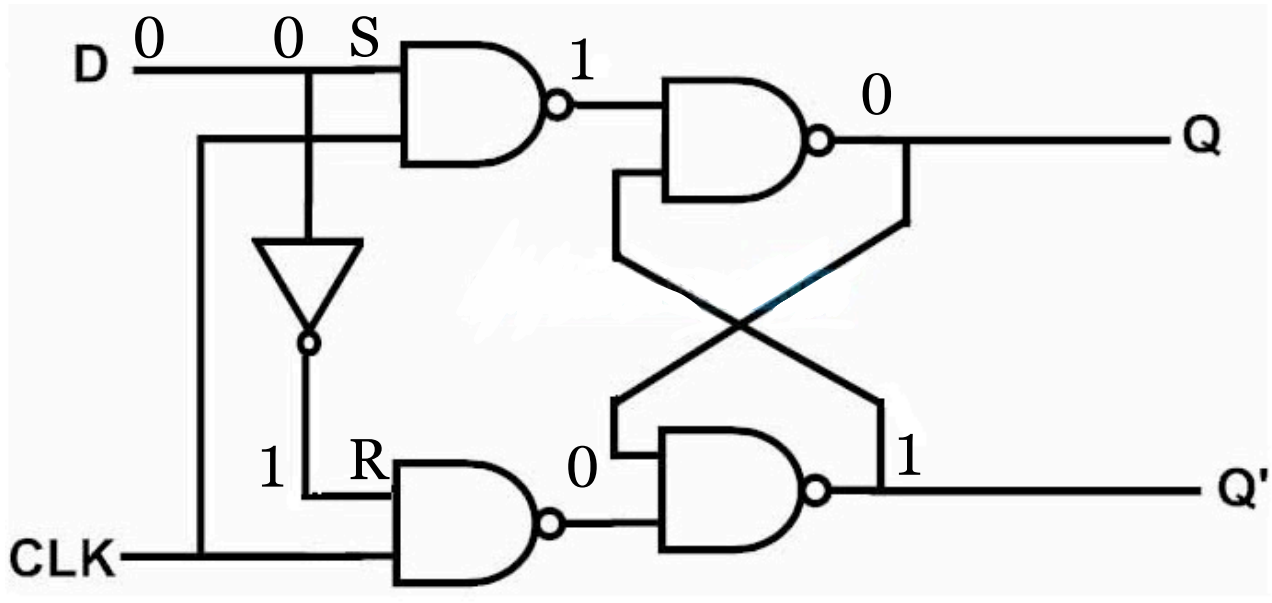
R = 1 → Output of the second NAND gate (N2) becomes 0.

Input to fourth NAND gate (N4) is 0 and so Q' becomes 1.

Due to this, input to third NAND (N3) gate becomes 1 and 1.

So Q becomes 0.

Q' is set to 1, and so Q is set to 0.



Working:

Case 2: D = 1

D = 1 → S remains 1 and R becomes 0

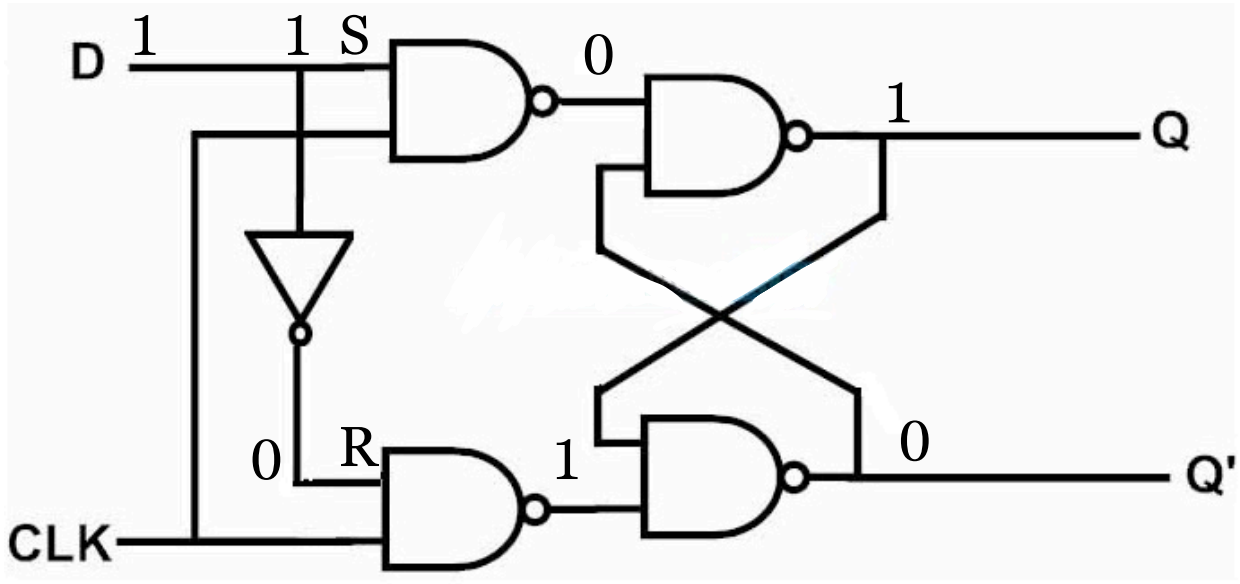
S = 1 → Output of the first NAND gate (N1) becomes 0.

R = 0 → Output of the second NAND gate (N2) becomes 1.

Input to third NAND gate (N3) is 0 and so Q becomes 1.

Due to this, input to fourth NAND (N4) gate becomes 1 and 1. So Q' becomes 0.

Q is set to 1, and so Q' is set to 0.

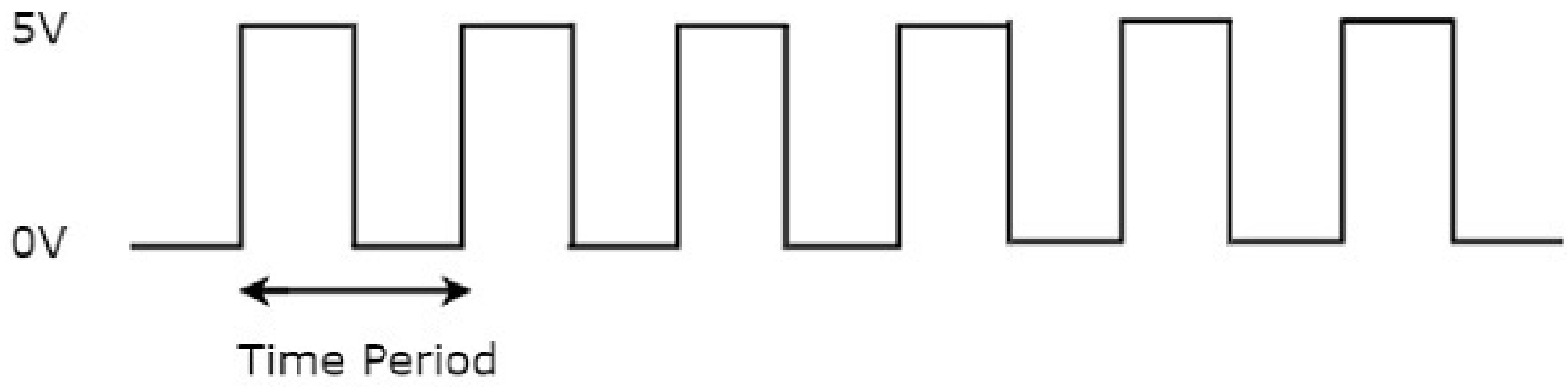


@Sree Vishnu Varthini

Day - 8

Embedded Systems Programming

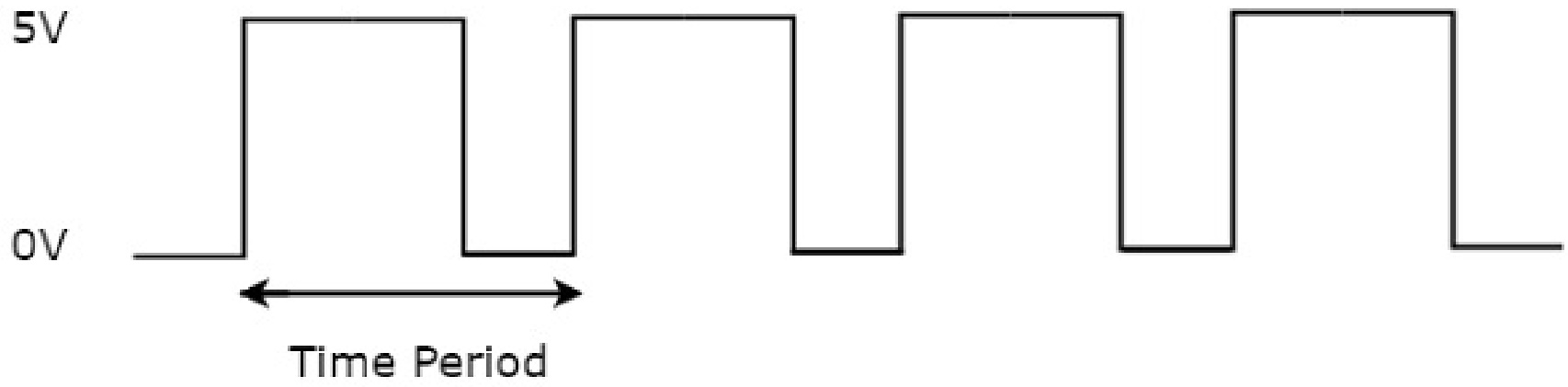
CLOCK SIGNAL



A **clock signal** is a timing signal used in digital circuits to synchronize the operations of different components. It provides a steady pulse at regular intervals, acting like a **rhythm keeper** for electronic devices. The ON time and OFF time of clock signal **need not be the same.**

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

CLOCK SIGNAL



In the above figure, train of pulses is considered as clock signal. This signal stays at **logic High (5V)** for some time and stays at **logic Low (0V)** for some other time. This pattern repeats with some time period. In this case, the time period will be **equal to sum of ON time and OFF time.**

The **reciprocal of the time period** of clock signal is known as the **frequency of the clock signal.** All sequential circuits are operated with clock signal.

KEY CHARACTERISTICS

- **Frequency:** The rate at which the clock ticks (measured in Hertz, Hz). For example, a 1 MHz clock signal ticks one million times per second.

- **Duty Cycle:** The percentage of one cycle in which the signal is high (active). A 50% duty cycle means the signal is high for half the time and low for the other half.

- **Waveform:** Clock signals typically take the form of a square wave, characterized by a rapid rise and fall between high and low states.

IMPORTANCE OF CLOCK SIGNALS

Clock signals are crucial for:

💡 **Synchronization:** Ensuring all parts of a digital system operate in unison.

💡 **Data Transfer:** Dictating when data is valid and can be read or written.

💡 **Timing Control:** Managing the sequence of operations in circuits, like adders, counters, and registers.

@Sree Vishnu Varthini

Day - 9

Embedded Systems Programming

TRIGGERING

Triggering refers to the action that occurs in response to a clock signal. It determines when a circuit should read or change its output based on the clock pulse.

Triggers determine how and when a circuit responds to clock signals, especially in sequential circuits like flip-flops and registers.

TYPES OF TRIGGERING

- Level triggering
- Edge triggering

LEVEL TRIGGERING

Level triggering responds to the level of the clock signal rather than its edges.

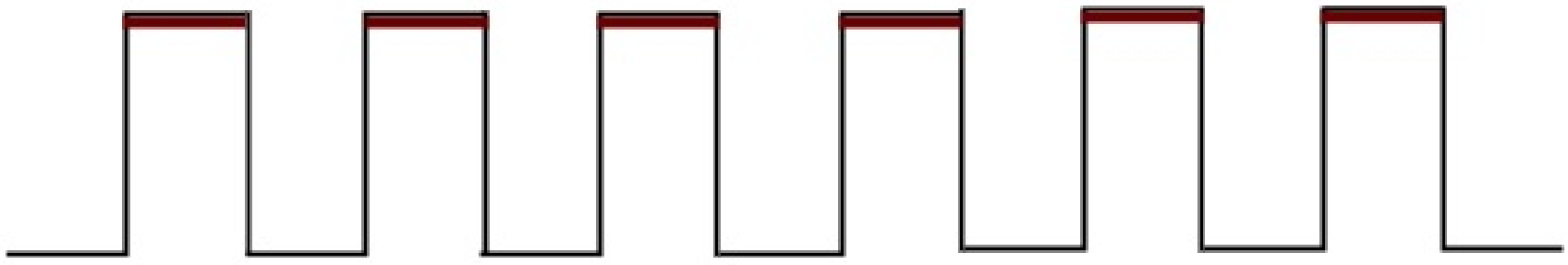
LEVEL TRIGGERING TYPES

- Positive Level triggering
- Negative Level triggering

POSITIVE LEVEL TRIGGERING

The circuit responds as long as the clock signal is high (1).

Visual Representation

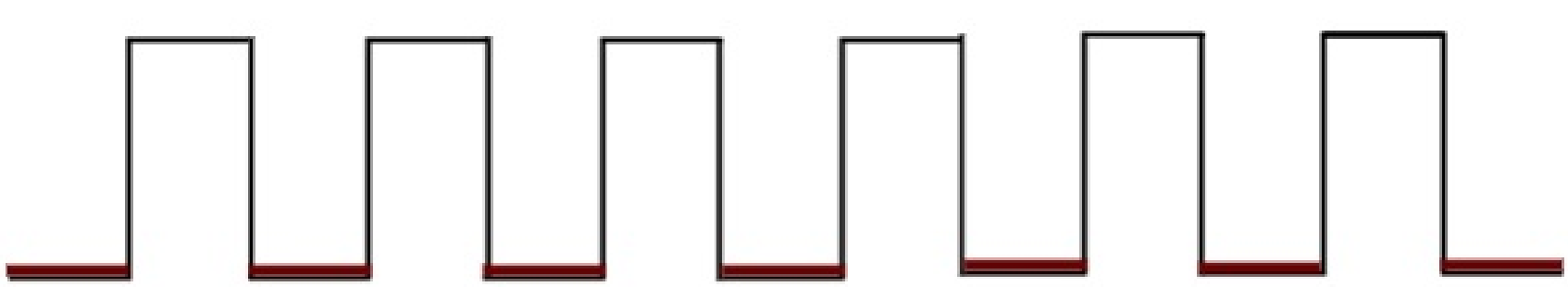


Positive level triggering maintains the output while the clock signal is high.

NEGATIVE LEVEL TRIGGERING

The circuit responds as long as the clock signal is low (0).

Visual Representation



Nagative level triggering maintains the output while the clock signal is low.

EDGE TRIGGERING

Edge triggering responds to changes in the clock signal, specifically to the rising or falling edges of the signal.

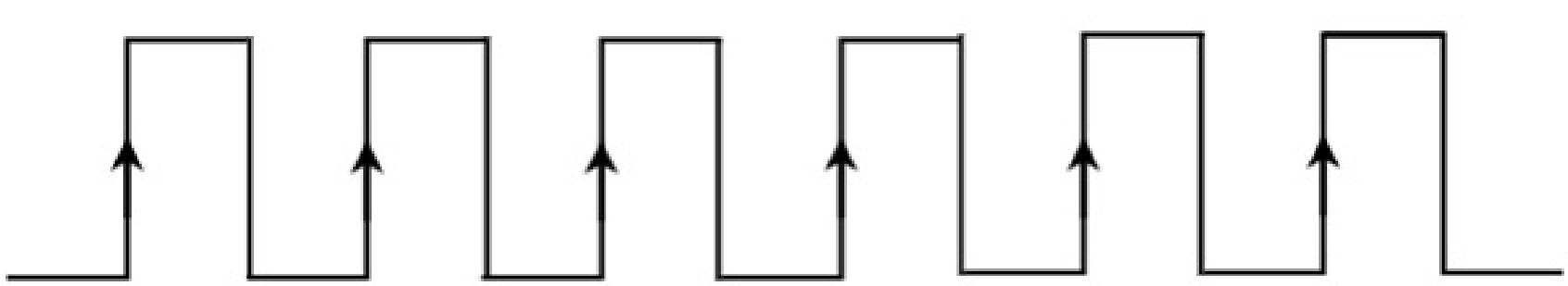
EDGE TRIGGERING TYPES

- Positive Edge triggering
- Negative Edge triggering

POSITIVE EDGE TRIGGERING

The circuit activates its output when the clock signal transitions from low (0) to high (1).

Visual Representation

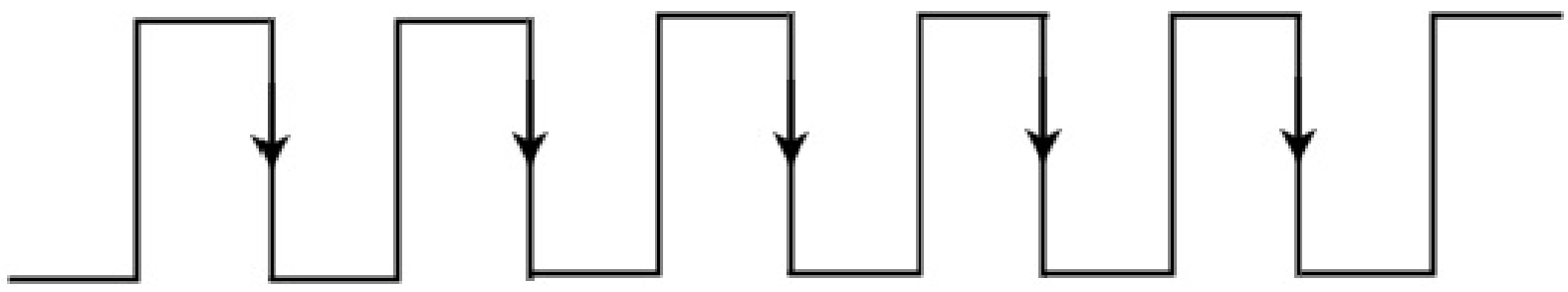


Positive edge triggering occurs on the rising edge of the clock signal.

NEGATIVE EDGE TRIGGERING

The circuit activates its output when the clock signal transitions from high (1) to low (0).

Visual Representation



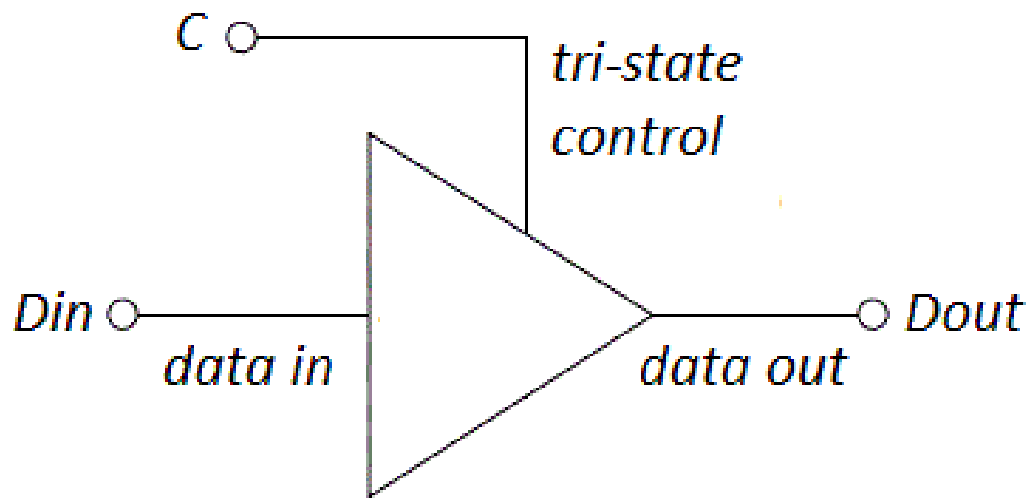
Negative edge triggering occurs on the falling edge of the clock signal.

@Sree Vishnu Varthini

Day - 10

Embedded Systems Programming

TRISTATE LOGIC



Truth Table

Enable PIN	IN	OUT
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

TRISTATE LOGIC

Tri-state logic is a digital logic design that allows an output to have **three** possible states instead of the usual two (high or low).

In traditional binary logic, a signal can either be:

- **High (1)** – representing a logical “1”
- **Low (0)** – representing a logical “0”

However, tri-state logic introduces a third state:

- **High Impedance (Z)** – a **disconnected or "off"** state, meaning the output is not driving any signal at all.

Think of this third state as a way for a device to "remove" itself from the circuit temporarily, avoiding interference with other signals on a shared connection. This is especially useful in shared bus systems, like data buses in a microcontroller or computer.

WORKING OF TRISTATE LOGIC

In a tri-state system, control signals determine whether the output is active or in the high impedance state. This control signal is called the **Enable signal,** typically labeled as **EN**.

- **When EN = 1:** The output is enabled and can drive either a "high" or "low" signal.
- **When EN = 0:** The output is in the high impedance (Z) state and effectively disconnected from the circuit.

IMPORTANCE OF TRISTATE LOGIC

- **Prevents signal conflict:** Only one device drives the signal at a time.
- **Efficient use of data buses:** Multiple devices can share the same line.
- **Flexibility in circuit design:** Tri-state logic allows for more efficient and scalable digital designs.

Practical Example: Using a Tri-State Buffer

A **tri-state buffer** is the simplest example of a tri-state device. It controls whether an input signal is passed to the output or disconnected (high impedance).

Input (A): The signal to be transmitted.
Enable (EN): Controls whether the signal is transmitted or not.

EN	A (Input)	Output
1	1	1
1	0	0
0	X	Z (high impedance)

- If **EN = 1**, the output follows the input (either 0 or 1).
- If **EN = 0**, the output is disconnected (Z), meaning it doesn't interfere with the bus.

@Sree Vishnu Varthini

Day - 11

Embedded Systems Programming

RAM (RANDOM ACCESS MEMORY)

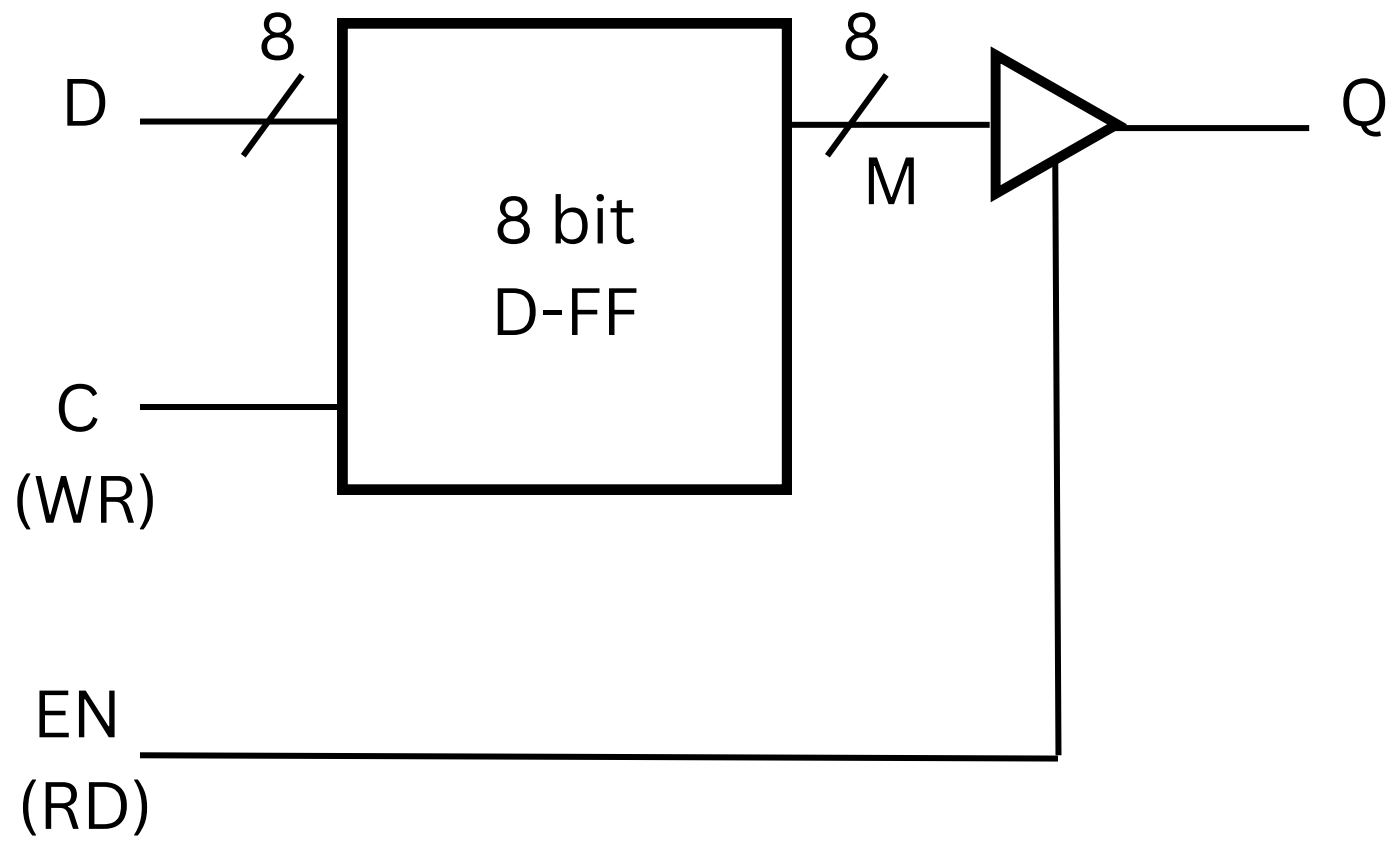
Random Access Memory (RAM) is a type of computer memory that **temporarily** stores data while the computer is running.

- It's called **"random access"** because the computer can quickly access any part of the memory directly.

- RAM is part of the main memory, allowing the system to perform both reading and writing operations. Hence, it is also known as **Read-Write Memory.**

- RAM is **volatile** i.e., it stores data as long as the computer is powered on and all data is erased when the computer is turned off.

1 BYTE READ WRITE MEMORY



- When **C = 1** and **EN = 0**, **Write** operation will be performed.
- When **EN = 1** and **C = 0**, **Read** operation will be performed.
- When **C = 0** and **EN = 0**, **Store** operation will be performed

WORKING OF RAM

👉 RAM is made up of tiny transistors and capacitors that store electric charges corresponding to data bits.

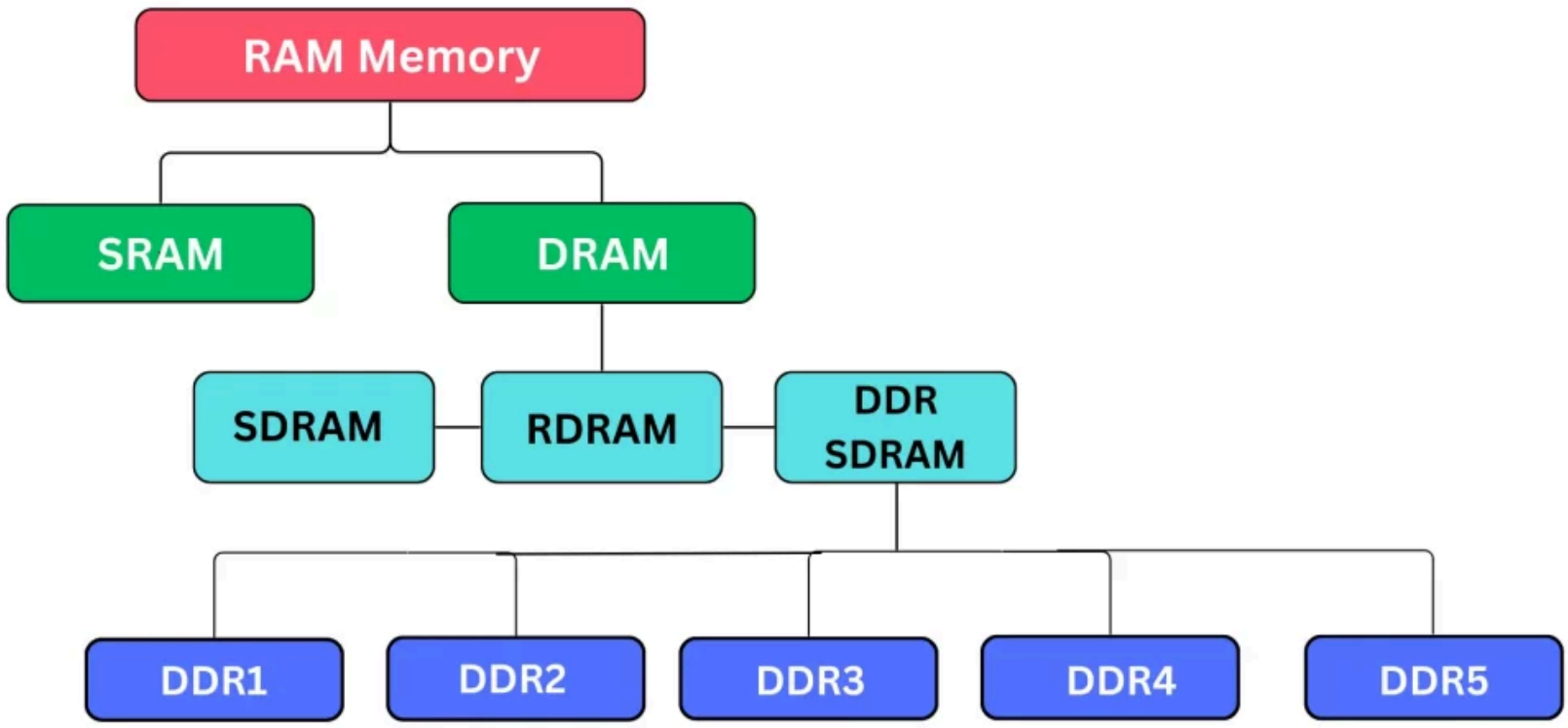
👉 The capacitors need a regular electric charge to retain the data.

👉 If this charge isn't refreshed, the data stored in RAM is lost as the capacitors lose their charge.

TYPES OF RAM

📍 DRAM (Dynamic RAM)
📍 SRAM (Static RAM)

TYPES OF RAM MEMORY



@Sree Vishnu Varthini

Day - 12

Embedded Systems Programming

HISTORY BEHIND THE INVENTION OF RAM

SEQUENTIAL ACCESS MEMORY (SAM)

Sequential Access Memory (SAM) is a type of data storage that retrieves information in a **fixed** order.

Think of it like a **magnetic tape** from old movie reels. To access specific information, you have to **roll the tape forward or backward** to the exact spot—there's no jumping directly to the data you need.

WORKING OF SAM

- Data is accessed **sequentially**, meaning the retrieval follows a strict order.
- The access time depends on the **location** of the data, so reaching some pieces of information takes longer than others.

LIMITATIONS OF SAM

- **Slow Data Access:** Since you have to go through data in sequence, reaching the required information takes time.
- **Variable Access Time:** Depending on where the data is located, it could take seconds or much longer, making SAM inefficient for real-time applications.

INVENTION OF RAM

Engineers wanted to solve this issue by ensuring all data could be accessed at the **same speed, no matter its location.**

This led to the creation of **Read Write Memory (RWM),** which they named **Random Access Memory (RAM)**.

Interestingly, **Read-Only Memory (ROM)**, developed later, is also a type of random access memory!

RAM IN A NUTSHELL

📍 Random Access Memory (RAM) is volatile memory that temporarily stores data while a computer runs.

📍 It allows fast, direct access to any location, supports both read and write operations, but loses all data when the power is off.

ROM IN A NUTSHELL

📍 Read-Only Memory (ROM) is also a type of random access memory, but as the name suggests, it can only be read from, not written to.

📍 Like RAM, ROM allows fast access to any data, but the data is permanent and cannot be altered.

@Sree Vishnu Varthini

Day - 13

Embedded Systems Programming

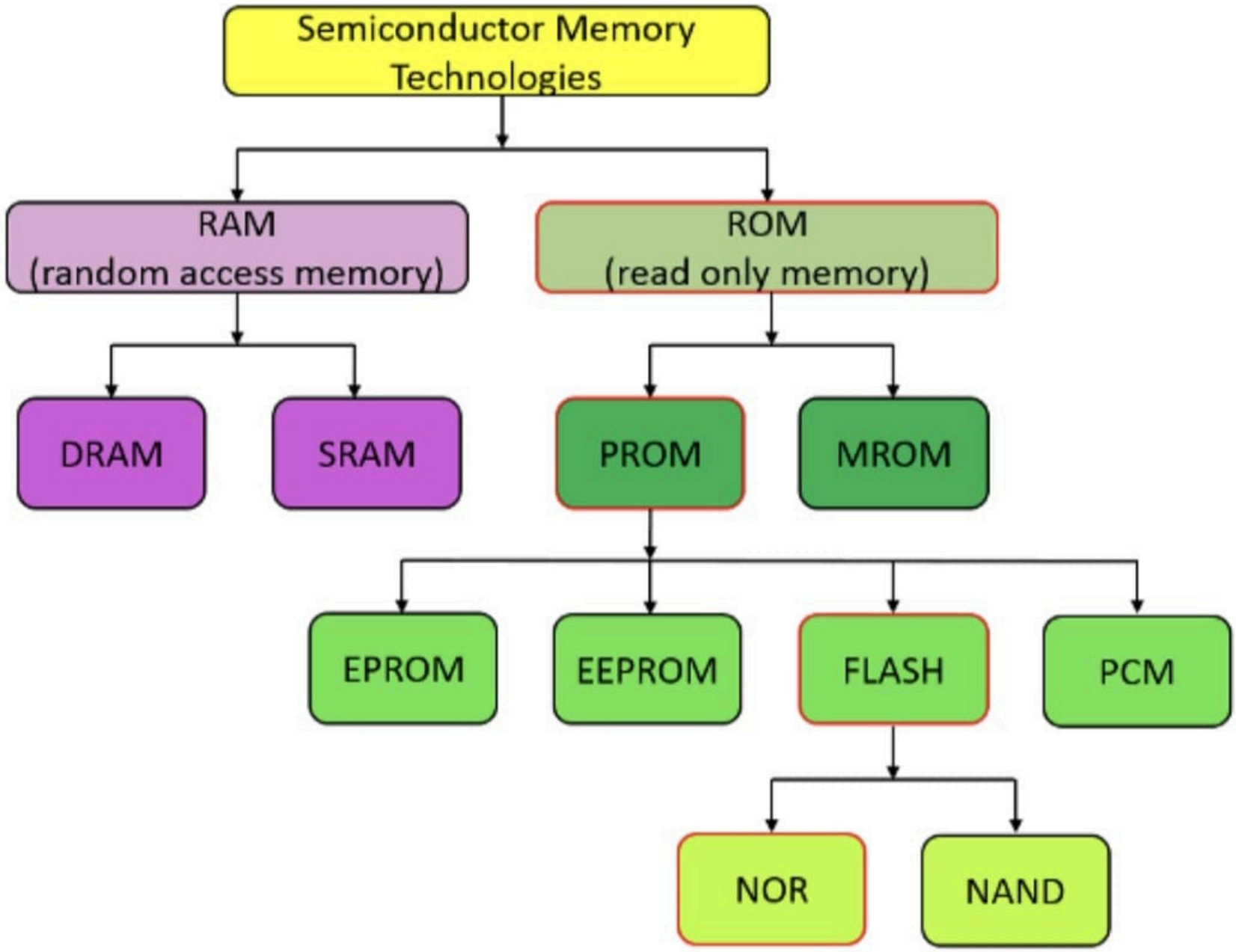
READ ONLY MEMORY - ROM

ROM (Read-Only Memory) is a non-volatile memory used to store essential data required to operate a system. As the name suggests, it can **only be read**; data stored in ROM **cannot be modified or erased**.

KEY FEATURES

- **Non-volatile:** Data remains even when power is off.
- **Permanent storage:** Information stored cannot be modified.
- **Read-only:** We can only access and read the data, not write over it.

TYPES OF ROM


Semiconductor Memory Technologies
RAM (random access memory)
ROM (read only memory)
DRAM
SRAM
PROM
MROM
EPROM
EEPROM
FLASH
PCM
NOR
NAND

TYPES OF ROM

✳️ MROM (Masked ROM)

- Original type of ROM that is permanently programmed during manufacturing.
- Cannot be altered after production.

✳️ EPROM (Erasable Programmable ROM)

- Type of ROM that can be erased and reprogrammed.
- Data is erased using ultraviolet (UV) light.

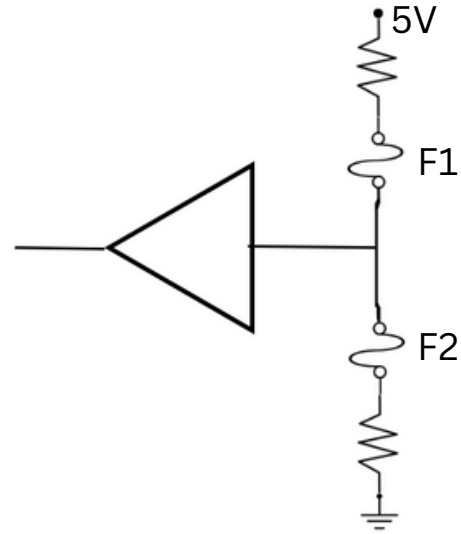
✳️ EEPROM (Electrically Erasable Programmable ROM)

- Allows erasing and reprogramming of individual memory locations.
- Can be erased electrically, without UV light.

TYPES OF ROM

❇️ PROM (Programmable ROM)

- One-time programmable ROM.
- Can be programmed once after manufacturing.



- **To set the enable signal to 0: Remove Fuse 1.** This will permanently disable the enable signal.

- **To set the enable signal to 1: Remove Fuse 2.** This will permanently enable the signal.

Once a fuse is removed, the corresponding value cannot be changed again.

TYPES OF ROM

✳️ Flash ROM

- Type of EEPROM that allows block-level erasure and reprogramming.
- Entire blocks can be erased at once.

✳️ NOR Flash

- A type of Flash ROM with random access capabilities.

✳️ NAND Flash

- Type of Flash memory designed for sequential access.

@Sree Vishnu Varthini

Day - 14

Embedded Systems Programming

MICROPROCESSOR

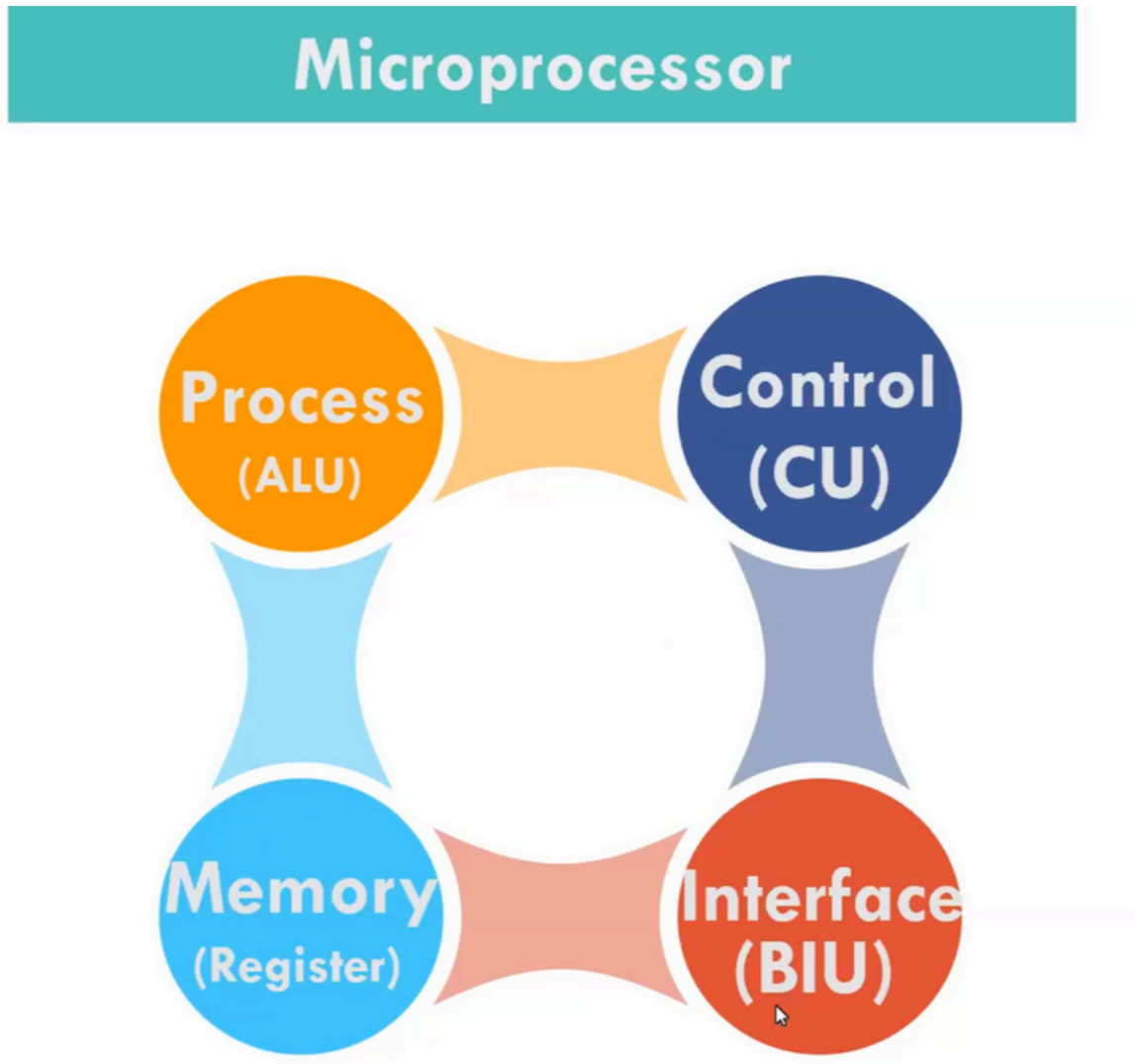
- A **microprocessor** is a compact digital device on a single integrated circuit (IC) or a small set of ICs.

- It handles data processing and control functions, fetching instructions from memory, decoding and executing them, and delivering results.

- A microprocessor is built from logical circuits, which are, in turn, composed of gates made from transistors.

MICROPROCESSOR

It performs four main functions: processing, control, memory, and interfacing.



CORE FUNCTIONS

- **Process:**

This is managed by the Arithmetic and Logic Unit (ALU), which handles all calculations and logical operations.

- **Control:**

The Control Unit directs the overall system operations, directing how tasks are executed within the microprocessor.

- **Memory:**

Responsible for data storage, this unit includes registers and various other memory components essential for holding instructions and data.

CORE FUNCTIONS

- **Interface:**

This unit bridges the microprocessor's internal components and connects it to external devices, enabling communication with the outside world.

KEY OPERATIONS

📥 **Reads from external memory and writes to internal memory :** Fetches data or instructions from external memory and stores them internally.

📤 **Reads from internal memory and writes to external memory :** Sends processed data from internal memory to external storage or devices.

@Sree Vishnu Varthini

Day - 15

Embedded Systems Programming

MICROPROCESSOR LANGUAGE

- **Assembly language** is a low-level programming language that is used to program microprocessors.

- Each microprocessor has its **own unique** assembly language, used to communicate directly with its hardware.

- To **simplify** programming across different processors, the **C language** was developed as a common, high-level language.

MICROPROCESSOR LANGUAGE

- When a program is written in C, a **compiler** translates it into assembly language specific to the microprocessor being used.

- An **assembler** then converts this assembly code into machine language, which the microprocessor can directly understand and execute.

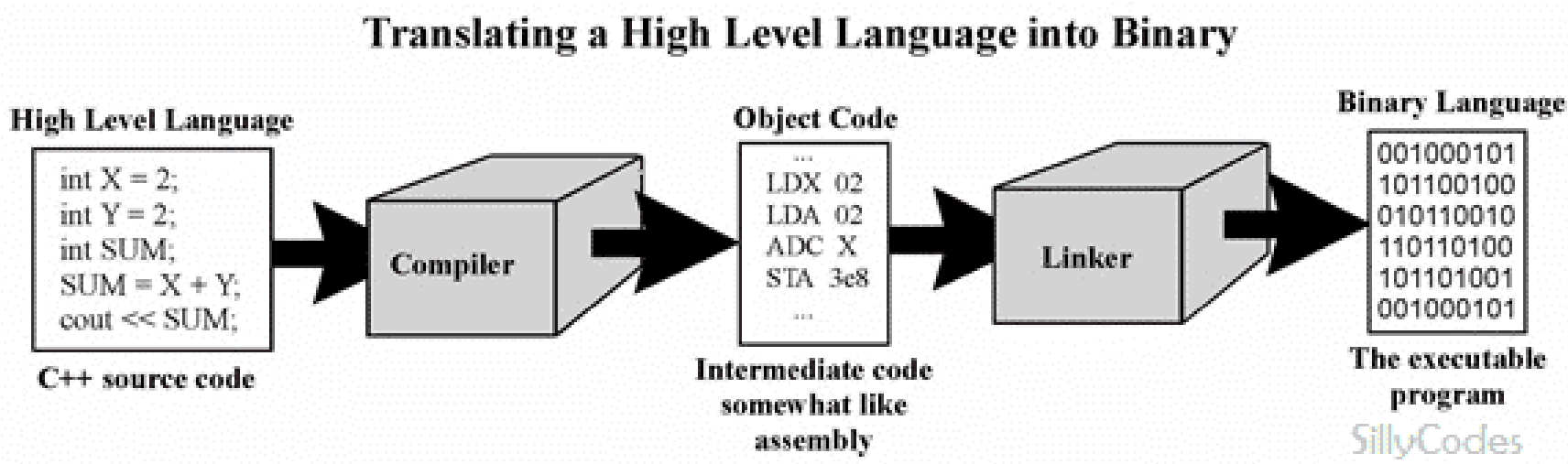
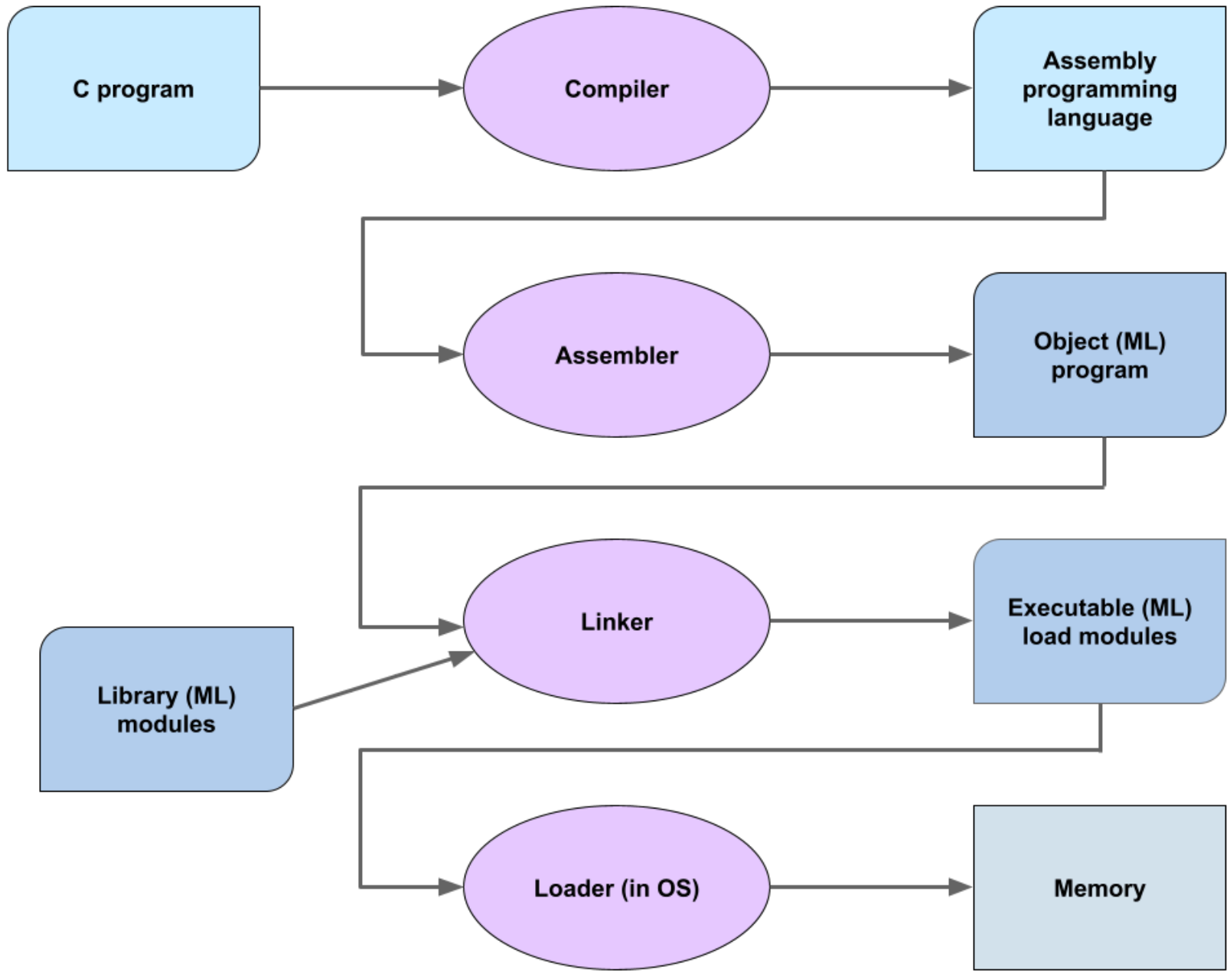


Diagram showing that a compiler and linker are both responsible for converting high-level languages to machine code.

MICROPROCESSOR LANGUAGE



MICROPROCESSOR LANGUAGE

Example Assembly code Snippet:

ld R0, 512

- **ld:** This stands for "**load**." It's an instruction that tells the CPU to load data into a register.
- **R0:** This specifies the target register where the data will be loaded. In this case, it is the register R0.
- **512:** This is the immediate value to be loaded into the register R0. In this example, the number 512 is being loaded.

@Sree Vishnu Varthini

Day - 16

Embedded Systems Programming

MICROPROCESSOR CLOCK

A **microprocessor** clock is a crucial component of a microprocessor that generates the timing signals needed for the processor to execute instructions and coordinate operations.

The microprocessor clock is an oscillator that produces a continuous square wave signal, typically measured in **Hertz (Hz).**

This clock signal determines **the timing and speed** at which the microprocessor operates, allowing it to synchronize its internal operations and communicate with other components in a computer system.

KEY CHARACTERISTICS

- **Clock Frequency:** Frequency is the number of cycles that occur in one second and is measured in hertz (Hz).

- **For example,** a frequency of 1 Hz means one cycle per second, while 1 MHz (megahertz) means one million cycles per second.

- A **higher clock frequency** means that the microprocessor can perform more cycles per second, leading to faster processing speeds.

KEY CHARACTERISTICS

- **Clock Cycle:** A cycle refers to one complete waveform of a clock signal. During each clock cycle, the microprocessor can execute an instruction, perform calculations, or access memory.

RELATIONSHIP

The relationship between cycle and frequency can be described mathematically:

Frequency (f) = 1 / Cycle Time (T)

Where:

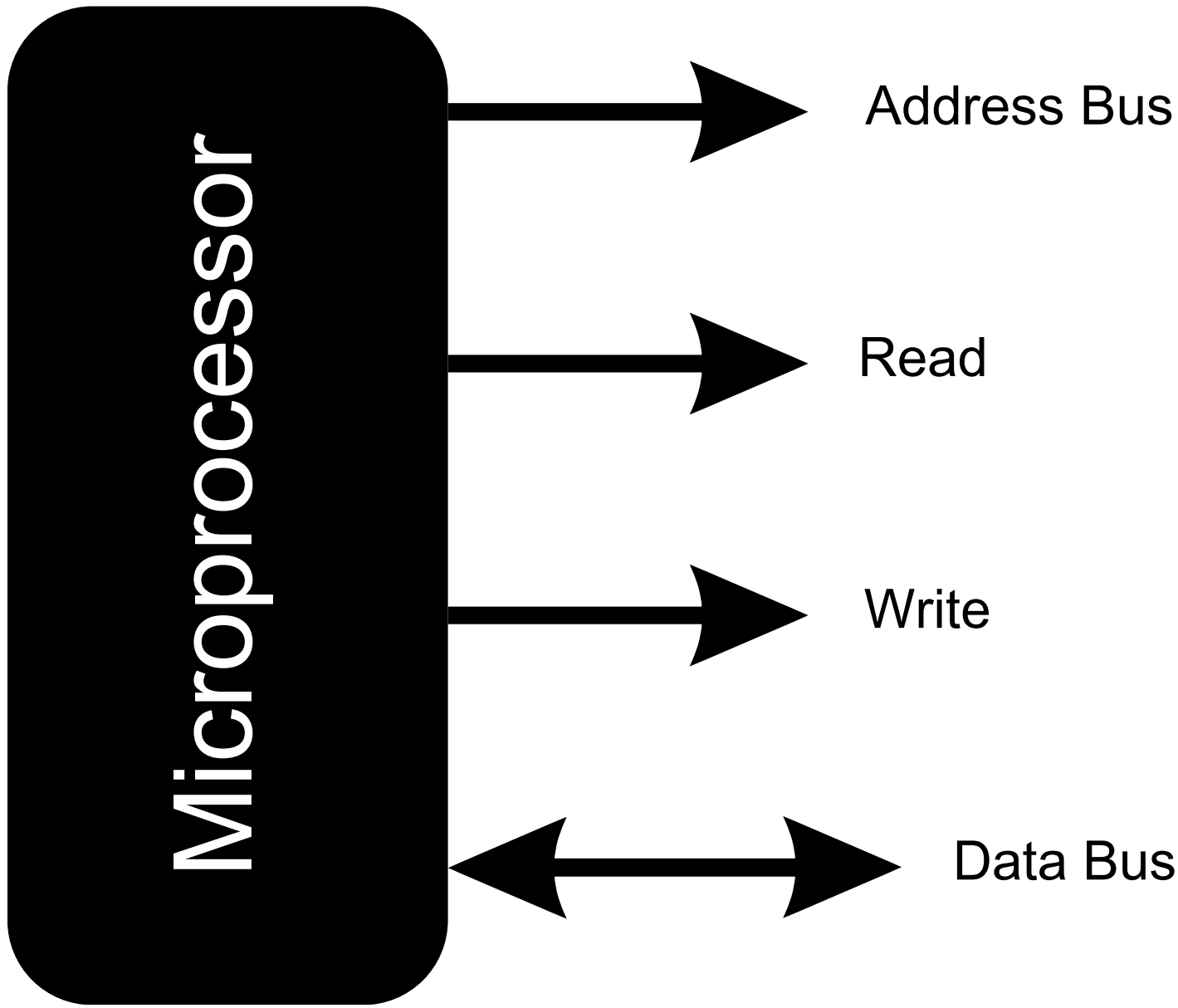
- **Frequency (f)** is measured in **hertz (Hz).**
- **Cycle Time (T)** is the duration of one cycle, measured in seconds.

@Sree Vishnu Varthini

Day - 17

Embedded Systems Programming

MICROPROCESSOR BUS SIGNALS



MICROPROCESSOR BUS SIGNALS

ADDRESS BUS SIGNALS

Unidirectional signals: Travel from the microprocessor to memory or I/O devices.

Purpose: Specify the address in memory or a peripheral device where data needs to be read from or written to.

DATA BUS SIGNALS

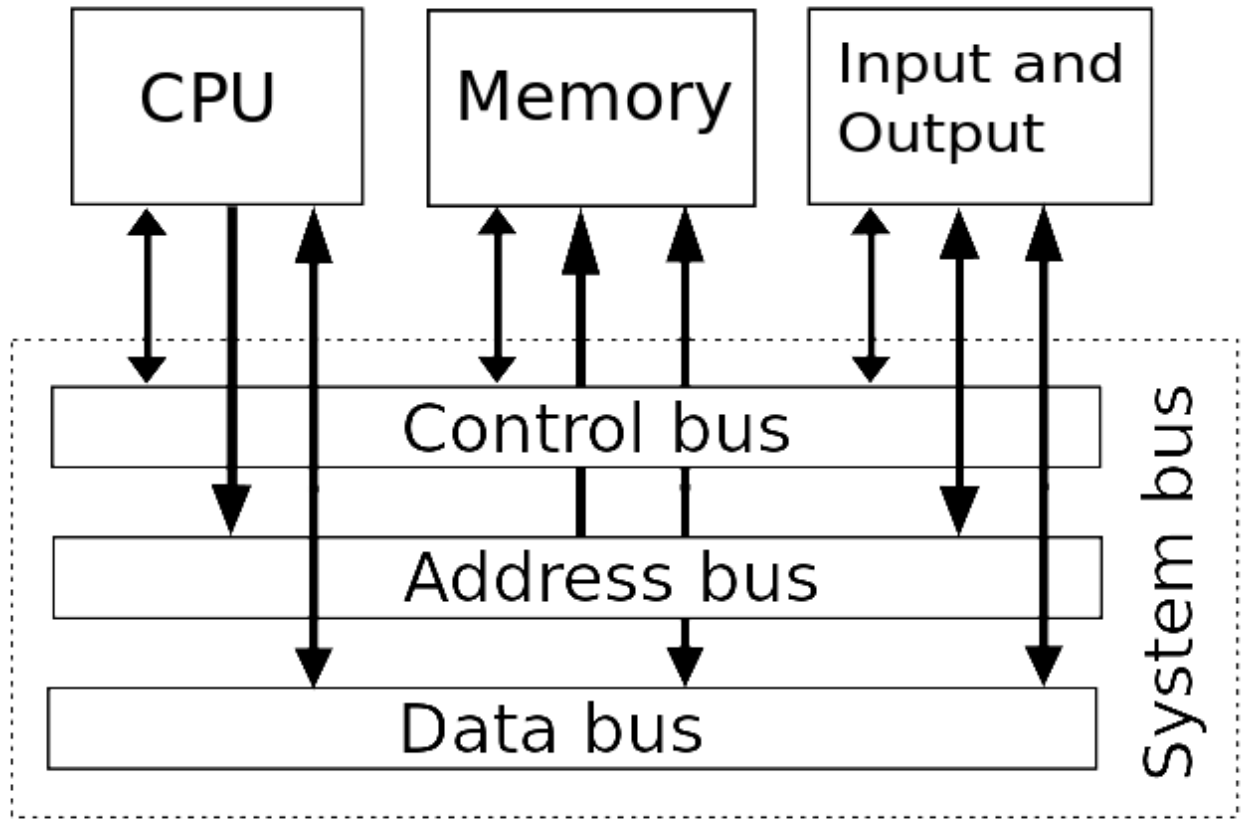
Bidirectional signals: Allow data to move in both directions, from the microprocessor to memory/peripherals and vice versa.

Purpose: Carry actual data that is being processed or transferred.

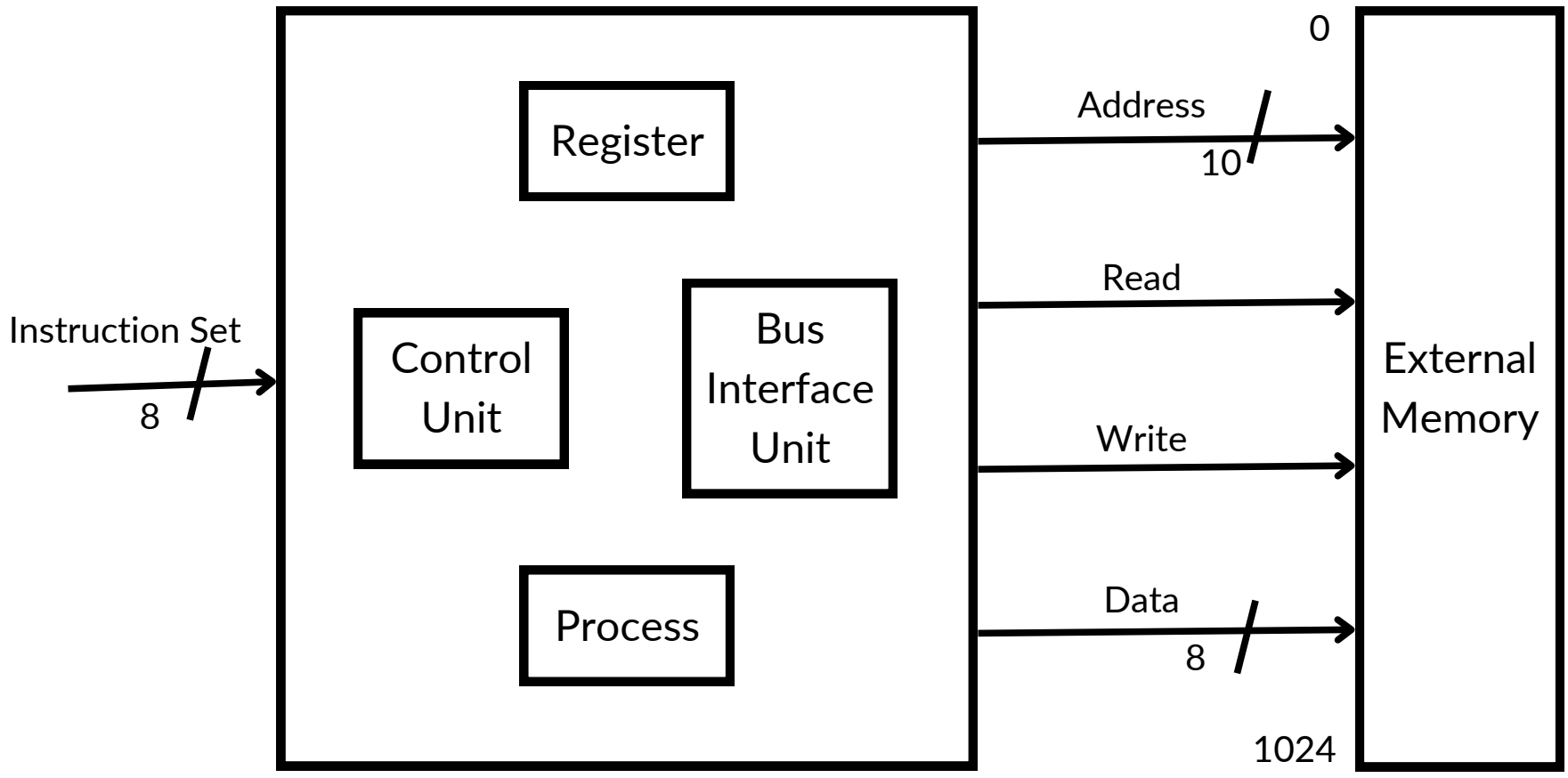
MICROPROCESSOR BUS SIGNALS

CONTROL BUS SIGNALS

Read (RD): Signals that the microprocessor wants to read data from a memory location or input device.
Write (WR): Signals that the microprocessor intends to write data to a memory location or output device.



BLOCK DIAGRAM

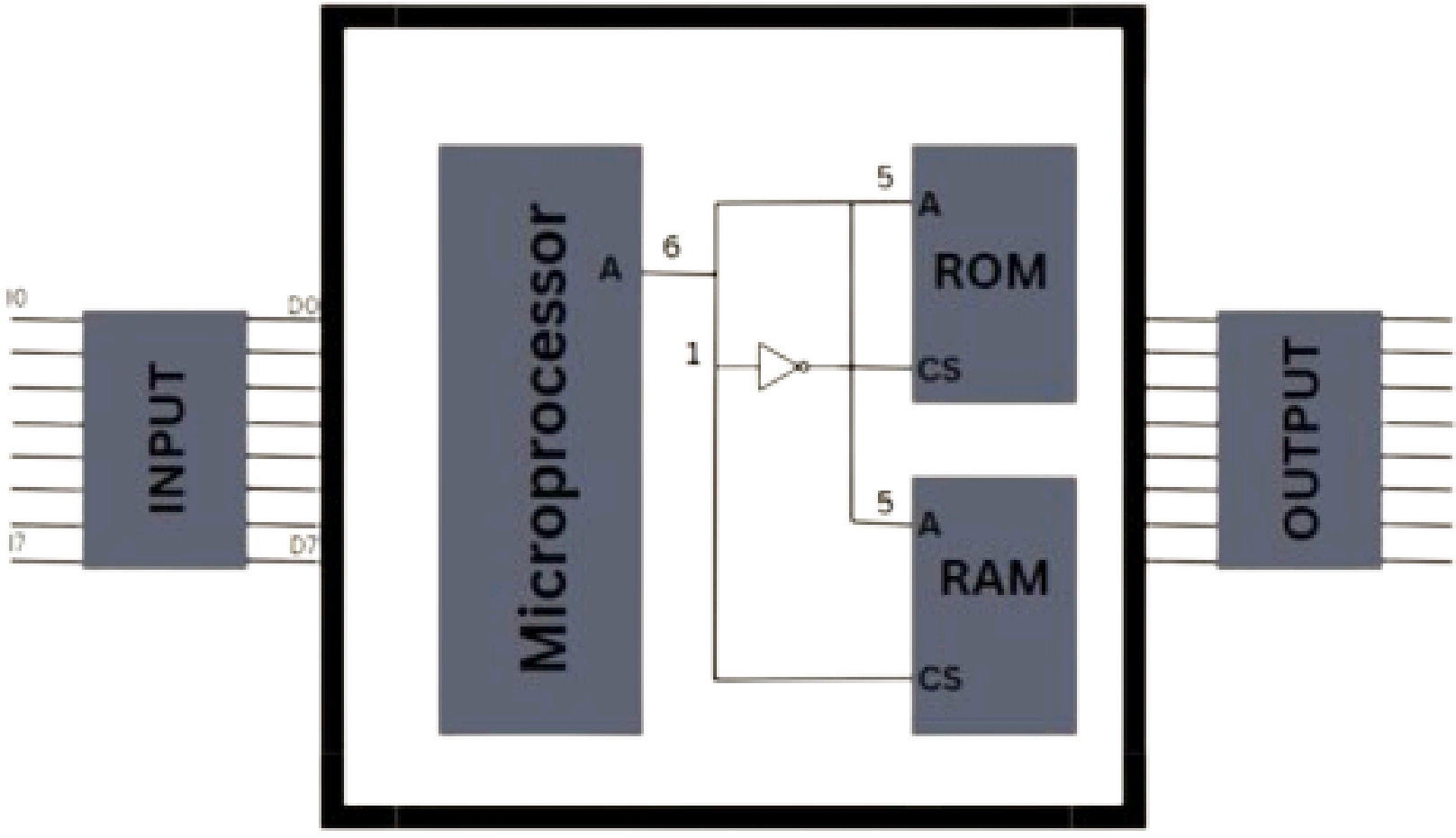


@Sree Vishnu Varthini

Day - 18

Embedded Systems Programming

MICROPROCESSOR COMPONENTS



Pic Credits: Manikanta Sai Vallamkonda

MICROPROCESSOR COMPONENTS

1. ROM (READ-ONLY MEMORY)

- **Purpose:** ROM stores permanent data and instructions essential for the system's basic functions.

- **Non-Volatile Memory:** ROM keeps its data even when power is off, making it ideal for firmware, boot instructions, and critical system code.

- **How it Works:** The microprocessor reads data from ROM but usually cannot write to it. ROM typically holds the bootloader and startup instructions needed to initialize the system.

MICROPROCESSOR COMPONENTS

2. RAM (RANDOM ACCESS MEMORY)

- **Purpose:** RAM provides temporary storage for data and instructions that the microprocessor is currently working on.

- **Volatile Memory:** RAM loses all data when the power is turned off, making it suitable for quick, temporary data access.

- **How it Works:** The microprocessor reads from and writes to RAM during operation. It can directly access any data location by specifying the memory address.

MICROPROCESSOR COMPONENTS

3. INPUT DEVICES

- **Purpose:** Allow data and commands to enter the system from external sources.

- **Examples:** Keyboards, mouse, sensors, touchscreens, or any device that provides input to the processor.

- **How it Works:** Input devices send data to the microprocessor through input ports. The microprocessor then processes this data according to the instructions it has been programmed to execute.

MICROPROCESSOR COMPONENTS

4. OUTPUT DEVICES

- **Purpose:** Display or convey the results of the microprocessor's operations to the user.

- **Examples:** Monitors, LEDs, printers, speakers, or any component that outputs information to the user.

- **How it Works:** The microprocessor sends data to the output device through output ports, allowing users to view or receive the processed data.

MICROPROCESSOR COMPONENTS

INTEGRATION IN A MICROPROCESSOR SYSTEM

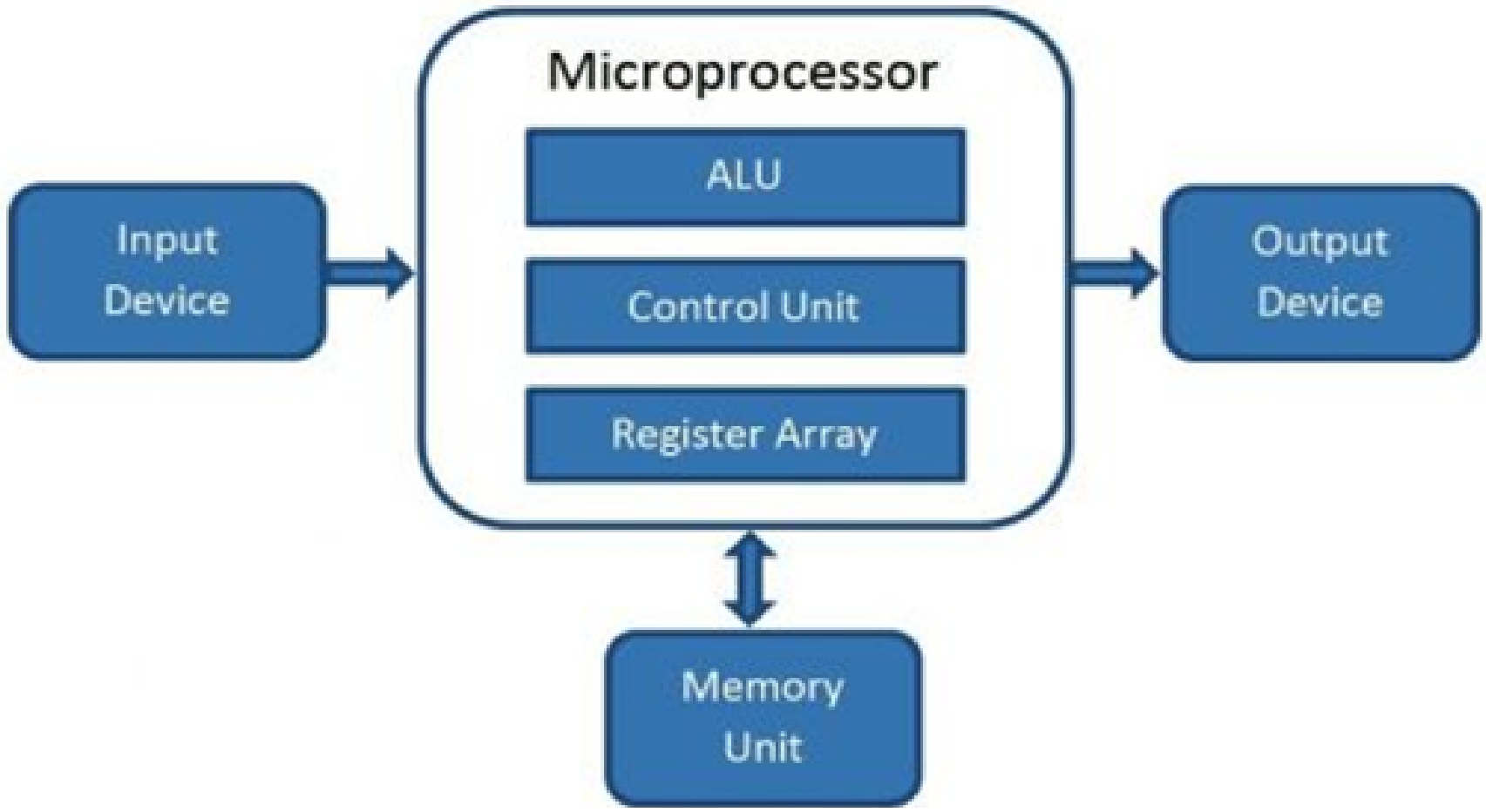
ROM: Holds primary programs like start-up and operational code.

RAM: Temporarily stores data, variables, and instructions during execution.

Input Devices: Send signals or data for processing.

Output Devices: Display or act on processed data, allowing user or environmental interaction

BLOCK DIAGRAM

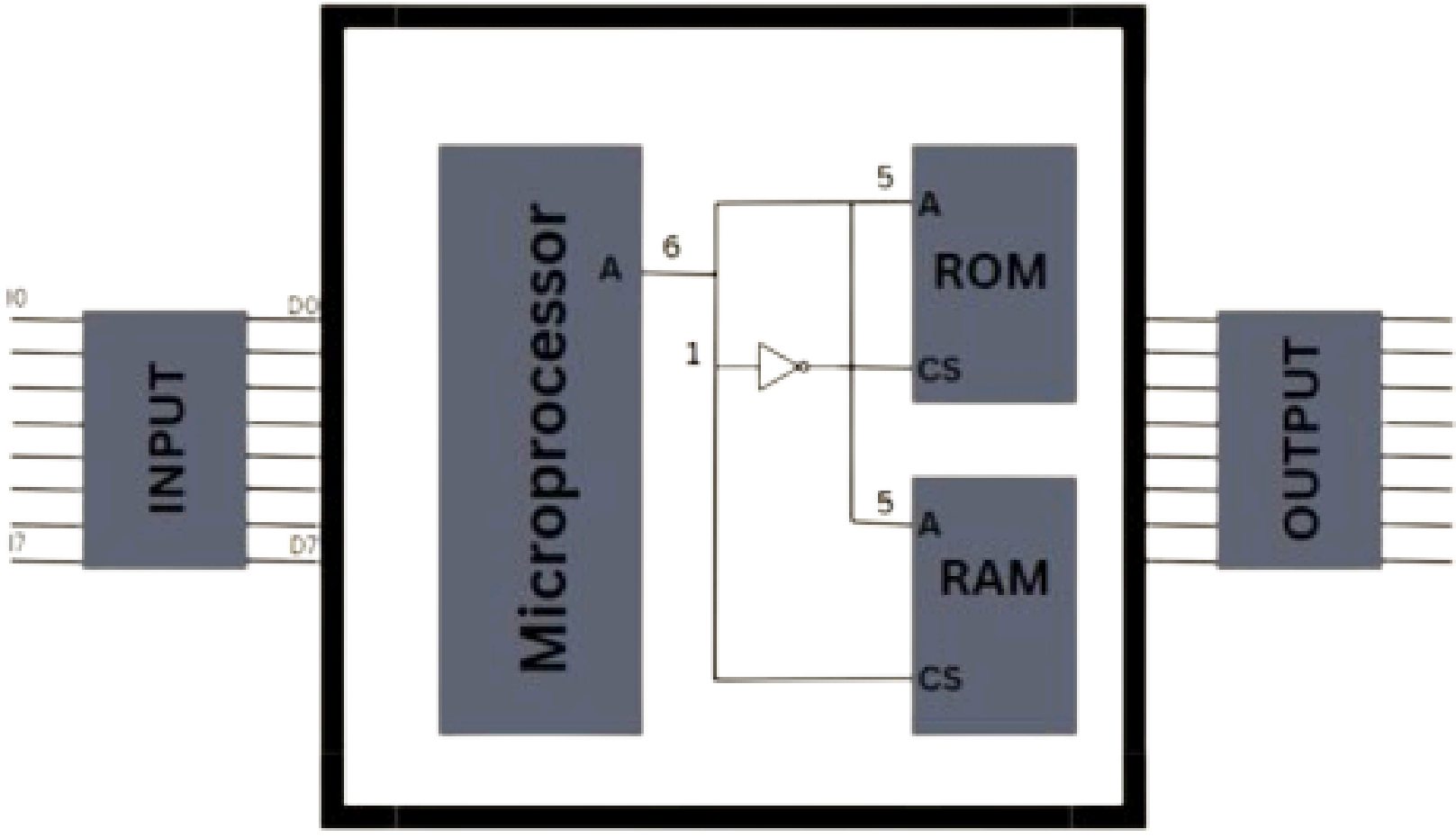


@Sree Vishnu Varthini

Day - 19

Embedded Systems Programming

MICROPROCESSOR COMPONENTS



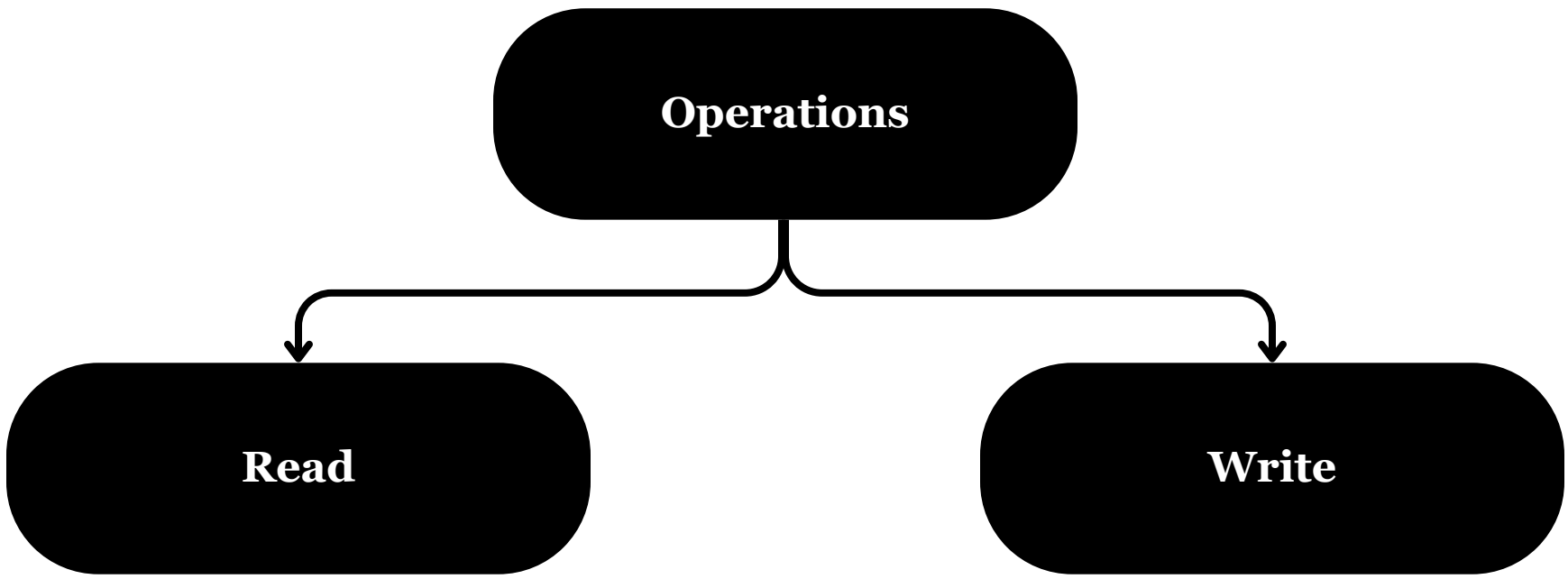
Pic Credits: Manikanta Sai Vallamkonda

MICROPROCESSOR OPERATIONS

A microprocessor performs different operations to handle data.

The two main types of operations are :

- Write Operations (Sending Data)
- Read Operations (Receiving Data)



WRITE OPERATIONS (SENDING DATA)

1. Write to Memory

- The microprocessor stores (writes) data into the computer's memory (RAM or other storage).
- **Example:** When you type a document, the text gets written to memory before you save it.

2. Write to Output

- The microprocessor sends data to an output device (such as a screen, printer, or speaker).
- **Example:** When you press a key on the keyboard, the letter appears on the screen because the processor writes data to the display.

READ OPERATIONS (RECEIVING DATA)

1. Read from Memory

- The microprocessor retrieves (reads) data from memory when needed.
- **Example:** When you open a saved file, the processor reads the data from memory to display it on your screen.

2. Read from Input

- The microprocessor gets data from an input device (such as a keyboard, mouse, or sensor).
- **Example:** When you move the mouse, the processor reads the movement data and moves the cursor accordingly.

@Sree Vishnu Varthini

Day - 20

Embedded Systems Programming

MICROPROCESSOR - INPUT LOGIC

WRITE OPERATION DEMONSTRATION

Address Range	Memory Type
0 - 15	ROM (Read-Only Memory)
16 - 31	RAM (Read-Write Memory)
32 - 47	OUTPUT (Output Devices)
48 - 63	INPUT (Input Devices)

The microprocessor interacts with these memory regions using **read and write operations**.

Code Snippet

```
int *x;
x = 33; // address line 33 (OUTPUT) selected
*x = 1; // writing value 1 in location 33
```

STEP-BY-STEP BREAKDOWN:

1. **int *x;**
    - Declares a pointer x, which will store a memory address.
2. **x = 33;**
    - Assigns address 33 to x.
    - The pointer x now points to address 33, which is inside the OUTPUT (32 - 47) range.
3. ***x = 1;**
    - This tells the microprocessor to write the value 1 to memory address 33.
    - **Microprocessor operation:**
        - Address Bus selects address 33.
        - Data Bus sends value 1.
        - Control Bus sets the signal to WRITE mode.
        - The value 1 is written at address 33 in OUTPUT.

💡 **Result:** Memory location 33 (OUTPUT) now holds the value 1.

READ OPERATION DEMONSTRATION

Address Range	Memory Type
0 - 15	ROM (Read-Only Memory)
16 - 31	RAM (Read-Write Memory)
32 - 47	OUTPUT (Output Devices)
48 - 63	INPUT (Input Devices)

The microprocessor interacts with these memory regions using **read and write operations**.

Code Snippet

```
int *x, y;
x = 49; // address line 49 (INPUT) is selected
y = *x; // reading operation done and value in
address 49 will be stored in y
```

STEP-BY-STEP BREAKDOWN:

1. **int *x, y;**
    - Declares a pointer x which will store a memory address and an integer y.
2. **x = 49;**
    - Assigns address 49 to x.
    - The pointer x now points to address 49, which is inside the INPUT (48-63) range.
3. **y = *x;**
    - This tells the microprocessor to read the value stored at address 49.
    - **Microprocessor operation:**
        - Address Bus selects address 49.
        - Control Bus sets the signal to READ mode.
        - Data Bus retrieves the value stored at address 49 and assigns it to y.

💡 **Result:** The value stored at address 49 (INPUT) is now in variable y.

@Sree Vishnu Varthini

Day - 21

Embedded Systems Programming

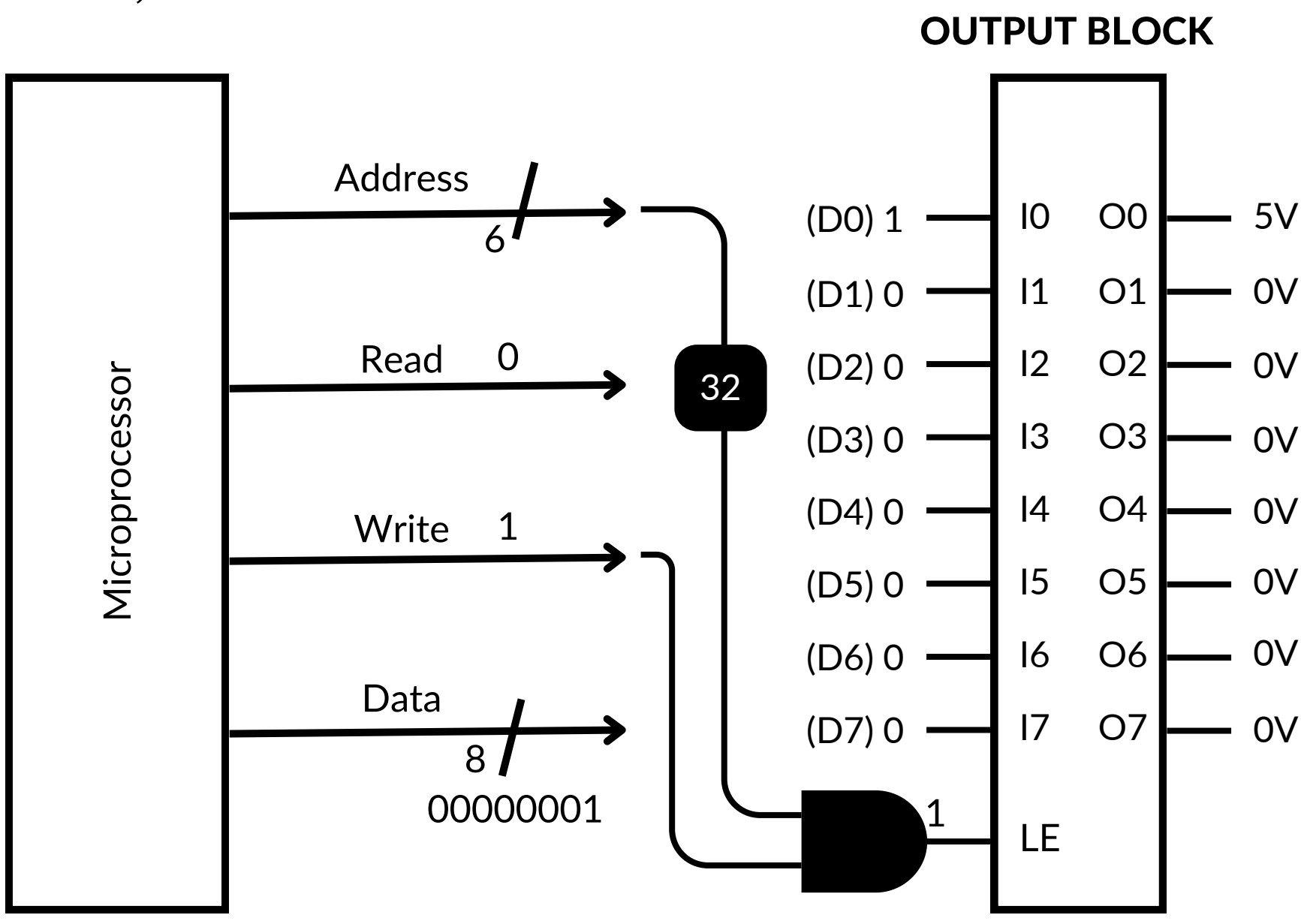
MICROPROCESSOR - OUTPUT LOGIC

Example Demonstration

int *x;

x = 32;

*x = 1;



Microprocessor- Output Logic - Explanation

When the **Write signal** is set to **1** and the **address** is set to **memory location 32**, the **latch enable** (AND gate) is activated, allowing the corresponding output block at address 32 to be enabled.

Once enabled, the 8-bit data from the data bus is written to the output block. Each bit in the data determines the corresponding output state:

- If a bit is **1**, the corresponding output pin is set to **5V**, turning **ON** connected devices (such as LEDs or relays).
- If a bit is **0**, the output remains at **0V**, keeping the device **OFF**.

In this example, the data 00000001 (binary) results in only the first output (O0) being set to 5V, while the rest remain at 0V. This means only the first connected device (e.g., an LED) will be switched ON.

@Sree Vishnu Varthini

Did you like the post?

follow for more!

P.S. You can access the **Embedded Systems Programming** course by contacting **Balajee Seshadri** Sir through the WhatsApp number provided on his linkedin profile.