PLC REPEATED NOTES FULL

📌 DEFINITION OF A PLC (Programmable Logic Controller)

A Programmable Logic Controller (PLC) is a solid-state, specialized industrial computer built for **harsh environments**, **storing** and **executing** control instructions to manage machines or processes in real-time with extreme reliability, essentially acting as the rugged "brain" of industrial automation engineered for 24/7 operation.

⚙️ TYPICAL INSTRUCTIONS STORED IN A PLC

When you write a PLC program, you're basically feeding it a bunch of instructions. Here are the classics:

🔄 Sequencing

Controlling the step-by-step progression of processes, much like following a TikTok recipe, is achieved through sequencing, often utilizing methods such as step coils, state machines, or sequential function charts.

🛺 Timing

Timers (like ON-delay, OFF-delay, and retentive timers) are crucial for precisely controlling time-based actions and transitions in processes, allowing you to implement delays or specific durations, such as waiting a set number of seconds before opening a valve.

🔢 Counting

Counters, including up-counters, down-counters, and combined counter-timer logic, precisely track events and control actions, such as counting products on a line and triggering an event when a specific number is reached, while also enabling batching and overflow management.

➕ Arithmetic

Supporting both integer and floating-point operations, arithmetic functions allow for basic mathematical calculations on machine data, such as adding, subtracting, multiplying, or dividing numbers, which is essential for scaling sensor data, calculating setpoints, or implementing custom algorithms.

📦 Data Manipulation

Bit-level and word-level operations, including masking, shifting, rotating, and BCD conversions, facilitate moving, comparing, or swapping numbers like playing cards, enabling complex signal conditioning or data packing and unpacking.

🌐 Communication

Industrial protocols like Modbus, Profibus, and Ethernet/IP enable PLCs to "gossip" with each other, HMIs, or SCADA systems, thus coordinating with external devices, peer PLCs, or supervisory systems.

Typical Instructions in a PLC program:

- 🔄 **Sequencing** – step-by-step process control.
- 🕑 **Timing** – timers for delays and durations.
- 🔢 **Counting** – counting events, products, or cycles.
- ➕ **Arithmetic** – math on inputs, outputs, or variables.
- 📦 **Data Manipulation** – move, compare, or edit data.
- 🌐 **Communication** – talk to other devices and systems.

📌 What Are PLCs Used For?

A PLC's whole reason for existing is **to control and automate stuff in the real world** — especially in places too rough or complex for a normal computer.
Here's where you'll see them flexing:

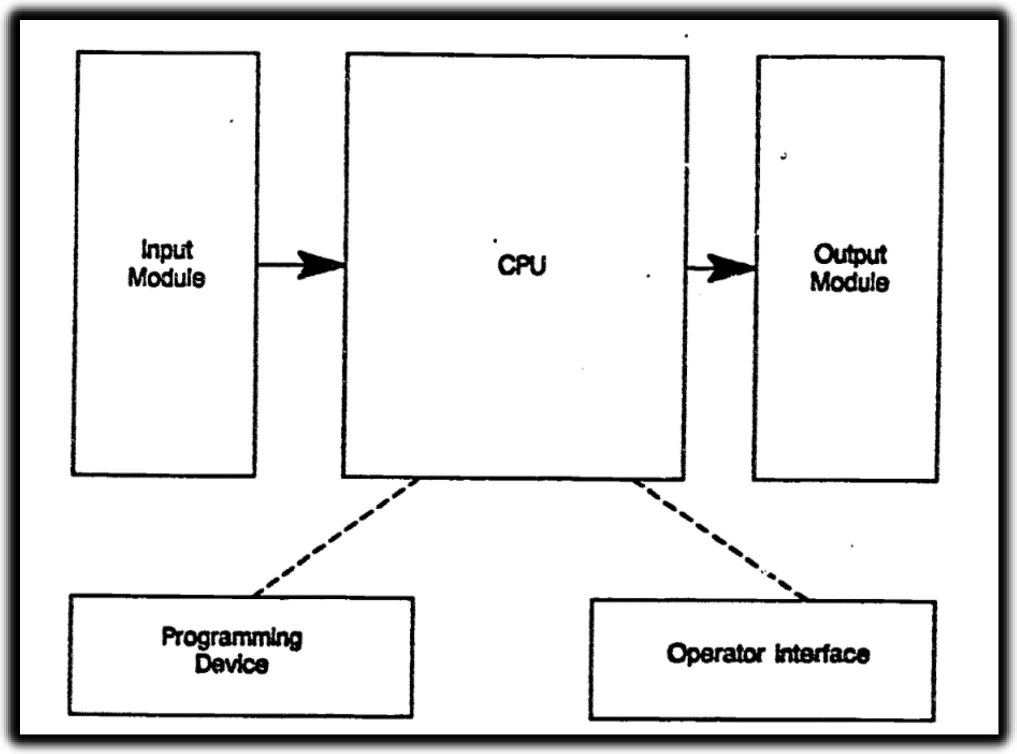
- ✅ **Machine Control:**
  They're the brain behind automated machines on factory floors — controlling motors, valves, and sensors with split-second precision.
- ✅ **Car Wash Systems:**
  Every step in an automatic car wash — soap, brushes, rinse, dryers — is sequenced by a PLC so your car gets clean without human intervention.
- ✅ **Bottling & Packaging Lines:**
  PLCs handle the timing, counting, and coordination needed to fill bottles, cap them, label them, and pack them at insane speeds.

- ✅ **Material Handling:**
  Conveyor belts, robotic arms, elevators, and palletizers all rely on PLCs to move products smoothly and safely through a plant.
- ✅ **Data Acquisition:**
  PLCs don't just control — they also **collect data** from sensors (temperatures, pressures, flow rates) and send it to HMIs, SCADA systems, or cloud dashboards.
- ✅ **Pipeline Monitoring:**
  Oil, gas, and water pipelines use PLCs to watch pressures, control valves, and trigger alarms or shutdowns if something goes wrong.
- ✅ **Hydroelectric Dams:**
  Gates, turbines, and safety interlocks are coordinated by PLCs to maintain power output and protect equipment.
- ✅ **Process Control:**
  Industries like pharmaceuticals or refineries use PLCs to handle multi-stage processes with loops, PID control, and precise logic.
- ✅ **Food Mixing or Cooking:**
  In large-scale food plants, PLCs run mixers, heaters, and coolers to hit exact recipes, temperatures, and timings.
- ✅ **Chemical Processing:**
  When handling chemicals, you need perfect timing, flow control, and safety shutdowns — PLCs are built for that level of reliability.

💡 Bottom line:

If it moves, mixes, counts, heats, cools, measures, or sequences — a PLC can control it. They're everywhere in modern automation, silently running the show behind the scenes.

Typical PLC System Components



A PLC isn't just a single magic box — it's a **team of modules** working together:

✅ Input Module

- Brings signals from the field (sensors, switches, limit detectors).
- Converts those real-world electrical signals into data the CPU can read.

✅ CPU (Central Processing Unit)

- The *brain* of the PLC.
- Reads input data, runs your program logic, makes decisions, and figures out what outputs should do.

✅ Output Module

- Takes decisions from the CPU and drives actuators (motors, solenoids, lights, valves) in the real world.
- Converts CPU signals into the correct power levels for field devices.

✅ Programming Device

- This is how *you* talk to the PLC.
- Used to create, download, and edit the logic (ladder diagrams, structured text, etc.).
- Think laptops, handheld programmers, or special config tools.

✅ Operator Interface (HMI – Human Machine Interface)

- The dashboard for operators.
- Shows process info (temperatures, states, alarms) and lets operators tweak parameters (like setting a new speed or temperature).

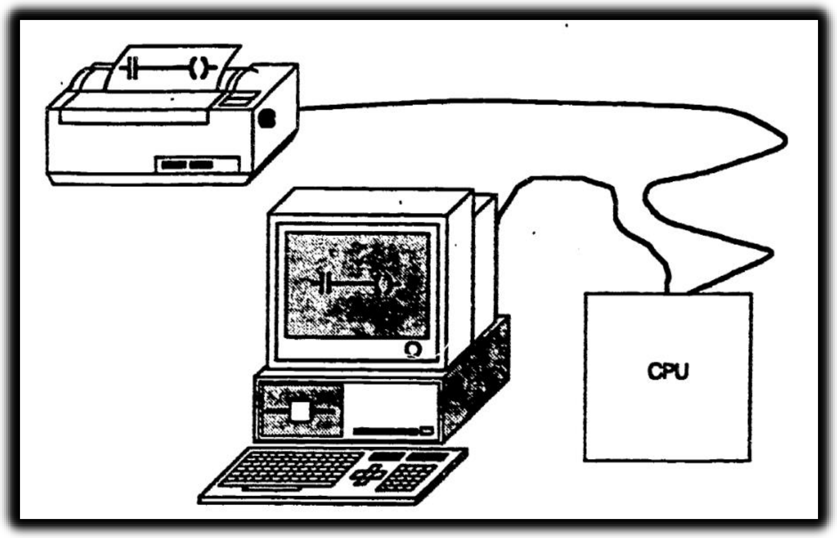
✨ How it all flows (based on the diagram):

1. **Input Module ➞ CPU:** Sensors send signals → CPU reads them.
2. **CPU does the logic:** Your program decides what should happen.
3. **CPU ➞ Output Module:** Sends commands to field devices.
4. **Programming Device:** Engineers use it to load or change the logic.
5. **Operator Interface:** On the plant floor, operators monitor and adjust the system in real time.

💡 Big picture:

A PLC is basically *eyes (inputs), a brain (CPU), hands (outputs)* — with engineers (programming device) teaching it, and operators (HMI) talking to it while it's running.

TERMINAL



📌 Terminals in PLC Systems

A **terminal** is a hardware device that acts as a **bridge between a human and a computer system** — basically, it's how people interact with machines before modern graphical interfaces became common.

💻 Types of Terminals:

- **CRT (Cathode Ray Tube Terminal)**
    - ➢ Old-school monitor + keyboard setup.
    - ➢ Used for fast, real-time interaction with the system (entering commands, viewing status, editing programs).
    - ➢ Think of it as the OG "command-line interface" workstation in industrial settings.
- **PRINTER Terminal**
    - ➢ Not for controlling — but for **documenting**.
    - ➢ Used to get a printed version of your PLC program, error logs, process reports, or configuration files.

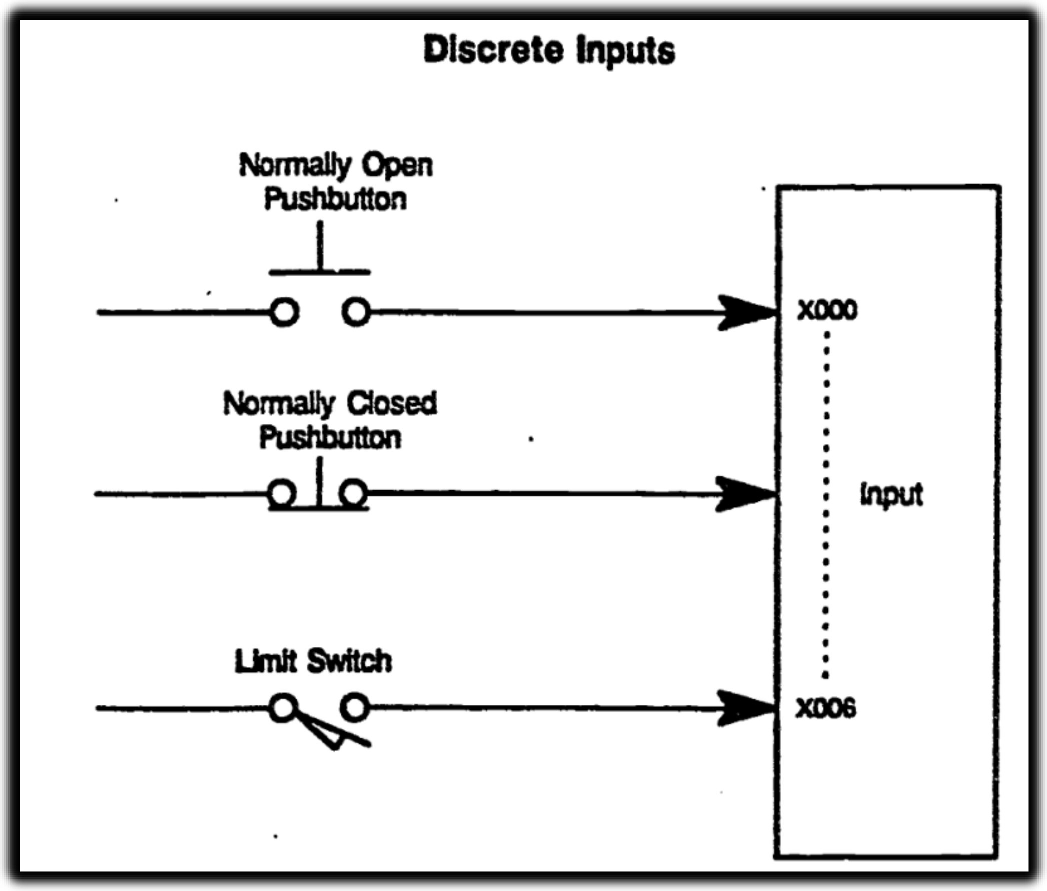
📃 Hardcopy

- Means a **physical printout**, usually on paper.
- For backups, records, or sharing with other engineers.

💡 Real Talk:

Before modern touchscreen HMIs and laptops, **terminals were the only way** to talk to your PLC. Today, they're mostly legacy — but understanding them helps when working on older systems or reading historical documentation.

📌 Discrete Inputs (a.k.a Digital Inputs)



💡 What is a Discrete Input?

A **discrete input** is a simple ON or OFF signal sent from a field device to the PLC.
It's **binary** — either HIGH (1) or LOW (0), no in-between.

🧠 Examples of Discrete Input Devices:

These devices act like *switches* — either giving power or not:

- 🟢 **Pushbuttons** (Normally Open / Normally Closed)
- ⚙️ **Limit switches** (detect positions of machine parts)
- 💧 **Float switches** (detect fluid levels)
- 🔄 **Toggle switches**
- 📈 **Flow switches / pressure switches**
- 🦶 **Foot pedals / safety interlocks**
- 🧲 **Proximity switches** (detect presence of objects without contact)

🔌 Wiring & Addressing: I/O Points

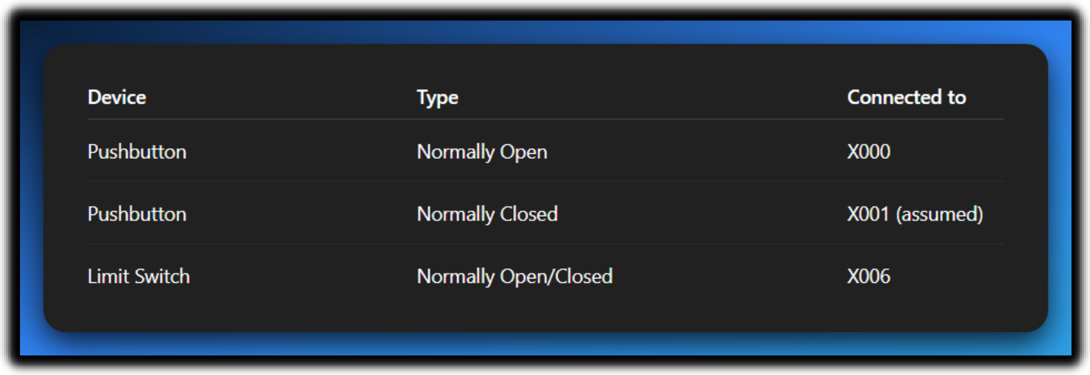
- **I/O points** are the terminals where field devices physically connect to the PLC input modules.
- Each input is mapped to a specific address in the PLC's memory.
  For example:

  X000 = 1st discrete input
  X006 = 7th discrete input

- This address can be auto-assigned or manually configured based on your PLC type and software.

✍️ Diagram Breakdown:

Let's decode the diagram you shared:



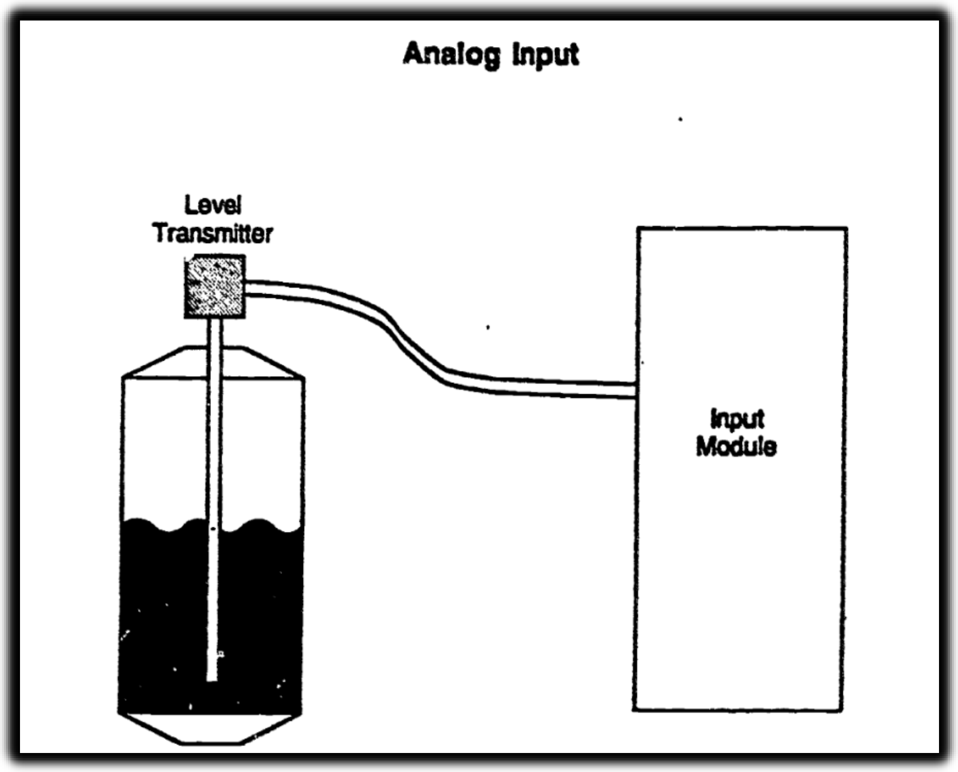
Device	Type	Connected to
Pushbutton	Normally Open	X000
Pushbutton	Normally Closed	X001 (assumed)
Limit Switch	Normally Open/Closed	X006

Each arrow represents a **signal wire** going into an **input point** on the PLC module. When the device is activated, the corresponding input bit (like X000) flips from **0 → 1** or **1 → 0** depending on the wiring logic.

🧠 Nick's Mental Model:

Think of discrete inputs as **the PLC's ears** — they tell it,
"Hey! This button was pressed."
"That valve reached its end."
"A box just passed the sensor."
And the CPU listens and acts based on your program logic.

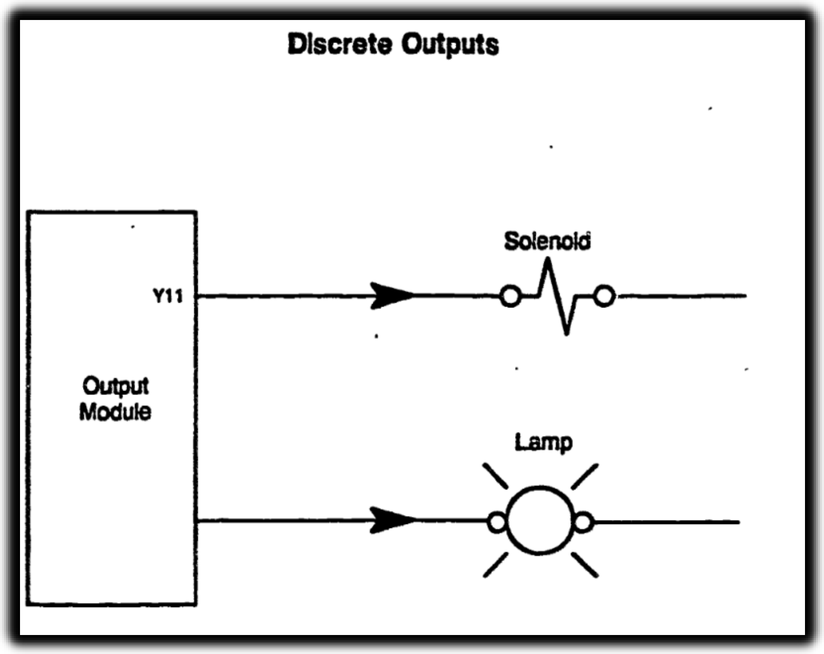
ANALOG INPUT



📌 Discrete Outputs (Digital Outputs)

Just like inputs tell the PLC what's happening outside, **discrete outputs** are how the PLC **tells machines what to do.**

A **discrete output** sends an ON or OFF signal to a field device — nothing fancy, just pure binary power delivery.



⚡ What Are Discrete Outputs Used For?

They control devices that either:

- turn ON/OFF
- open/close
- light up or go dark

🔧 Common Examples:

- **Solenoids** – to open/close fluid or air valves
- **Contactor Coils** – for switching high-power circuits
- **Indicator Lamps / Pilot Lights** – to show machine status
- **Buzzers / Alarms** – to signal errors or events
- **Small Motors or Relays** – in basic control setups

🧰 Wiring & Addressing:

Just like inputs, outputs are also connected to specific terminal points on the **output module**.

Each output gets a **unique address**, e.g.:

Y011 = 12th discrete output (starting from Y000)

These addresses can be set automatically or manually during setup/configuration.

🧠 Nick's Mental Model:

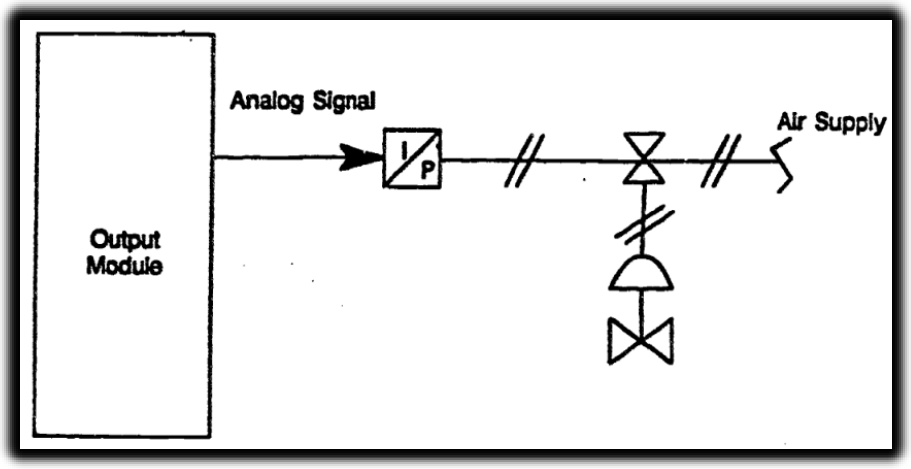
If **discrete inputs are the PLC's ears**,
then **discrete outputs are its hands.**
The CPU listens to what's happening → runs the logic → and then uses outputs to take action.

Example:

If X000 (button press) is ON, then Y011 (lamp) turns ON.

That's classic **input → logic → output** flow.

Analog Outputs



Analog Output refers to an electrical signal sent *from* the PLC *to* an external field device, where the value of the signal is continuously variable, not just ON or OFF. These outputs are typically used to control devices that need **precise, adjustable control**, rather than just binary states.

🌀 What Is an Analog Signal?

An **analog signal** is a continuously variable signal. It can represent a whole range of values — for example, anything from **0 to 10 volts**, or **4 to 20 milliamps** (mA). This is different from digital/discrete signals which are either **on (1)** or **off (0)**.

Think of it like a dimmer switch on a light — not just ON/OFF, but gradually increasing brightness. That's what analog signals do: provide a smooth range of control.

📤 How Analog Outputs Are Used

They're mainly used for **fine control** of physical devices that require variable levels of operation. Here are key examples:

- **Motor speed controllers** – to control the RPM of a motor smoothly
- **Flow control valves** – to let more or less fluid through
- **Temperature regulation systems** – for variable heating or cooling
- **Positioning systems** – like for a robotic arm that needs gradual, smooth movement

🛠 Real-World Example:

Current-to-Pneumatic Transducer (I/P Transducer):

The PLC sends a **4–20 mA** current signal to the transducer. The transducer converts this electrical signal into a proportional **air pressure** that adjusts a **pneumatic valve** in a flow-control system.

- If the PLC outputs **4 mA**, the valve may open only slightly.
- If the PLC outputs **20 mA**, the valve may fully open.
- Anything in between gives **partial openings**, allowing **precise control of flow**.

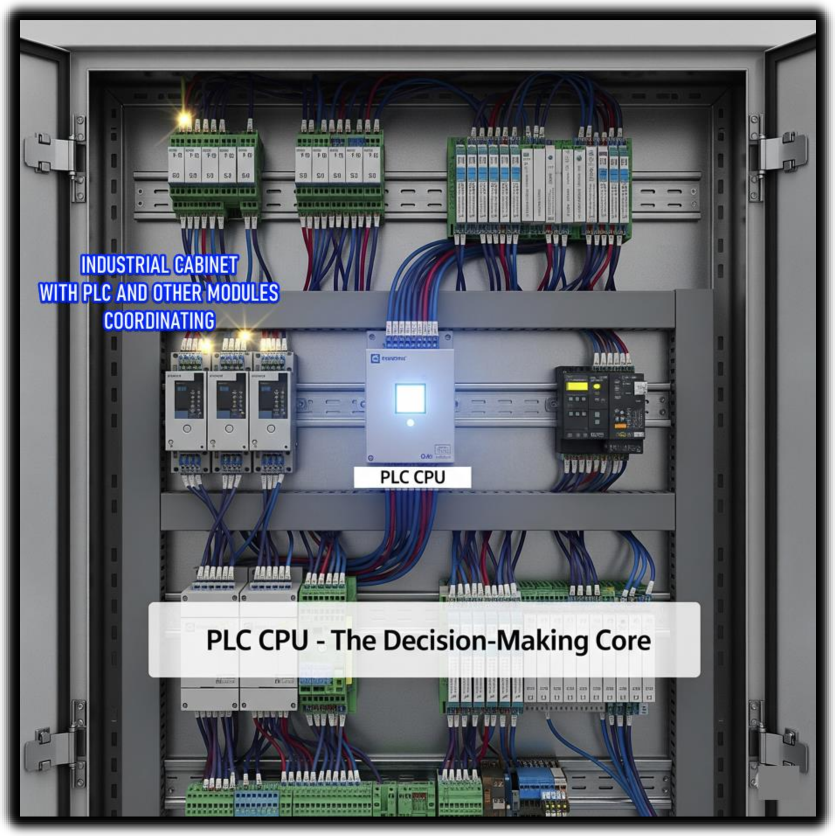
🧠 Extra Details to Keep in Mind:

- The **proportionality** of analog output depends on the whole control loop:
    - ➢ The **primary element** (sensor or input device)
    - ➢ The **transmitter** (converts sensor data to electrical signal)
    - ➢ The **controller (PLC)** and its output scaling
    - ➢ The **actuator** or device receiving the analog signal
- Sometimes, due to mechanical or electrical factors, the system might behave in a **non-linear (disproportionate)** way — meaning that doubling the signal doesn't necessarily double the effect.

📃 Summary (Quick Bites):

- **Analog output = variable signal from PLC (e.g., 4-20mA or 0-10V)**
- Used for **precise control** over devices like valves and motors
- Enables **gradual change**, not just ON/OFF
- Real-life use: controlling a valve in a water plant, adjusting motor speed in a conveyor system, etc.

🧠 Central Processing Unit (CPU) — The PLC's Brain


INDUSTRIAL CABINET
WITH PLC AND OTHER MODULES
COORDINATING
PLC CPU
PLC CPU - The Decision-Making Core

The **CPU** is the heart of the PLC. It's the boss that runs the show. It handles **decision-making** and manages **user memory**, both of which are **fully controlled by the programmer i.e. you 😈).**

The CPU has two main responsibilities:

🧩 1. Decision-Making Section

This is the logic engine. It's where all the **"if this, then that"** type of rules are executed.

✅ What does it do?

- It reads signals coming in from the field (sensors, switches, etc.)
- It **compares those inputs against your programmed instructions**
- Based on that, it decides what outputs to activate or deactivate

⚙️ Example:

"If the water tank is full, then turn off the inlet valve."

This logic is written in your program using conditions. The CPU constantly checks input values (like a float switch in the tank) and executes this rule if the condition is true.

So really, it's like a 24/7 vigilant robot, constantly scanning inputs, and flipping outputs based on your custom logic.

📦 2. User Memory Section

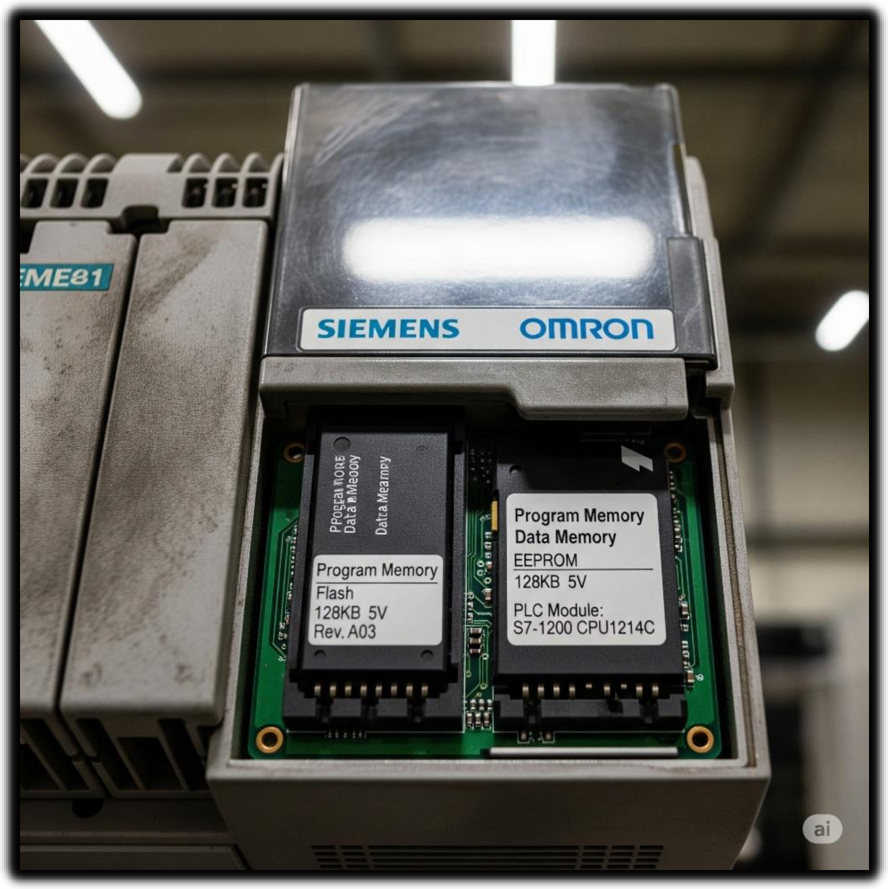
This is where your program and working data live.

🧠 What's stored here?

Instructions (Your actual program — logic, sequences, etc.)

Application-specific data, like:

- Current status of inputs and outputs
- Temporary storage for values (like timers, counters, internal flags)
- Recipes, setpoints, operator-entered data



📝 Real Talk:

If the CPU is the brain, then the **user memory** is like a combo of short-term memory (RAM) and long-term habits (program logic). Your logic tells it what to do, and the CPU keeps checking reality to follow those instructions.

🧪 Summary with Example Flow

1. Input says: “Tank is full” → sensed by a float sensor.
2. CPU reads this input.
3. It checks your program (user memory): IF tank is full THEN close valve.
4. Decision is made: Close the valve.
5. Output is activated to shut off the valve via a relay/solenoid.

That’s how **industrial automation magic** happens. Simple, logical, powerful.