PROGRAMMABLE LOGIC CONTROLLERS

✅ 1. PLC Fundamentals (Core Concepts)

- **What is a PLC?**
  - Definition & role in automation (why PLCs?) ✅
  - Advantages over hard-wired relay logic. ✅

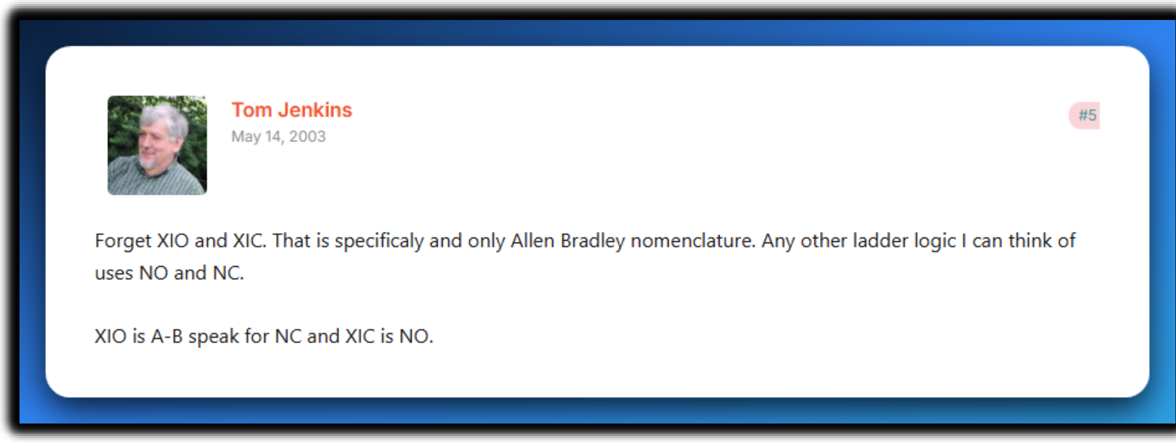
- **PLC Architecture**
  - CPU (control + scan cycle) ✅
  - Memory (program memory, data table, I/O image table) ✅
  - Input modules (digital/analog) ✅
  - Output modules (digital/analog) ✅
  - Power supply ✅

- **PLC Scan Cycle**
  - Input scan → Program execution → Output update ✅
  - Relation to continuous loops in software ✅

- **PLC Programming Languages (overview)**
  - Ladder Diagram (main focus) ✅
  - Mention of others: FBD, ST, IL, SFC (just awareness) ✅

✅ 2. Ladder Logic Essentials (Main Exam Meat)

- ### Contacts and Coils
  - Normally Open (XIC) 🚨
  - Normally Closed (XIO) 🚨 Forget XIO/XIC those are for Allen Bradley nomenclature. Any other ladder logic I know uses simple NO and NC.
  - Output Coil (OTE) ❌
  - Latch (OTL) and Unlatch (OTU) ❌



Tom Jenkins

May 14, 2003

#5

Forget XIO and XIC. That is specificaly and only Allen Bradley nomenclature. Any other ladder logic I can think of uses NO and NC.

XIO is A-B speak for NC and XIC is NO.

Source: plctalk.net

- ### Logical Operations
  - AND (series contacts) 🚨
  - OR (parallel contacts) 🚨
  - NOT (using NC contacts) ❌

- ### Internal Memory Bits
  - How to use internal relays (B3, M bits) for intermediate logic ❌

- ### Basic Programs
  - Start/Stop motor latch circuit ❌
  - Simple interlock (two conditions must be true) ❌

✅ 3. Timers (Time-Based Control)

- On-Delay Timer (TON) ❌
- Off-Delay Timer (TOF) ❌

- Retentive Timer (RTO) ❌
- Timer Reset (RES) ❌
  (Know their use-cases and how to wire them in ladder logic)

✅ 4. Counters (Count-Based Control)

- Count Up (CTU) ❌
- Count Down (CTD) ❌
- Counter Reset (RES) ❌
  (Be able to create simple counting circuits)

✅ 5. Data Handling & Math

- **Comparison Instructions**
  - Greater Than (GRT) ❌
  - Less Than (LES) ❌
  - Equal (EQU) ❌
- **Math Instructions**
  - ADD, SUB, MUL, DIV ❌
  - Scaling analog inputs (basic understanding) ❌
- **Move Instruction (MOV)**
  - Copy data between memory locations ❌

✅ 6. Program Structuring Techniques

- Sequencing (step-by-step control, e.g. washing machine cycle) ❌
- Subroutines / JSR (jump to subroutine) ❌
- State machines (using memory bits to track steps) ❌

✅ 7. Troubleshooting & Simulation

- ### Diagnostics
  - Going online with PLC software ❌
  - Forcing I/O (know concept) ❌
  - Monitoring and interpreting real-time values ❌
- ### Common Faults-
  - Inputs not wired or misaddressed ❌
  - Timers/counters not resetting as expected ❌
- ### Simulation Practice *(you already reached this in class)*
  - Start/Stop motor latch ❌
  - Traffic light (timers) ❌
  - Bottle filling/counting (counters) ❌
  - Door interlock (AND/OR logic) ❌

✅ 8. Extra (If Time Allows / Bonus Points)

- Analog I/O handling (4–20 mA sensors, scaling raw values) ❌
- Safety interlocks (why certain conditions must be enforced before outputs energize) ❌ (we didn't learn this part 8 in class only part 7 was the last part.

✨ Your Study Priority

If time is tight, focus in this order:
Ladder Logic Basics → Timers → Counters → Comparisons/Math → Structuring → Troubleshooting/Simulation.

💡 Practical Tip

✅ Spend most of your time in a simulator (OpenPLC or LogixPro).
✅ Build small circuits for each topic.
✅ Write tiny notes after each topic with a quick ladder snippet.

✅ 9. Minor Topic Additions for a Full House 🏠

Data Addressing & Naming Conventions: Understand how different PLC brands label inputs, outputs, timers, and internal bits (e.g., I:1/0, O:2/1, M0.0), so you can read and write ladder logic accurately.

Basic Error Handling & Safety Chains: Learn how to use latches/unlatches or fault bits to flag abnormal conditions, and design safety interlocks (like E-stops) that override all other logic when triggered.

PLC Operating Modes: Know the difference between RUN, PROG/STOP, and REMOTE modes, and how each affects program execution and editing.

Troubleshooting Scenarios: Be ready to diagnose practical issues (e.g., motor not starting despite input signal, counters double-counting) by tracing logic, wiring, and I/O behavior.

✨ WHAT IS A PLC?

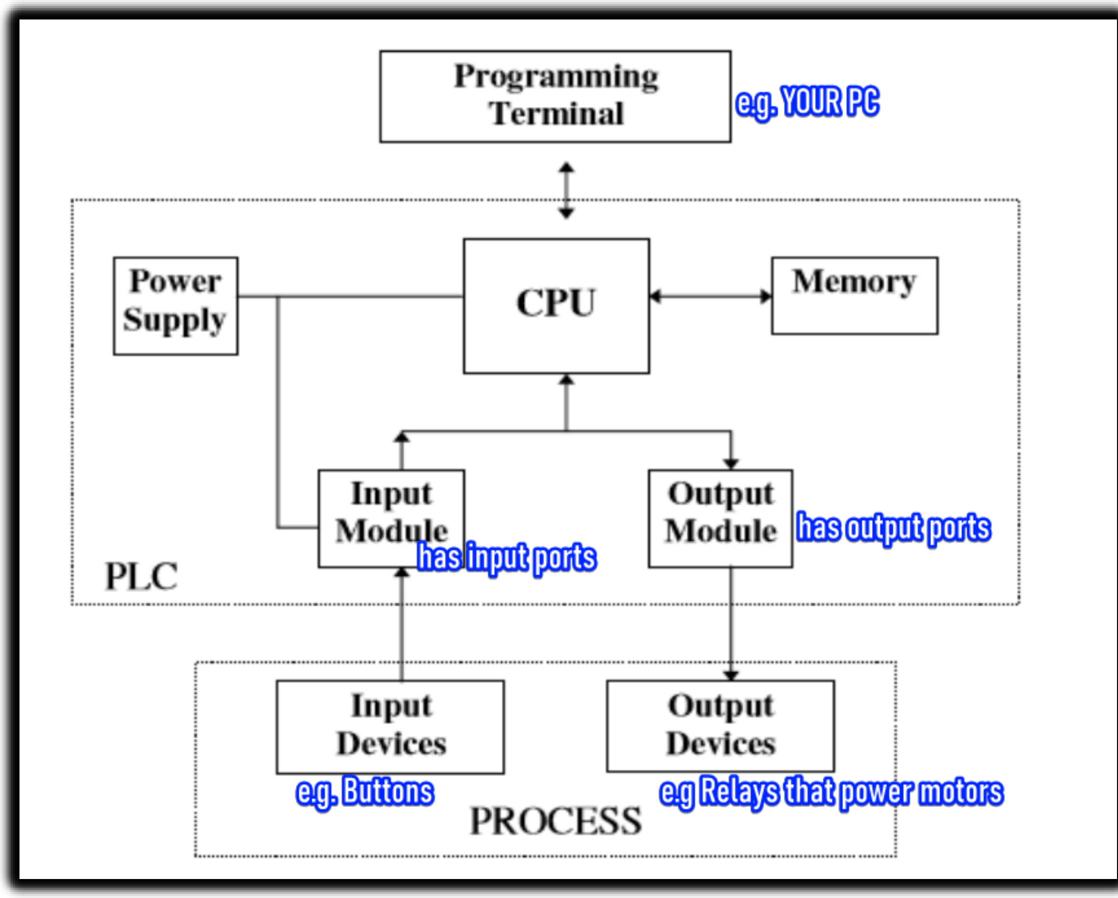
PLCs are a specialized industrial computer that controls and automates processes in various industries.

PLCs are designed to be rugged, reliable, and easily programmed to **monitor** and **control** machines and processes.

A **Programmable Logic Controller (PLC)** is basically a rugged, specialized industrial computer built to run automation tasks. Rugged means they can operate in harsh environments.

👉 It *takes in signals* from sensors and switches, processes them according to a stored program, and then sends commands to outputs like motors, valves, or lights.

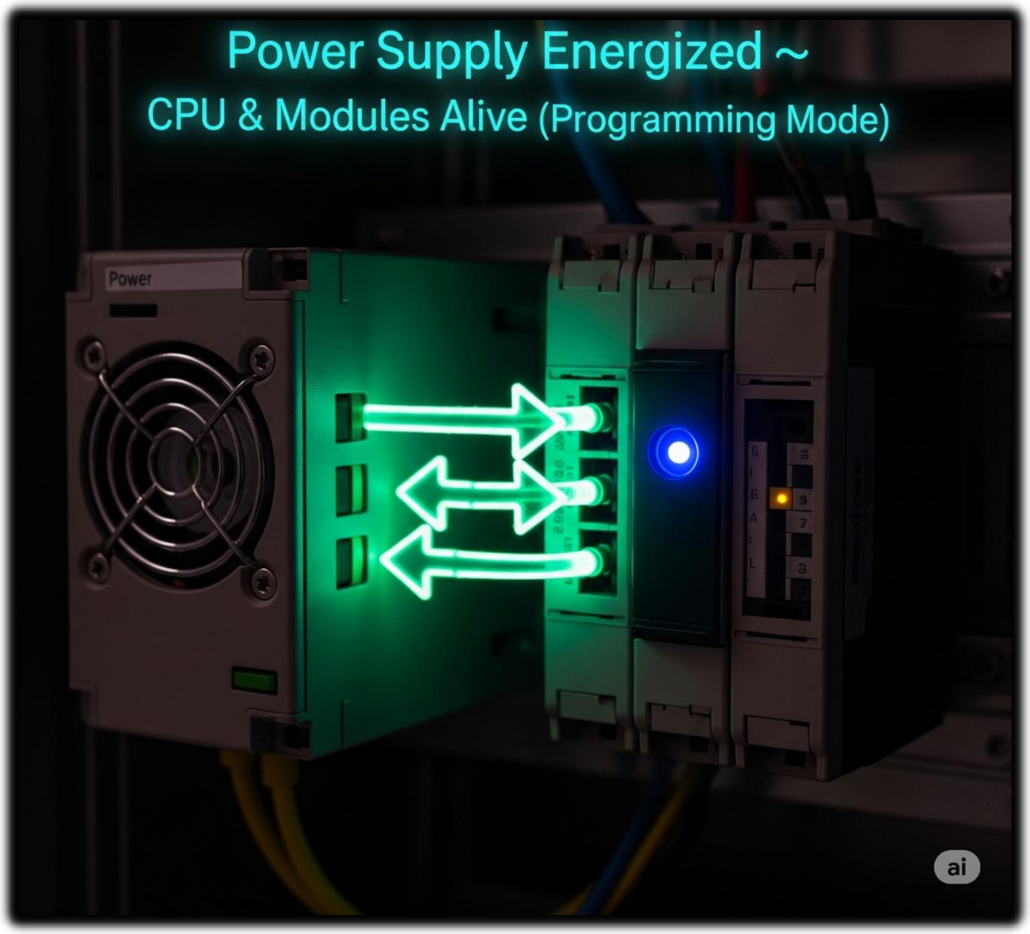
PLC Operation in real life/ PLC architecture



Process 1: Initializing the PLC System

Upon arriving at the company, the first step involves powering up the PLC system. This means energizing the **Power Supply** unit, which then provides stable DC power to the PLC's internal components, including the CPU and I/O modules, bringing the entire controller to life.

At this point, the PLC's CPU typically enters a **Programming Mode** (or Stop Mode), where it is powered on but not actively executing any control logic, waiting to receive instructions.



Process: Programming and Downloading Logic

Next, using a **Programming Terminal** (your PC or specialized device), you develop your control logic. This logic can be written in various IEC 61131-3 standard languages, such as Ladder Logic, Function Block Diagram, Structured Text, or Instruction List.

Once the program is complete and debugged in the software, it is then compiled and downloaded from your Programming Terminal to the PLC's **Memory**.

During this download, the PLC's CPU accepts and saves the new program, essentially acquiring its "new skill set" or updated operational instructions.

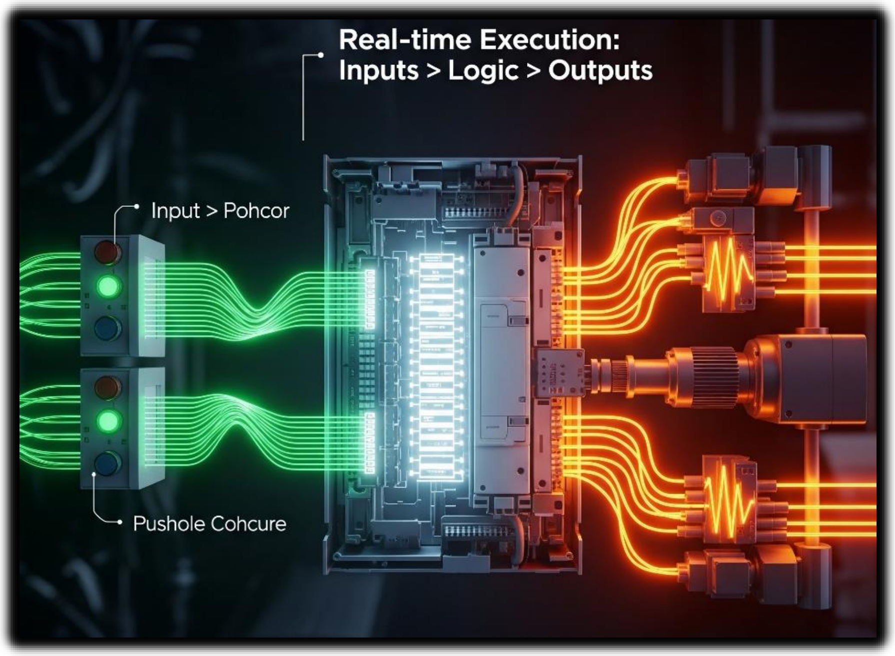
Process: Transition to Run Mode and Program Execution

After a successful program download, the PLC's CPU is switched from Programming Mode to **Run Mode**. In this mode, the PLC begins its continuous, high-speed **scan cycle**.

This cycle involves three primary phases: first, the CPU reads the current states of all **Input Devices** via the **Input Module**, updating its internal memory image of the inputs.

Second, it executes the downloaded program rung-by-rung (or instruction-by-instruction), processing the logic based on the current input states and internal memory bits.

Finally, it updates the states of the **Output Devices** based on the results of the executed program.



Process: Real-World Actuation via Output Chain

When the PLC's program logic dictates that a specific output should be activated (e.g., a motor needs to start), the CPU sends a low-power digital signal to the corresponding point on the **Output Module**.

This Output Module then converts that low-power digital signal into a suitable electrical signal (e.g., 24V DC). This low-power signal from the Output Module is then typically routed to the coil of an **external control relay or motor contactor**.

This external relay, acting as a robust intermediary, receives the small signal from the PLC, and in response, its heavy-duty contacts close, allowing high-power electricity to flow to and energize the **Output Device** (e.g., the motor), thereby initiating the desired machine action.

Key real-world use cases:

- Controlling machines on a factory assembly line.
- Managing amusement ride rollercoasters as they twist and turn.
- Automating food-processing machinery that mix ingredients for your favorite snack.

Why PLCs instead of normal PCs?

Designed to Handle Digital & Analog I/O:

PLCs have built-in I/O modules to directly handle digital and analog signals from industrial devices, while regular PCs need extra hardware and software, making them slower and harder to use for real-time control.

Survive Extreme Temperatures:

PLCs are rugged and built to survive extreme temperatures, while regular PCs can't handle the heat—or the cold—of harsh industrial environments.

Immune to Electrical Noise:

PLCs are built to resist electrical noise from heavy machinery, while regular PCs can glitch or fail in such interference-heavy environments.

Resist Vibration and Impact:

PLCs are designed with sturdy, shock-resistant casings and components, enabling them to withstand vibrations, impacts, and rough handling while regular PCs are too fragile and prone to hardware failures under physical stress.

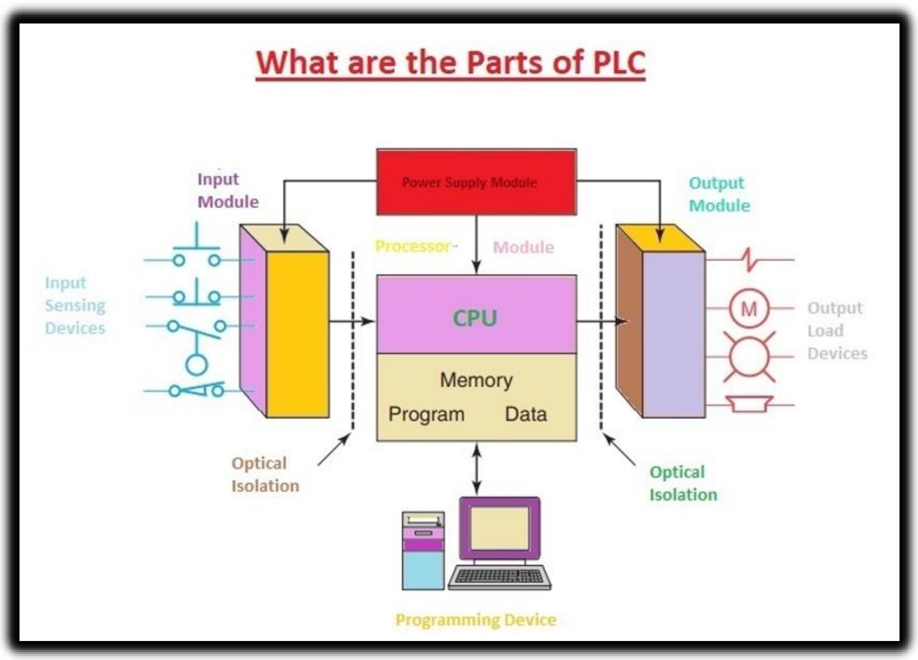
🧠 Core Sections of a PLC

CPU (Central Processing Unit)

The PLC's "brain."

Contains a microprocessor, memory chips, and circuits for control logic, monitoring, and communication.

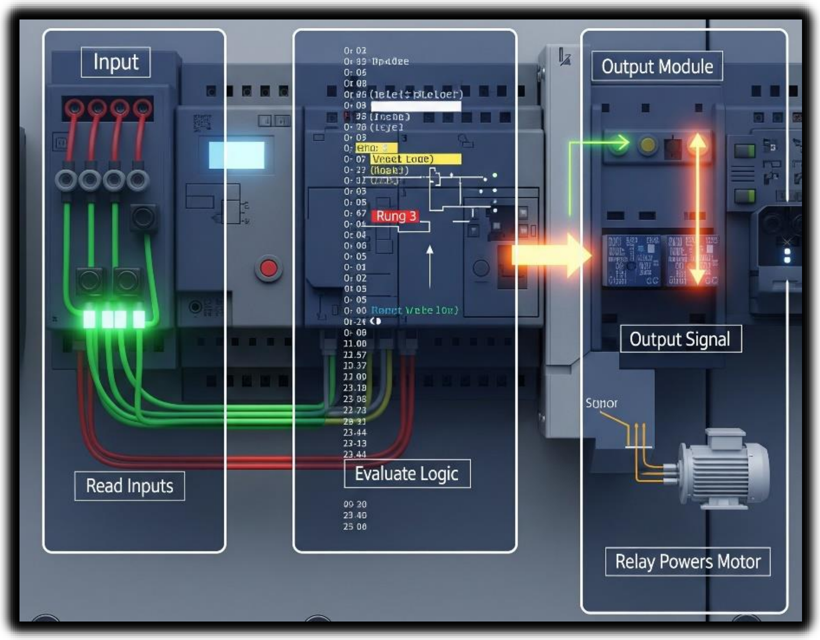
- **Microprocessor**: The brain of the PLC, executing tasks and calculations quickly.
- **Memory Chips**: Store the program and real-time data, like sensor states.
- **Control Logic & Communication Circuits**: Internal pathways for CPU communication with memory, I/O, and other devices.



⚡ CPU Operating Modes

Just like you might switch between "work mode" and "chill mode," the PLC's CPU has distinct operating modes:

- **Programming Mode:** In this mode, the PLCs CPU is ready to receive **new instructions**. When you connect a PC with a PLC programming software, the CPU is in this mode, **ready to accept and save those changes** to its memory. It's like updating the PLC's brain with new skills.
- **Run Mode:** Once the program is downloaded and confirmed, you switch the CPU to Run Mode, where the PLC **springs into action**! In this mode, the CPU constantly executes the program stored in its memory.

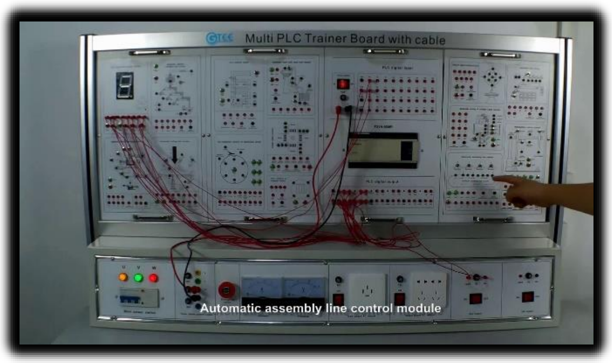


⚡I/O Interface System

PLC's "nerves": These are the input/output (I/O) systems that receive signals from sensors/switches and send commands to actuators (motors, lamps, relays).

Inputs (The Senses – How the PLC Gathers Info): 👂👀

- This part of the PLC is constantly **listening and watching** for signals from the outside world.
- **Field devices** are like the "reporters" sending info *to* and receiving instructions *from* the **PLC**. They gather real-world info (like temperature, position, pressure) or act on commands (like turning a motor on).
- **Sensors:** Sensors give the PLC *status updates*. A proximity sensor says, "Product is in place." A temperature sensor warns, "It's getting hot!" 🌡️🔥
- **Switches:** That button which sends an electrical signal (an input) to the PLC, saying, "Let's get started/stop!", controlled by humans.



PLC Training Board.

💾 PLC CPU Memory – What it stores and tracks

- Stores the **program logic**, the actual instructions.
- Tracks the current **status of inputs and outputs**
- Data Values – Stores values for:
  - ➢ Timers ⏱️ (e.g., wait 5 seconds before starting)
  - ➢ Counters 🔢 (e.g., count 10 items passing a sensor)
  - ➢ Internal Bits ⚙️ (virtual switches used inside the program)

Outputs (The Muscles – How the PLC Takes Action): 🦾🗣️

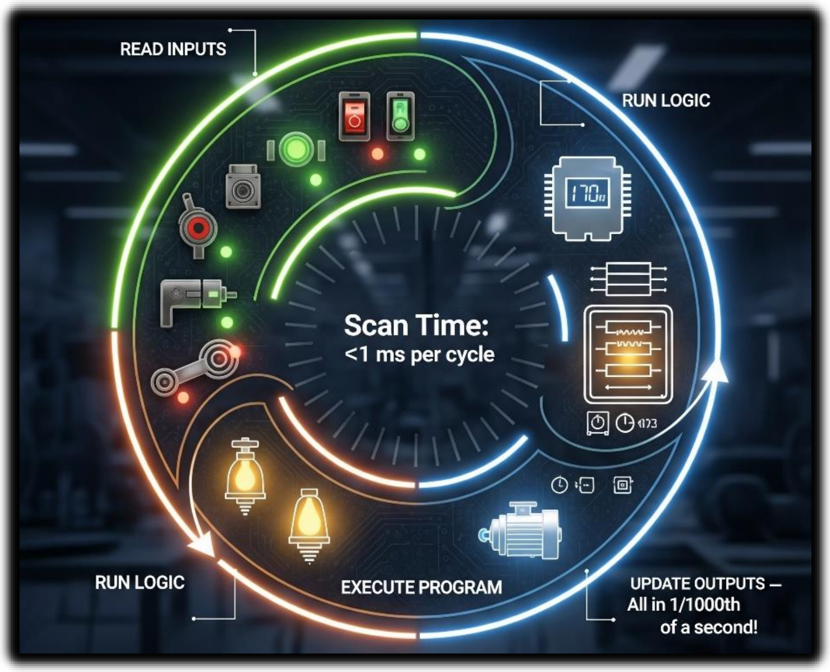
- Once the PLC's brain (CPU) has processed the inputs and made a decision, the output interface is how it **sends commands** back out to the real world to make things happen.
- **Actuators** are the "doers" that receive commands from the PLC and perform a physical action.
- **Motors:** If the PLC decides to start a conveyor belt, it sends an electrical signal (an output) to the motor, telling it to spin.
- **Lamps/Lights:** If a machine completes a task, the PLC might turn on a *"Task Complete"* light. Or, if there's an error, it might flash a warning lamp. 🚨
- **Relays:** PLCs can't directly power **big machines** — they're not strong enough. So they use a **relay** (like a remote-controlled switch) to turn on bigger devices.
  🗣️ PLC says: *"Relay, switch that heavy-duty pump on!"*
  ➡ Relay flips the power to the big machine.

The Scan Time: Blazing Fast Automation! 🚀

A PLC is a dedicated industrial controller, built to run a single control program — and it does so continuously and extremely fast.

1. 🔍 **Read Inputs**
   The PLC checks the status of ***all input devices*** (e.g., are switches pressed? Is the sensor triggered?).
2. 🧠 **Execute Program**
   Using the input data, the CPU **runs the logic** stored in its memory — this includes timers, counters, and all the if/then conditions in the control program.
3. ⚡ **Update Outputs**
   Based on the results, the PLC **sends commands to output devices** (e.g., turn on a motor, light a lamp, or open a valve).

The time it takes for the CPU to complete **one full cycle** – reading inputs, executing the program, and updating outputs – is called the **scan time**.



This happens with mind-blowing speed, often in the range of **1/1000th of a second** (that's 1 millisecond!).

Imagine blinking, and the PLC has already completed several hundred scans!

This rapid cycling is crucial for ensuring that industrial processes respond instantly to changes in the real world.

⚙️ PLC Scan Cycle Recap:

Every PLC constantly runs in a **loop** like this:

```
1. Input Scan        → read sensor/input states
2. Logic Execution   → run the ladder program using those inputs
3. Output Update     → update real-world outputs based on logic
```

Then back to step 1. Over and over. Super-fast. Like 1,000 times per second.

🤖 "Relation to Continuous Loops in Software" — What does that mean?

That sentence is drawing a comparison between **how PLCs run** and how **normal programs** in, say, C or Python, run loops like this:

```
while (1) {
    read_inputs();
    run_logic();
    update_outputs();
}
```

The **PLC scan cycle** is basically a **hidden while-loop** that never stops.
It's as if the PLC firmware is running something like:

```
while (true) {
    scan_inputs();
    execute_ladder_logic();
    update_outputs();
}
```

Except it's optimized, real-time, and done in **hardware** + **firmware**.

🤯 Why is this even important?

Because it explains:

- Why your outputs **don't update instantly** when an input changes. They only change on the **next scan**.
- Why timing issues happen if your ladder logic takes too long to execute.
- Why some instructions (like SET/RESET) persist across scans while others (like OUT) only fire if the rung is *true every scan*.

It's literally how the PLC *lives*. It's not event-driven like Windows code; it's **scan-loop based like embedded systems.**

🎮 Final Analogy:

Think of a video game:

- Every frame, it:
  - *Checks input (keyboard/mouse)*
  - *Runs game logic*
  - *Draws the frame (output)*

That's exactly what the PLC is doing. But instead of FPS, it's scan time.

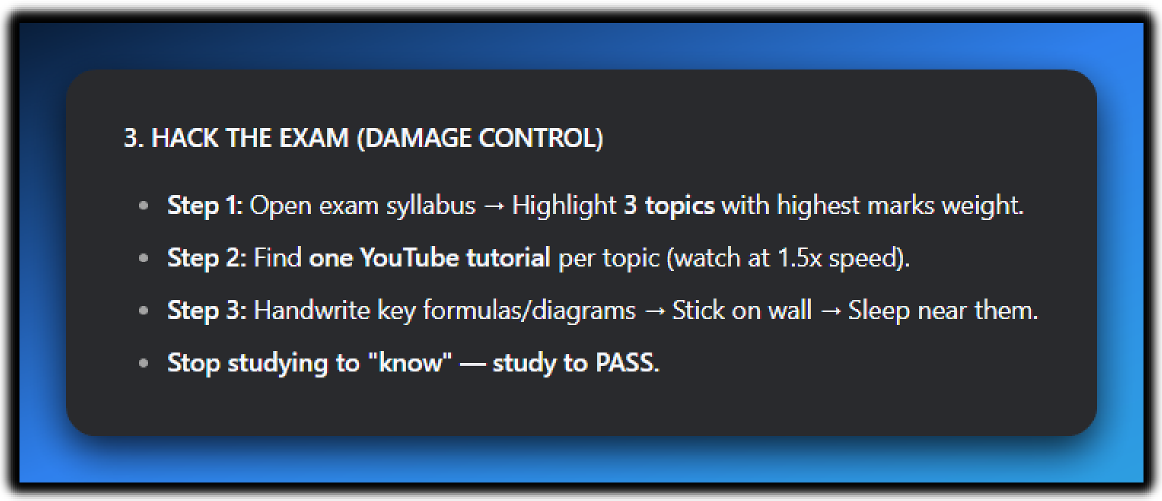
⚙️ Why is This Important?

This ultra-fast cycling is **critical** in automation:

- Ensures machines respond **instantly** to changes.
- Keeps operations **precise, reliable, and safe.**

🔥 Why this matters for your exam

- You can now clearly define what a PLC is, where it's used, and why it's built tough.
- You can explain its main parts (CPU and I/O), modes (program/run), and how the scan cycle works.
- You understand what "scan time" means and why memory is crucial.



✅ Main parts of a PLC (Programmable Logic Controller)

1. **CPU (Central Processing Unit)**

   👉 *The brain.*

   - Executes the control program (ladder logic, etc.)
   - Does all the decision-making.
   - Handles communication with other modules.
   - Stores the logic in memory and updates outputs based on inputs.

2. **Power Supply**

   👉 *The heart pumping electricity.*

   - Feeds the CPU and I/O modules with stable DC power (often 24volt DC).
   - Without this? Dead PLC. 💀

3. **Input Module(s)**

   👉 *The senses (eyes, ears).*

   - Reads signals from field devices (sensors, pushbuttons, limit switches).
   - Converts them into logic levels the CPU understands.

4. **Output Module(s)**

   👉 *The muscles (hands, feet).*

   - Sends commands to actuators (motors, relays, lamps, solenoids).
   - Converts CPU logic to real-world signals.

5. **Programming Device (not always mounted, but essential)**

   👉 *The interface.*

   - Laptop or handheld used to load/edit programs into the PLC.
   - Without this, you can't tell the PLC what to do.

6. **Communication Ports / Interfaces**

👉 *The mouth and ears for networking.*

- Ethernet, RS-232, Profibus, etc. for talking to HMIs, SCADA, or other PLCs.

In exam-mode language:

A PLC is made up of a **CPU** (processes the program), a **power supply** (provides DC power), **input modules** (receive signals from field devices), **output modules** (drive actuators), and **programming/communication interfaces** (for configuration and data exchange).

🔥 **Pro tip:** If they want extra sauce, throw in: *"All these parts work together to replace hard-wired relay logic with flexible, programmable control."* 🎯