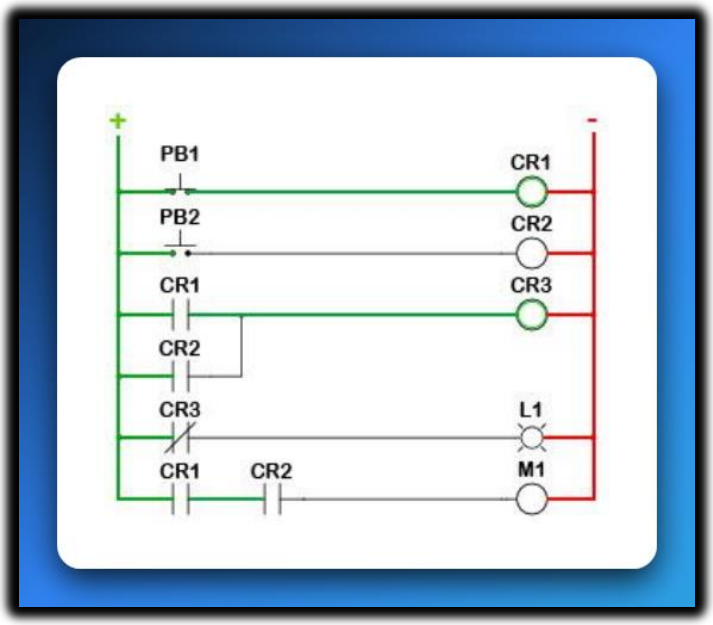
LADDER LOGIC

The Foundational Structure: From Physical Relays to Digital Logic

At its heart, **Ladder Logic** is the main language used to program **PLCs**—those industrial computers that keep machines running like clockwork.

But to really understand it, you've gotta know where it came from: the old days of **relay logic** and **hard-wired control panels**.

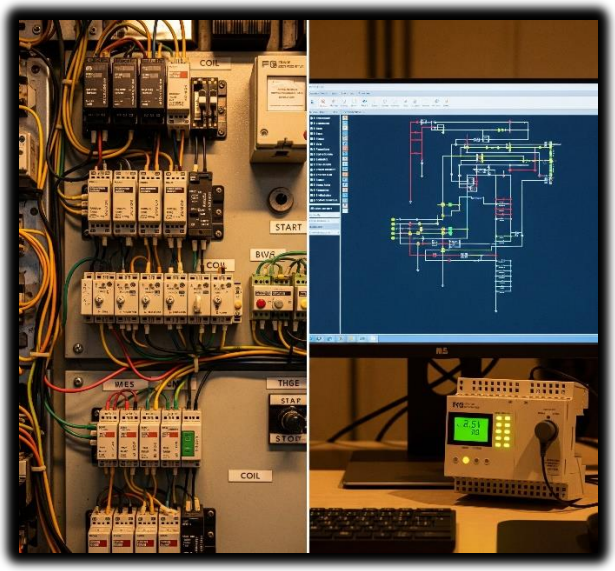


⚡ 1. The Roots: Relay Logic & Hardware Automation

Before microprocessors were everywhere, factories used **physical relays** to automate machines.

A relay is basically an electrically operated switch:

✅ Send a small current through its coil → it creates a magnetic field → that pulls or releases a metal arm → opening or closing contacts to let power through.



💥 Why It's Called a "Ladder"

It's not just a cute name. Ladder Logic diagrams **look exactly like old relay wiring diagrams**:

Rails (Vertical Lines):

- The **left rail** is the power source (e.g., +24 V DC or 120 V AC).
- The **right rail** is the return path (common/ground).
  Power always flows left to right.

Rungs (Horizontal Lines):

- Each rung is like one mini-circuit or one logic rule.
- On each rung, you place components—switches, contacts, coils—in **series** or **parallel**.
- If the path on that rung is complete, the output on the far right energizes.

⚙️ How It Worked (and Still Does in Principle)

In an old relay cabinet:

- Power starts at the **left rail**, passes through inputs like pushbuttons or sensors, then through relay contacts, and finally reaches an output device like a motor, light, or another relay coil.
- **If every condition on that rung is true (closed contacts, active sensors)** → the circuit completes → the output device gets power and turns on.

Modern Ladder Logic in a PLC mimics this exact same idea—*but instead of physical wires and relays, it's all done in software.*

✨ 2. The Evolution: From Tangled Wires to Clean Software Bits

The genius of **Ladder Logic** is how it took the messy world of physical relays and turned it into a neat software language.

Instead of grabbing a screwdriver and wiring real relays together, you **"draw" your logic on-screen** with symbols that look just like the old electrical components.

When the PLC runs your program, it's basically *pretending* electricity is flowing through those virtual circuits.

💻 Digital Translation – How It Maps Over

Power Flow:

In software, there's no real current. Instead, a logical **TRUE (1)** means "power is flowing" along that rung. If the path from the left rail to an output is logically true, that output gets activated.

Open vs. Closed Contacts:

Physical relay contacts become simple Boolean conditions in the PLC:

- **Normally Open (NO)** – shown as —| |—
  ✅ True when the input is ON (button pressed, sensor triggered, relay energized).
- **Normally Closed (NC)** – shown as —|/|—
  ✅ True when the input is OFF (button released, sensor not detecting, relay not energized).

Coils / Outputs:

Outputs are shown as —( )—.

🚨 When the rung logic leading to that coil is TRUE, the PLC sets that output bit to **1**, which energizes the real-world device (motor spins, light turns on, valve opens) or even an internal memory relay.

👉 Big Picture:

Instead of digging through wires in a control cabinet, you're now dragging and dropping logic in software.

Same principles, zero mess. That's the magic of Ladder Logic.

🏗️ Key Ladder Logic Components & Their Analogies

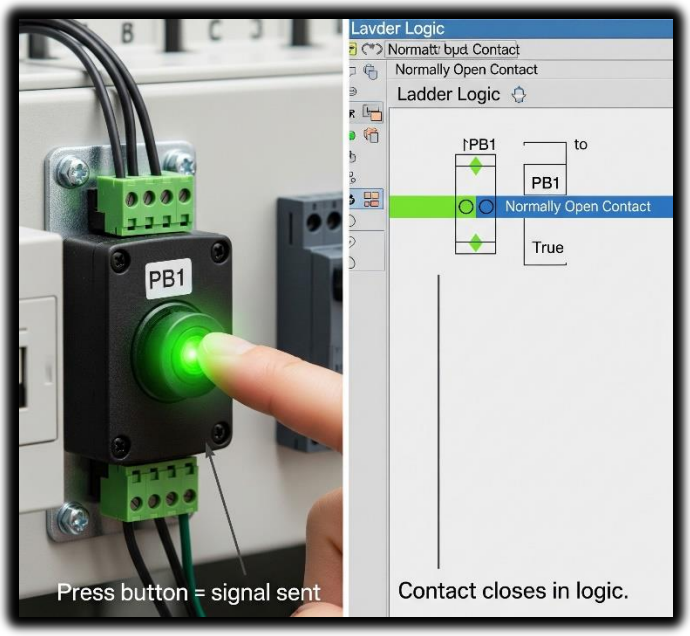
Let's break down those classic symbols from your image and see how they vibe both in hardware *and* in software logic:

🔘 PB (Pushbutton) – Your Event Trigger

PB1, PB2: These are **inputs**.
On a real panel, pressing a pushbutton closes a circuit and sends a signal into a PLC input terminal.
In Ladder Logic, it's represented by a contact symbol.

👉 Analogy:

A pushbutton is like calling a function or triggering an interrupt in code.
When you hit it, you're saying: *"Yo, start that sequence!"*

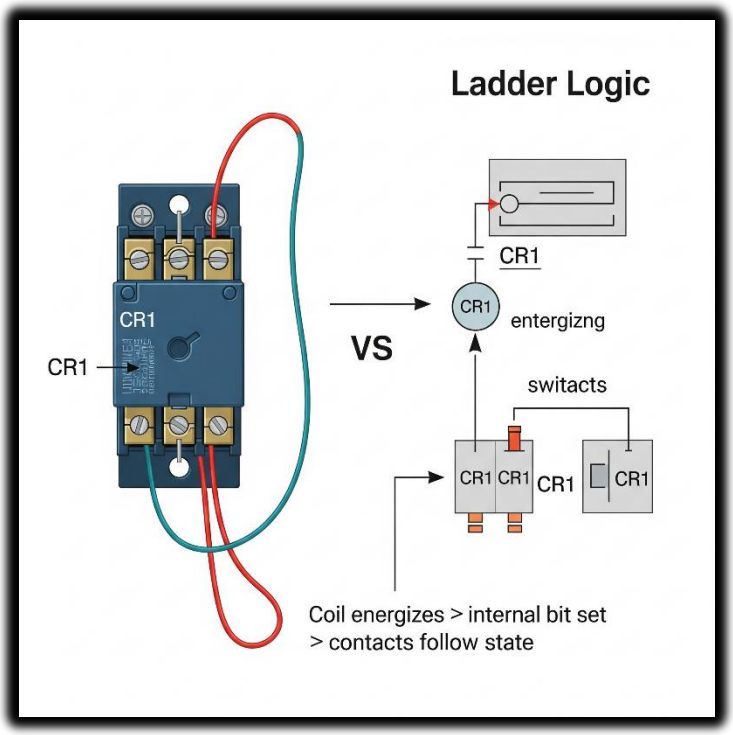


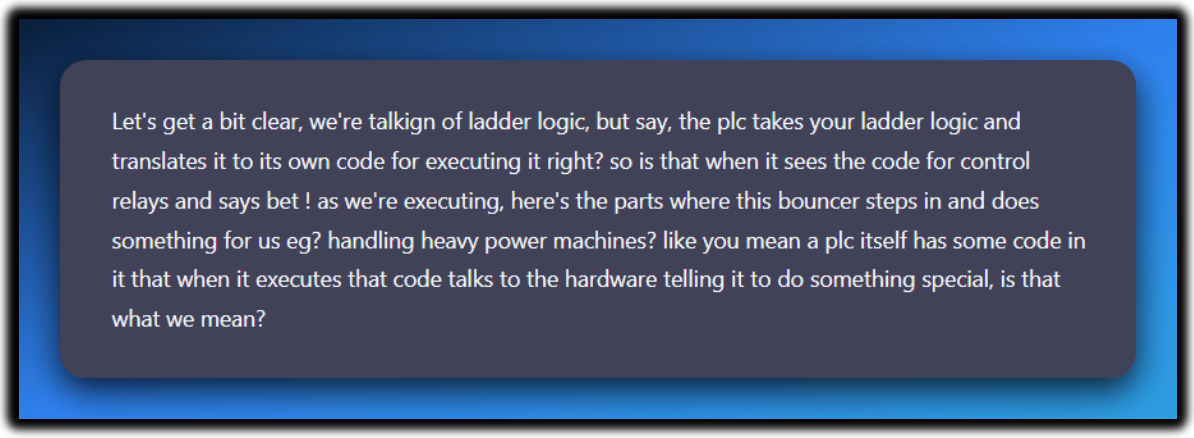
⚡ CR (Control Relay) – Your Internal State & Your Electrical Middle-Man

CR1, CR2, CR3:
Inside Ladder Logic, we treat these as **internal memory bits** — energize the coil (CR1) and every CR1 contact in your logic instantly follows that state.

But in the real hardware world?
A control relay also acts as an **electrically controlled switch**.
It lets the PLC — which only pushes tiny, low-power signals — safely control **higher-power devices** like motors, solenoids, or large lamps.
It's the buffer between the fragile logic electronics and the beefy machinery, giving you:
✅ **Electrical isolation** (protects your PLC from power spikes),
✅ **Safety** (no direct heavy current through your delicate PLC board),
✅ **Stability** (reliable switching for big loads).

👉 **Analogy:**
Think of it as a bouncer at a club.
Your PLC quietly whispers, "let them in," and the control relay (the bouncer) swings open the big heavy door to the motor or solenoid.
The PLC never has to wrestle with high current directly — the relay does the heavy lifting.





Inside your ladder, when you drop in a coil (CR1) or an output (M1) you're basically saying to the PLC:

"Hey PLC, if the logic on this rung is true, energize this output (or set this internal bit)."

What happens next depends on what type of coil it is:

💥 Internal CR (Control Relay):

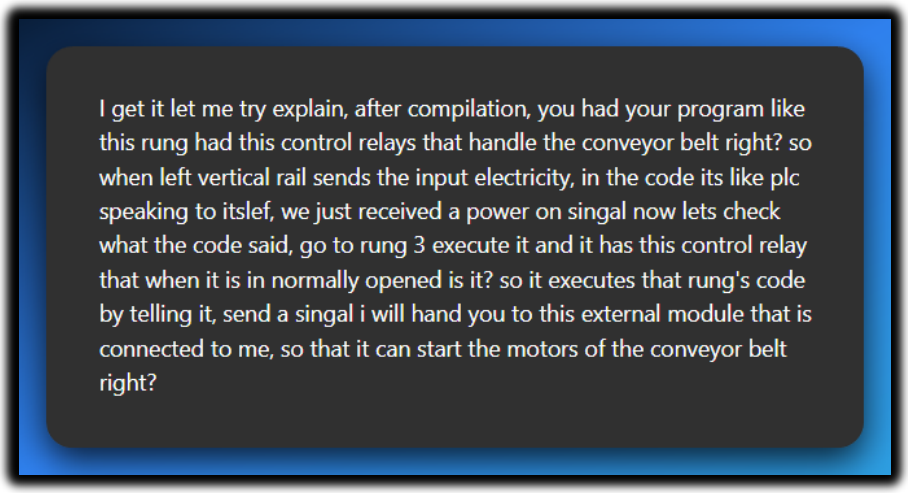
- This one lives purely in PLC memory.
- When the CPU executes that rung and it's true, it sets a *bit* in memory to 1.
- Any rung reading that contact (—| |— CR1) will now see it as "closed."
  👉 No actual electricity is switching heavy loads yet — it's just logic inside the PLC.

💥 Physical Output Coil (like M1 for a motor):

- When the CPU executes that rung and it's true, it flips a transistor or energizes a tiny driver circuit on the PLC's output card.
- That tiny driver is *not* strong enough to power a motor directly — and that's where real **control relays** or **contactors** out in the panel come in.
- The PLC output energizes the relay coil (low current).
- The relay's contacts then safely switch the **big current** going to your motor or solenoid.

The PLC scan loop runs through your compiled logic thousands of times a second. Each time it hits an output coil instruction, the CPU updates the hardware output register.

That register is connected to output modules — which *physically* drive relay coils, solid-state switches, or transistors on your control panel. The PLC isn't literally muscling 10 amps into a motor — it's flipping a small digital signal that *commands* the relay/contactors to handle the big current.



After your ladder logic is compiled, the PLC is constantly scanning. Imagine we're mid-scan:

⚡ Input side:

The PLC reads the status of all input terminals (pushbuttons, sensors, etc.).
👉 The left rail is like "power" coming in, but in PLC land it's really just reading TRUE/FALSE from each input.

🧠 Logic side:

Now the PLC runs through your ladder rungs, top to bottom: *"Ok, rung 3... oh, this rung has a coil that controls the conveyor motor. Let's check the contacts in this rung."*

It evaluates your contacts (normally open/closed) based on the input states and internal bits. If the logic path is TRUE (like that normally open contact is now closed because the button was pressed): *"Alright, condition is true — energize that output coil!"*

📤 Output side:

The PLC doesn't directly blast **3-phase power(typically provides higher electrical power)**. Instead, it sends a low-power signal out of an output pin on the output module:

"Hey output module, set your transistor/relay ON for Conveyor_Motor."

🔌 External world:

The output module energizes a control relay or contactor coil out in the panel.
That relay/contact closes heavy-duty contacts that feed the actual conveyor motor's power circuit. 👉 And boom — your conveyor motor spins up. 🚀

🤖 So your sentence becomes:

"When the PLC is scanning and gets to rung 3, it evaluates that rung's logic (including your control relay contact). If the logic is TRUE, the PLC sends a signal from its output module to energize an external relay or contactor, which then powers the conveyor motor."

Your idea is absolutely right.

✅ Left rail = inputs read.
✅ Code rung-by-rung.
✅ If logic says "go," PLC outputs a low-power signal.
✅ Output module reads the signal → control relay/contact closes → heavy machine (like the conveyor starts moving).