LADDER LOGIC 2

🤖 Chapter 5-2: Basic Instructions in Ladder Logic

Boolean Logic and the DNA of Control

Now that you know what Ladder Logic looks like, it's time to understand **how it actually decides things**. And that takes us straight into **Boolean logic** — the bedrock of all decision-making in PLCs.

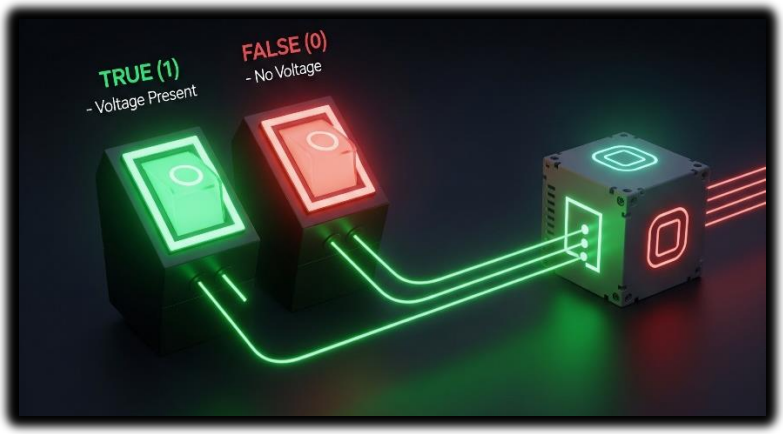
Don't worry, this isn't Digital Systems 101 — we're not breaking out Karnaugh maps or Boolean algebra proofs. We're just gonna look at the essentials: **AND** and **OR** logic.

🔲 Boolean Logic in Plain Language

PLC logic is built on simple True/False decisions — like light switches:

- **TRUE (1)** → There's voltage or a condition is satisfied
- **FALSE (0)** → No voltage or the condition isn't satisfied

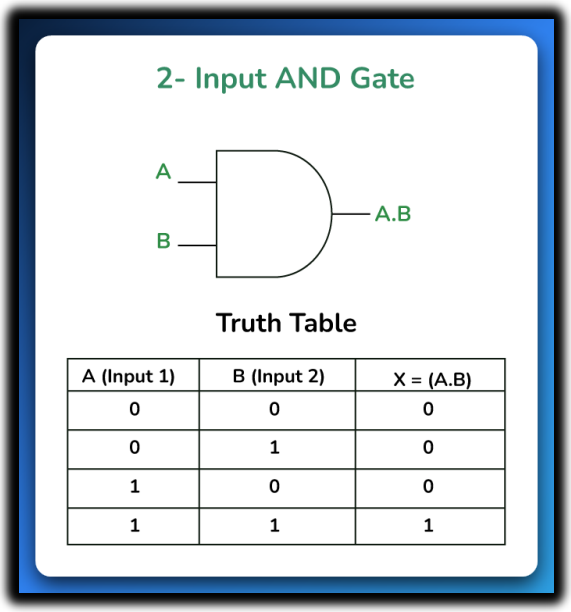
From this binary setup, we build logic gates. In Ladder Logic, these gates are **not separate components** — they're created through **how you arrange your rungs and contacts**.



The Gates — Series Logic

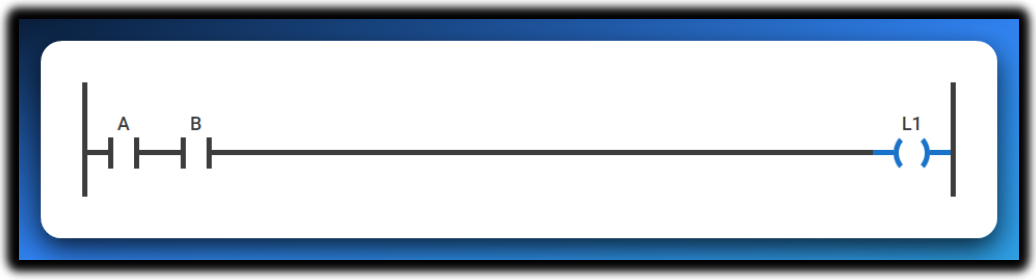
🔌 AND gate Ladder Implementation:

AND operations are analogous to **multiplication.**



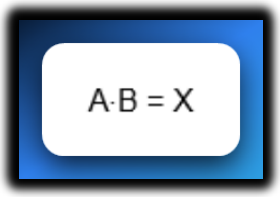
Use **normally open contacts in series**.

This is saying:



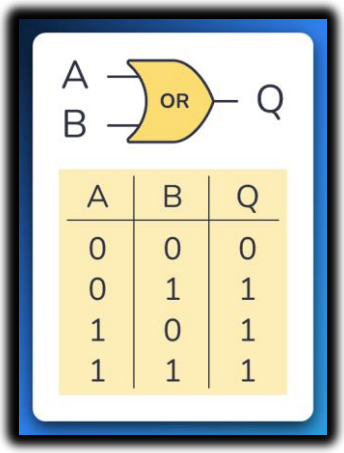
*"Only if A **AND** B are both TRUE, then turn ON the output."*

If either A or B is FALSE, current stops flowing — just like if one switch in a series circuit is off.



🔌 OR gate Ladder Implementation:

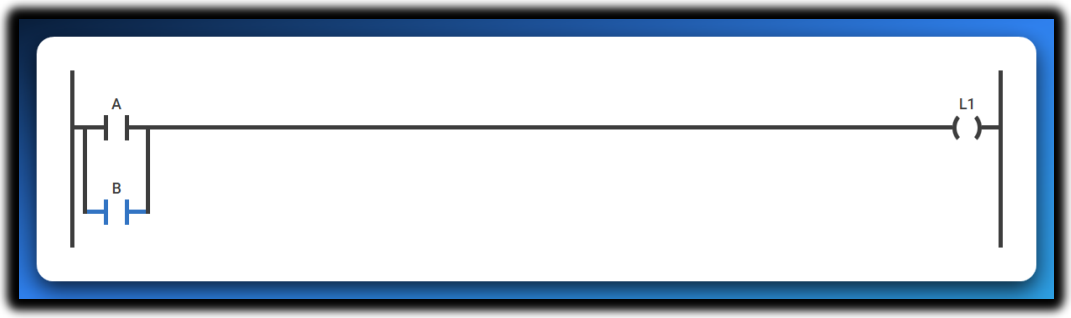
Use **normally open contacts in parallel**.
OR operations are comparable to **addition**.



This is saying:

*"If A **OR** B is TRUE, then turn ON the output."*

As long as at least one of them is ON, the rung is complete and current flows to the output.

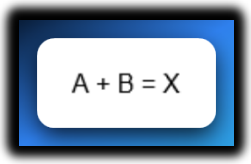


If **A is TRUE** (and B is FALSE), the path through A "closes," allowing logical power to flow to the output.

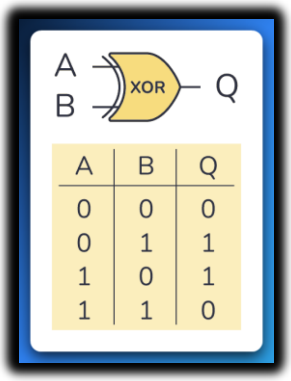
If **B is TRUE** (and A is FALSE), the path through B "closes," allowing logical power to flow to the output.

If **both A and B are TRUE**, both paths "close," and logical power still flows to the output.

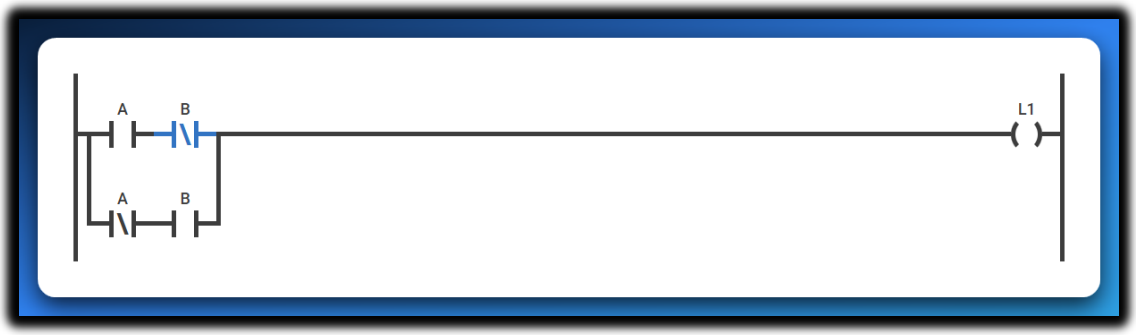
Only if **both A and B are FALSE** will both paths remain "open," stopping the logical power flow and keeping the output OFF.



🔌 XOR gate Ladder Implementation:



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

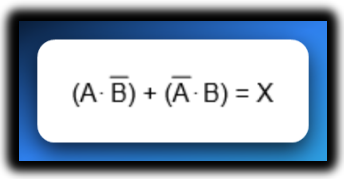


Use a combination of normally open (NO) and normally closed (NC) contacts.
This represents the logic: **"If A OR B is TRUE, but not both, then turn ON the output."**

How it works:

- **If A is TRUE and B is FALSE**:
  The path through A (NO) and B (NC) closes → Output turns ON.
- **If B is TRUE and A is FALSE**:
  The path through B (NO) and A (NC) closes → Output turns ON.
- **If both A and B are TRUE**:
  Both paths are blocked because NC contacts open → Output stays OFF.
- **If both A and B are FALSE**:
  Neither NO contact allows current → Output stays OFF.

✅ **Summary:** Output is ON **only when A or B is TRUE**, but **not both at the same time** — exactly what XOR logic means.



💡 Key Concept: Contacts Are Logic Conditions

In Ladder Logic, your contacts (—| |—) aren't checking voltage directly — they're checking **bit states** in memory.

- A contact in a rung is **like a mini IF-statement**.
- When you place them **in series**, it means "all must be true" (AND).
- When you place them **in parallel**, it means "any can be true" (OR).

And these logic structures determine whether **your output coils (—( )—)** get energized or not.

🧠 Analogy Time:

- Think of series logic like **security clearance**:
  "You need both a keycard **and** a password to enter."
- Think of parallel logic like **alarm triggers**:
  *"If **any** of the sensors trip, sound the alarm."*

📌 Summary:

- **AND Logic = Series contacts** → All must be TRUE.
- **OR Logic = Parallel contacts** → At least one must be TRUE.
- No separate gate components — it's all about **how you arrange your contacts** in the rung.

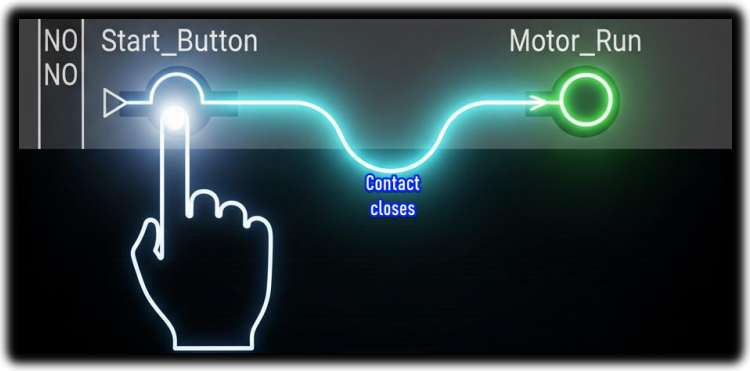
The contacts

🔌 Normally Open (NO) Contact – The "Push-to-Start"

Think of a normally open contact like a simple doorbell button.

- **Default (not pressed):** The internal path is open. No current flows. In PLC terms, the memory bit is FALSE (0).
- **When pressed:** The path closes. Electricity flows. In PLC terms, the bit flips to TRUE (1), and logical power can now travel through that rung.

✅ **Analogy:** Press button → bridge closes → current flows → light turns on.

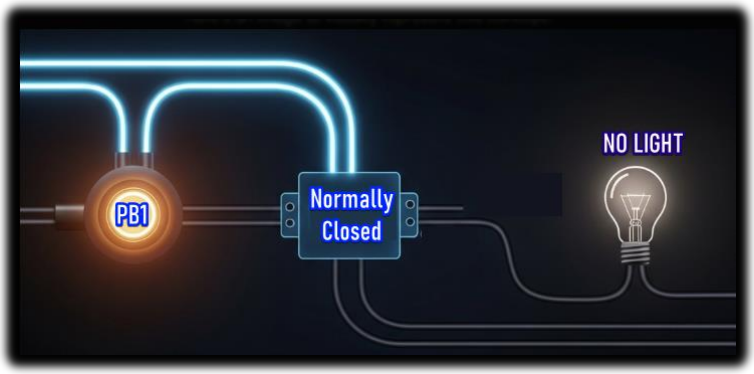


🛡️ Normally Closed (NC) Contact – The "Safety Gate"

Now flip the logic. A normally closed contact is like a safety gate that's already allowing current through until you interfere.

- **Default (not pressed):** The path is closed. Electricity flows. In PLC terms, the memory bit is TRUE (1).
- **When pressed:** The path opens. Current stops. In PLC terms, the bit reads FALSE (0), blocking logical power on that rung.

✅ **Analogy:** Press button → bridge opens → no current → light stays off.



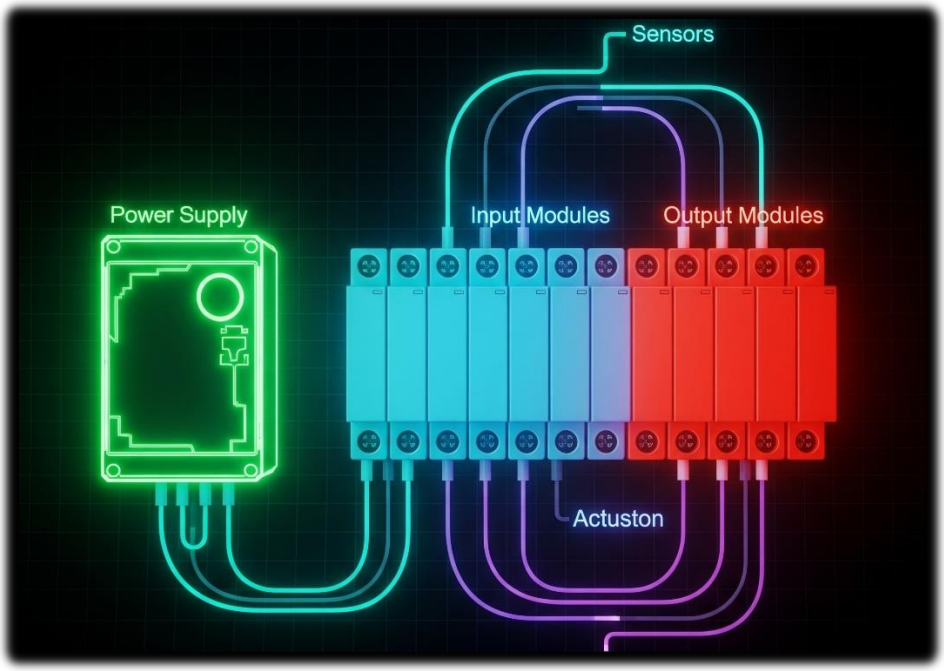
💡 Bottom line:

NO let's power through *when active.* NC stops power *when active.*

Together, they're the Lego blocks of ladder logic—defining exactly when your rungs conduct or block logical "power."

Simple, powerful, and everywhere in your PLC world.

A PLC.



By now we're good to go forward with this topic.

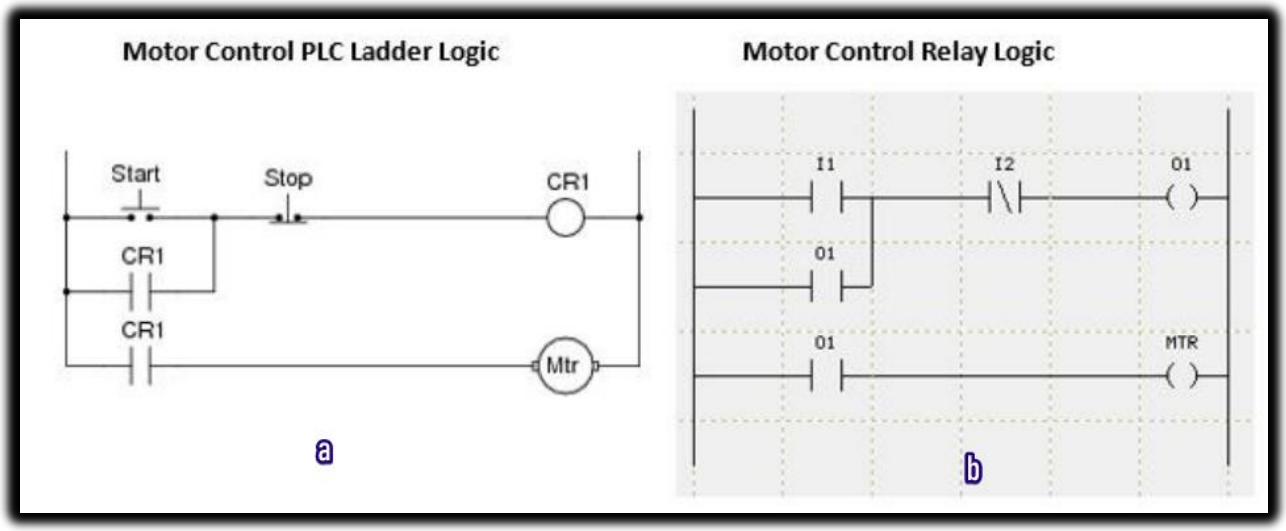
LADDER LOGIC PART 3

Elements of ladder logic

- **Rails** – Vertical lines that carry electrical power to the control circuit.
- **Rungs** – Horizontal lines where the logic is built; they contain inputs, outputs, and branches.
- **Branches** – Parallel paths on a rung that allow multiple input conditions.
- **Inputs** – Devices like switches or sensors that control the logic flow.
- **Outputs** – Devices like motors or lights activated by the logic.
- **Timer** – Delays actions for a set time (e.g., turn on a light after 5 seconds).
- **Counter** – Counts events or cycles and triggers actions after a set number.

I see a small naming confusion I have always had.

In my school, we used “Ladder Logic” to refer to the programs we were drawing.

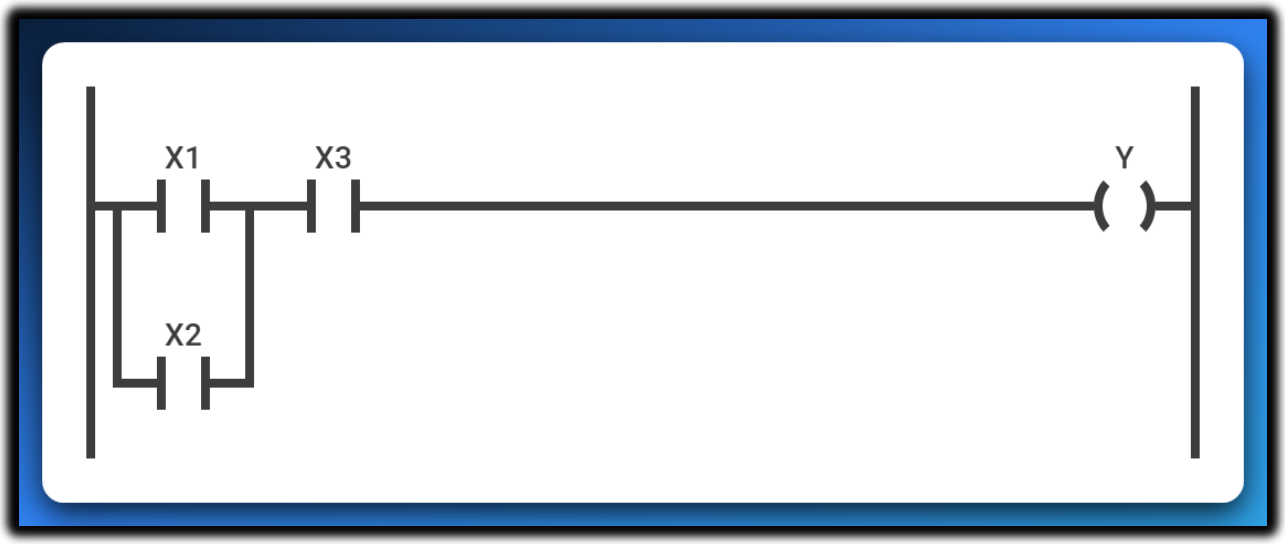


So, I had this conflict calling the **image b** ladder logic.
Now am seeing it being called **relay logic.**

- **Diagram 'a' (Motor Control PLC Ladder Logic):** This is a **program** that runs inside a PLC. It's software.
- **Diagram 'b' (Motor Control Relay Logic):** This is a **wiring diagram** for physical electrical components (relays, switches, motor). It's hardware.

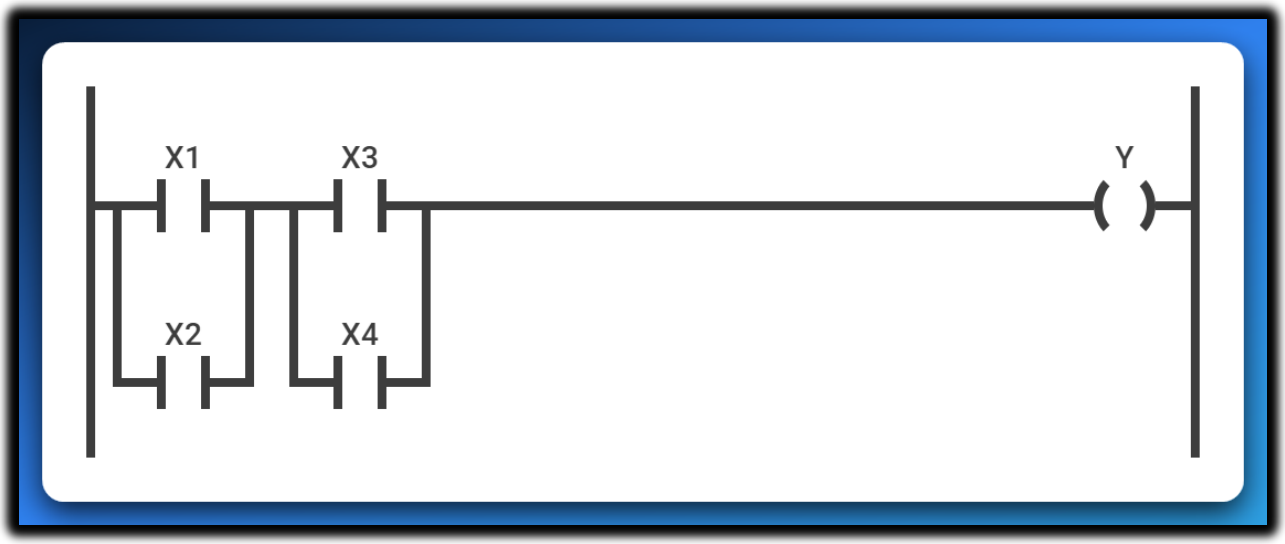
Some Practice

Y = (X1+X2)X3



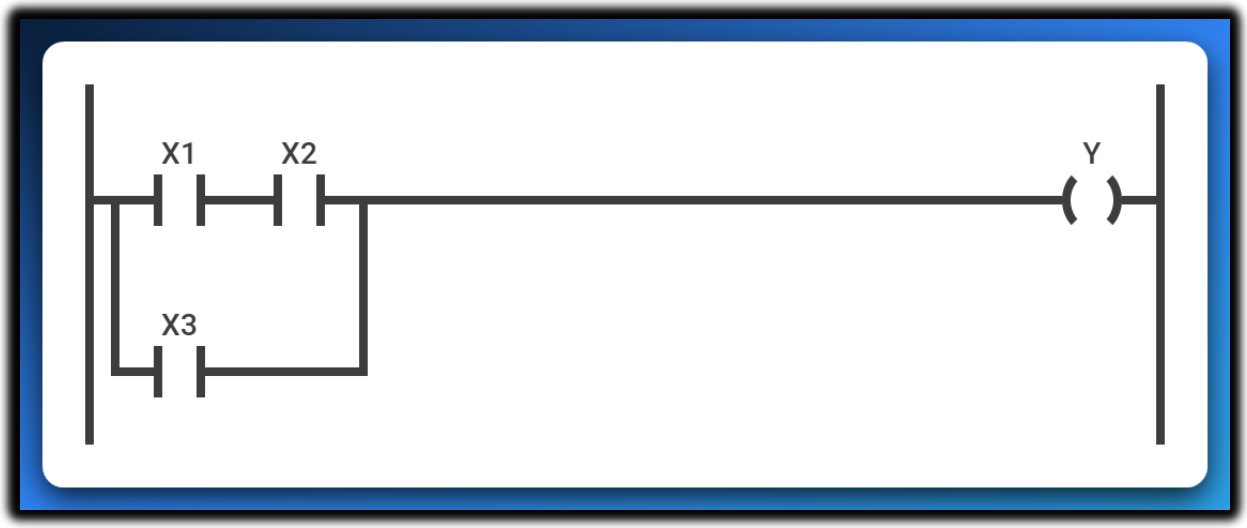
Read it using the AND, OR gates.

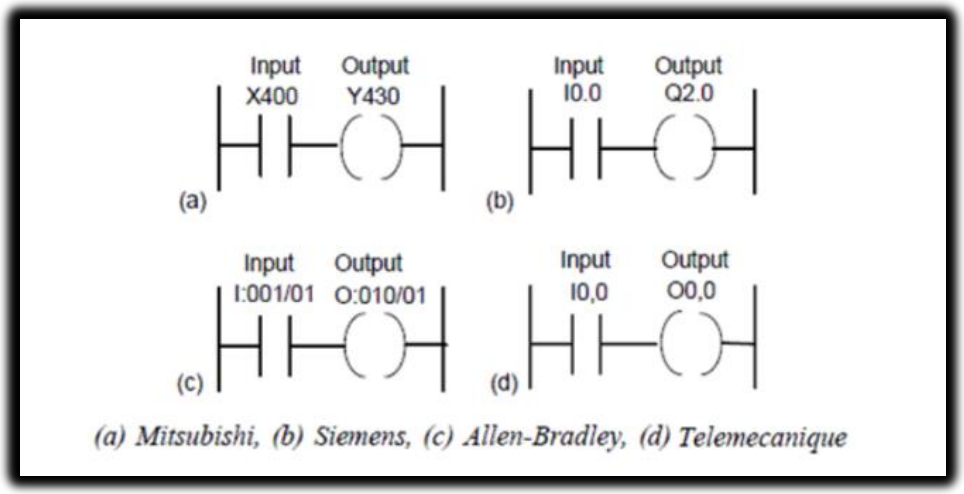
Y=(X1+X2)(X3+X4)



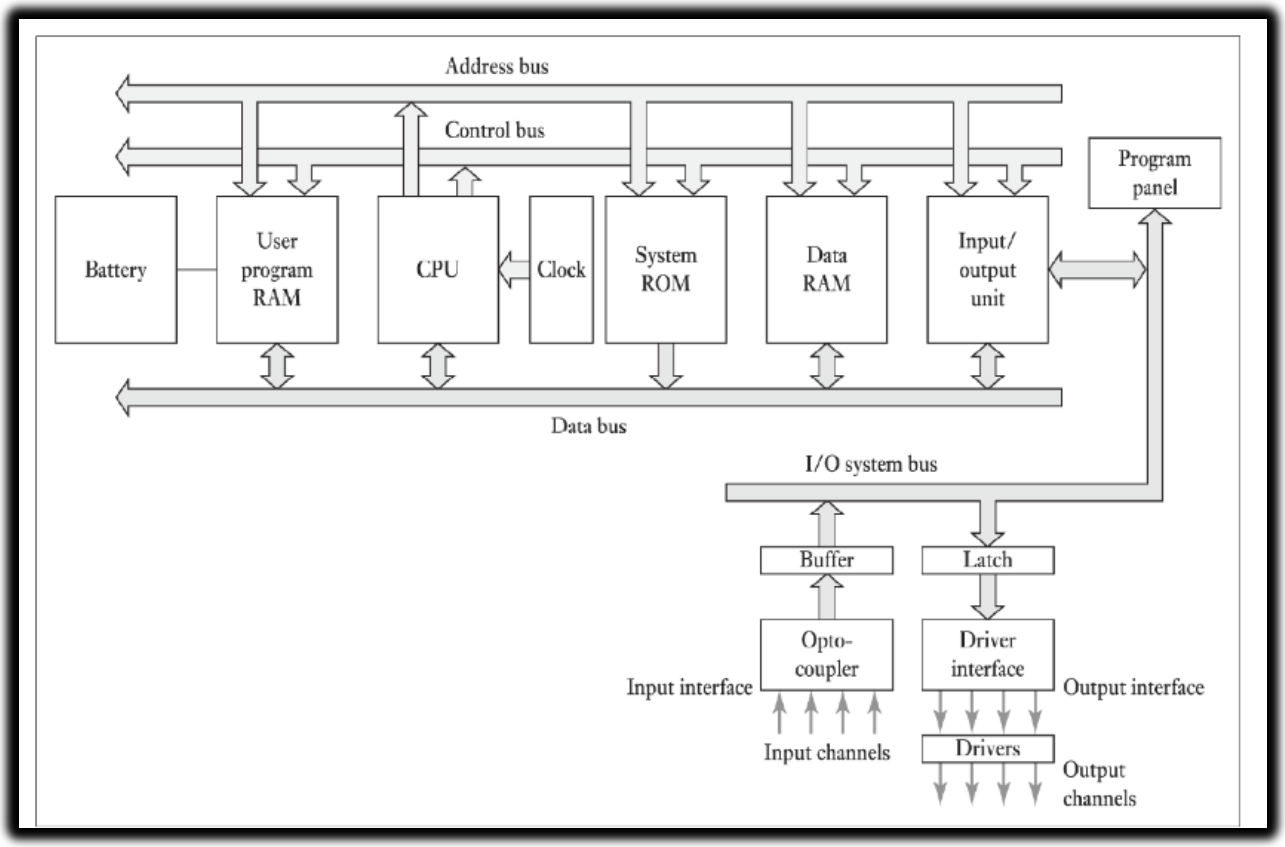
You could attach the midpoint lines as one, or separate them as above.

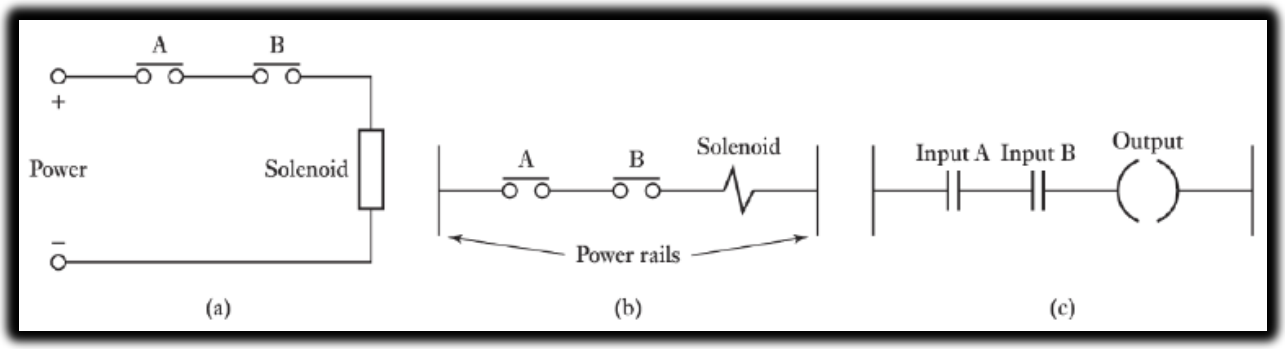
Y= (X1X2) + X3

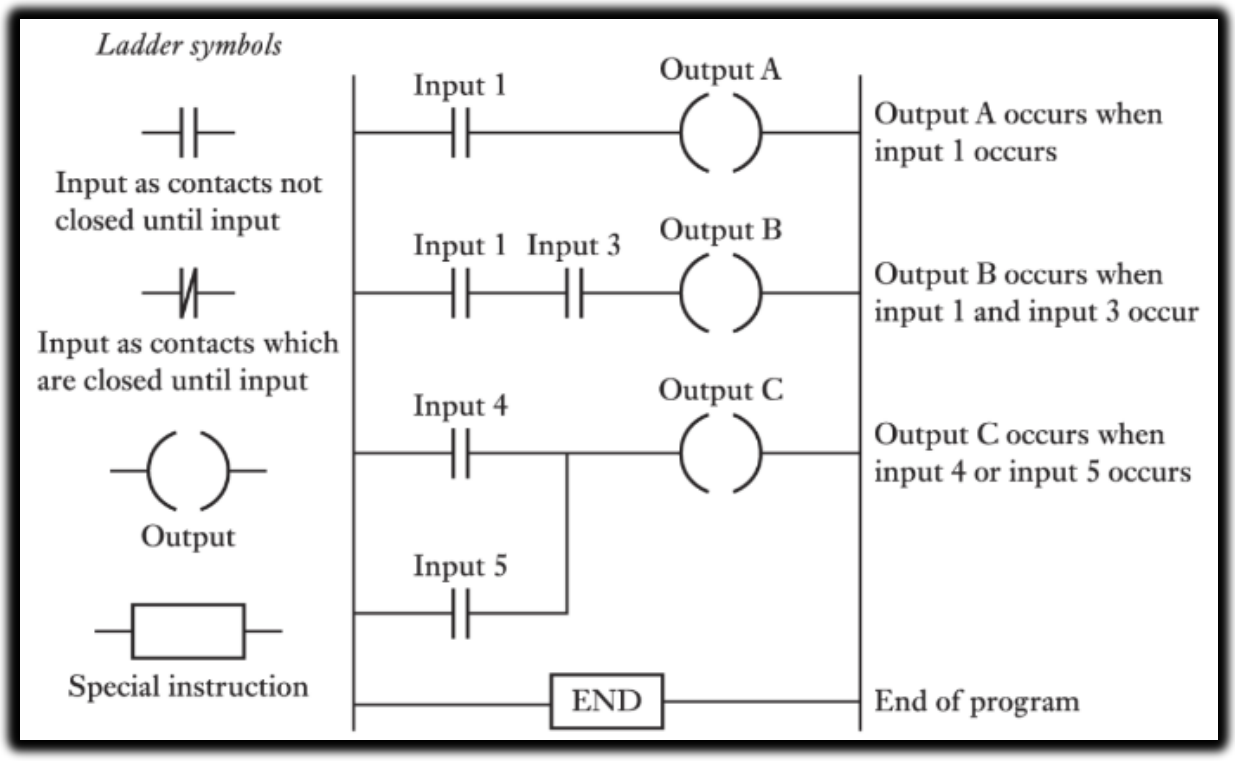


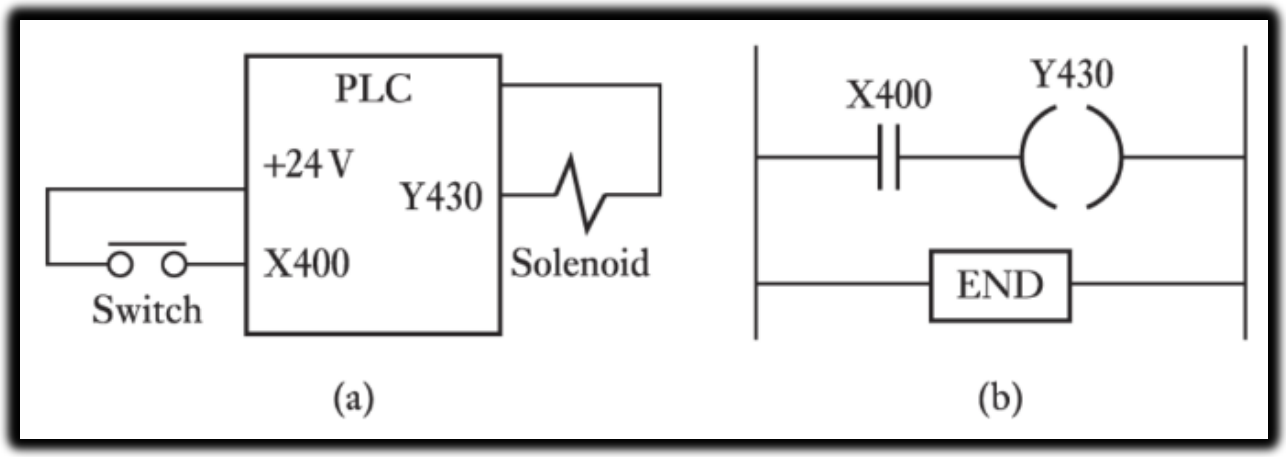


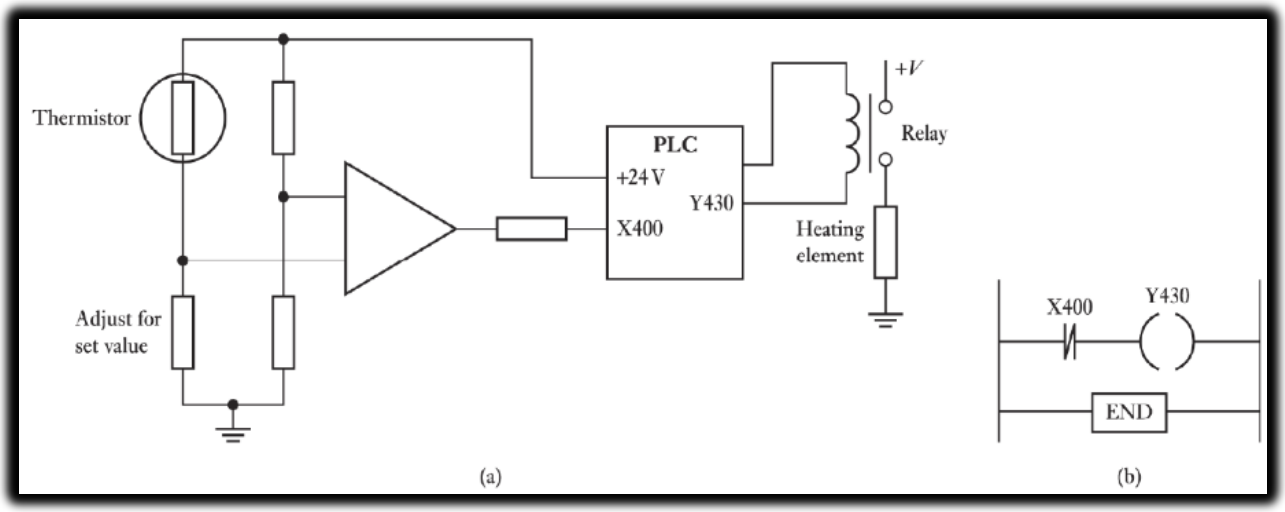
(a) Mitsubishi, (b) Siemens, (c) Allen-Bradley, (d) Telemecanique

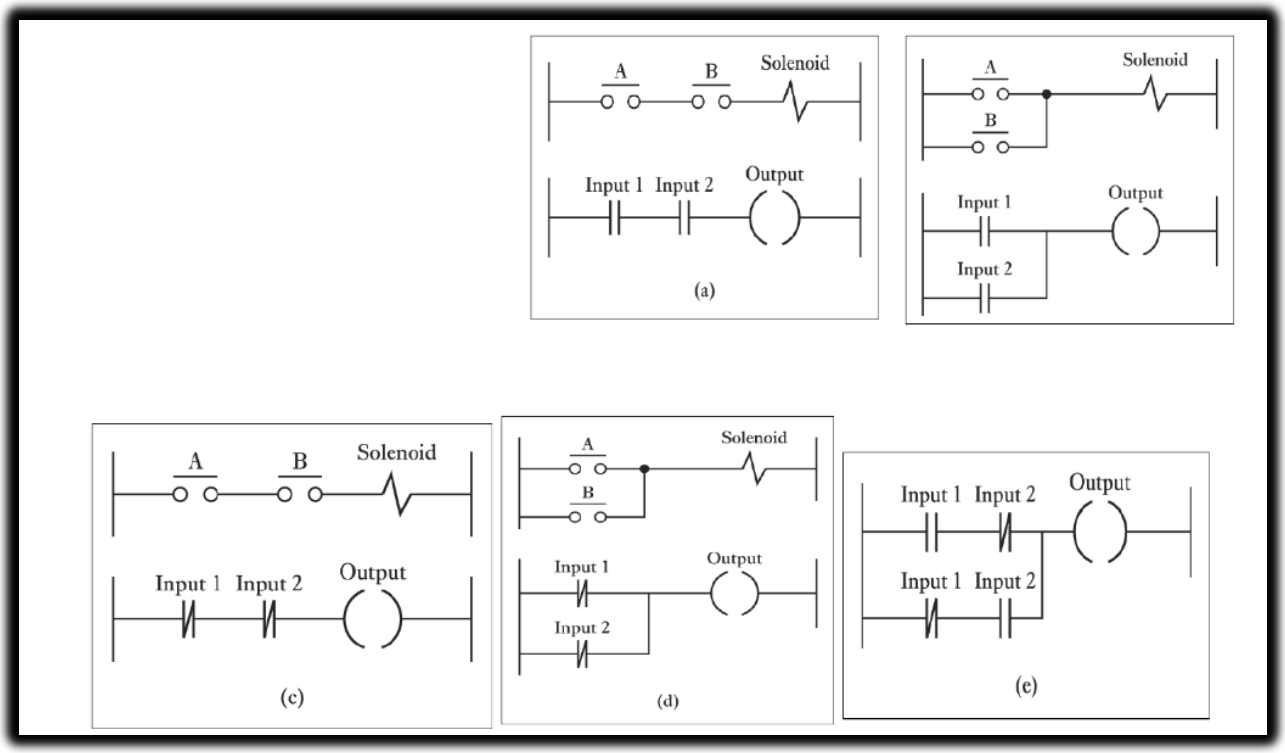

Address bus
Control bus
Program panel
Battery
User program RAM
CPU
Clock
System ROM
Data RAM
Input/ output unit
Data bus
I/O system bus
Buffer
Latch
Opto-coupler
Driver interface
Input interface
Output interface
Input channels
Drivers
Output channels

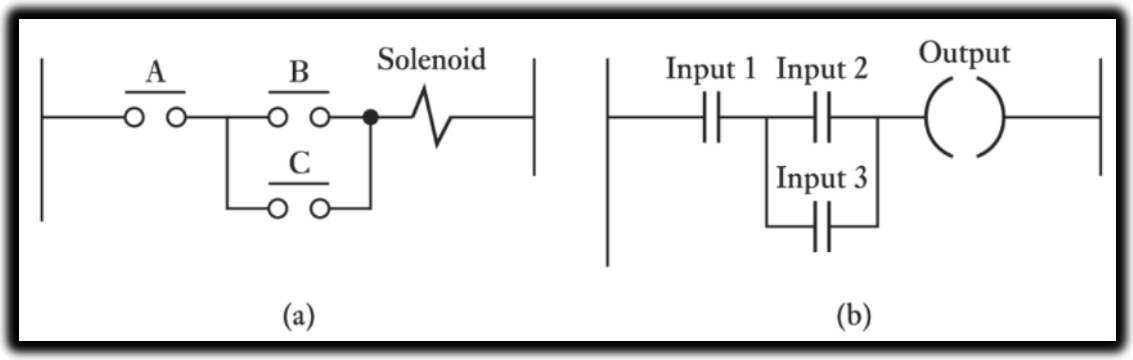

A
B
+
Power
Solenoid
–
(a)
A
B
Solenoid
Power rails
(b)
Input A
Input B
Output
(c)

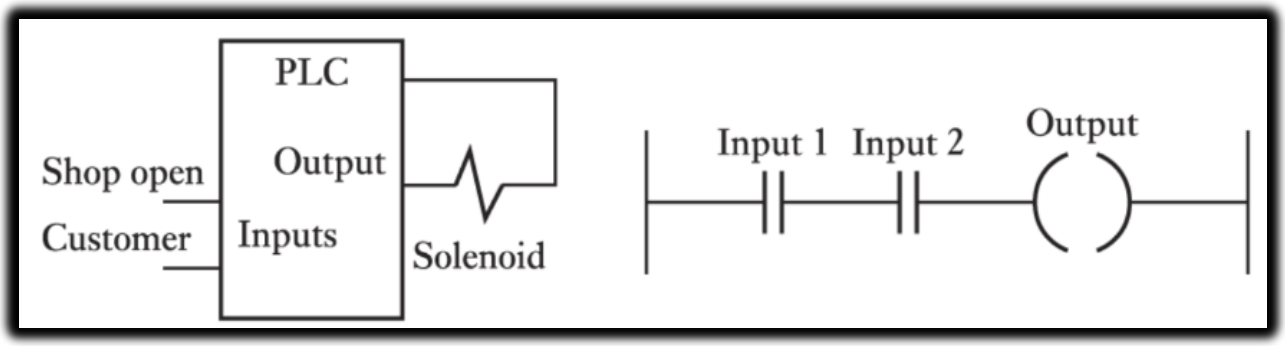

Ladder symbols
Input as contacts not closed until input
Input as contacts which are closed until input
Output
Special instruction
Input 1
Output A
Output A occurs when input 1 occurs
Input 1
Input 3
Output B
Output B occurs when input 1 and input 3 occur
Input 4
Output C
Output C occurs when input 4 or input 5 occurs
Input 5
END
End of program


PLC
+24 V
Y430
X400
Switch
Solenoid
(a)
X400
Y430
END
(b)

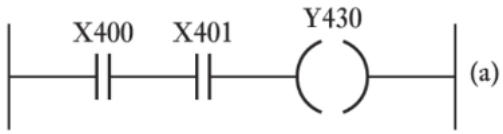

Thermistor
Adjust for set value
PLC
+24 V
X400
Y430
+V
Relay
Heating element
(a)
X400
Y430
END
(b)


A
B
Solenoid
Input 1
Input 2
Output
(a)
(c)
(d)
(e)

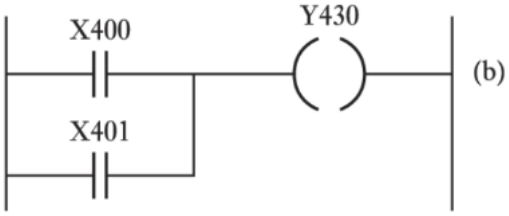

A
B
C
Solenoid
Input 1
Input 2
Input 3
Output
(a)
(b)



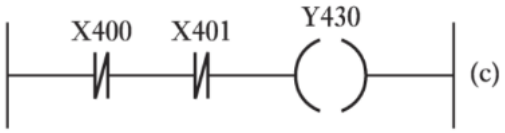
IEC 1131-3	Mitsubishi	OMRON	Siemens	Operation	Ladder diagram
LD	LD	LD	A	Load operand into result register	Start a rung with open contacts
LDN	LDI	LD NOT	AN	Load negative operand into result register	Start a rung with closed contacts
AND	AND	AND	A	Boolean AND	A series element with open contacts
ANDN	ANI	AND NOT	AN	Boolean AND with negative operand	A series element with closed contacts
OR	OR	OR	O	Boolean OR	A parallel element with open contacts
ORN	ORI	OR NOT	ON	Boolean OR with negative operand	A parallel element with closed contacts
ST	OUT	OUT	=	Store result register into operand	An output from a rung



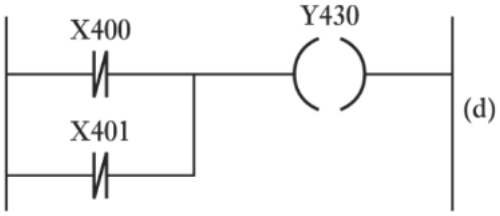

```
LD   X400 (*Input at address X400*)
AND  X401 (*ADD input at address X401*)
OUT  Y430 (*Output to address Y430*)
```



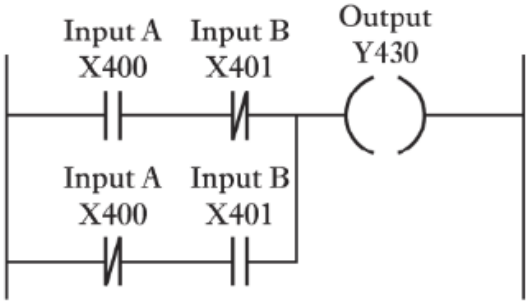

```
LD   X400 (*Input at address X400*)
OR   X401 (*OR input at address X401*)
OUT  Y430 (*Output to address Y430*)
```




```
LDI  X400 (*NOT input at address X400*)
ANI  X401 (*AND NOT input at address X401*)
OUT  Y430 (*Output to address Y430*)
```

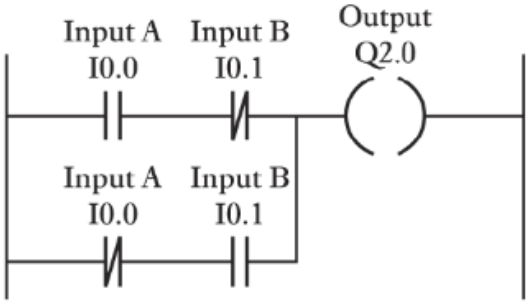



```
LDI  X400 (*NOT input at address X400*)
ORI  X401 (*OR NOT input at address X401*)
OUT  Y430 (*Output to address Y430*)
```



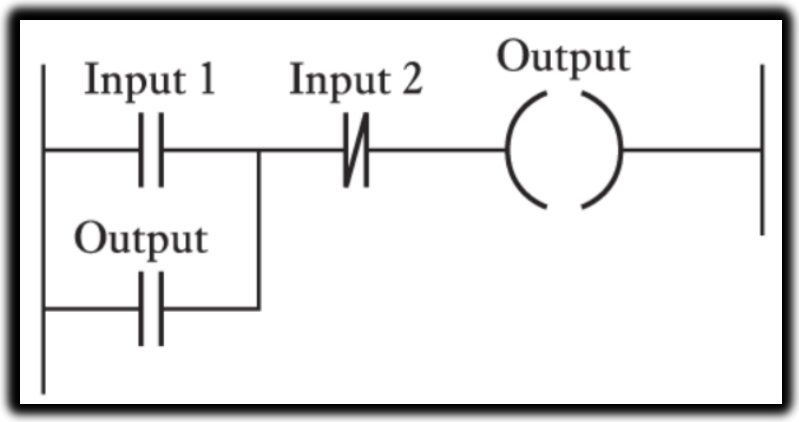
(a)

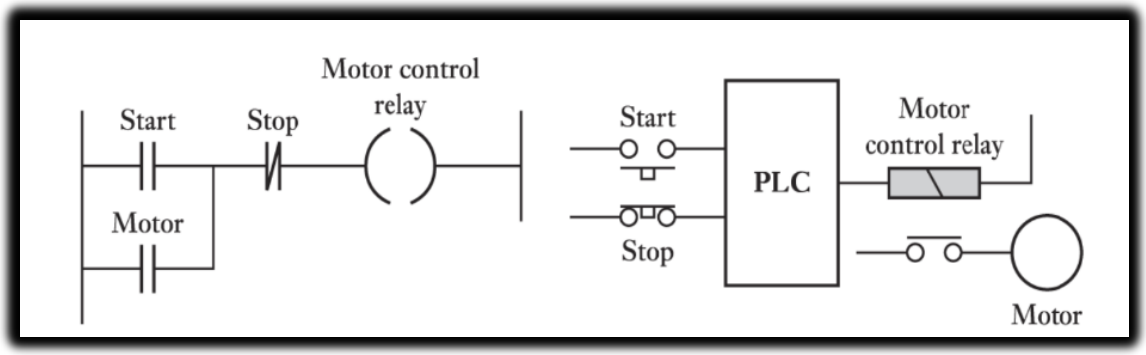
```
LD   X400  (*Load input at address X400*)
ANI  X401  (*AND NOT input at address X401*)
LDI  X400  (*Load NOT input at address X401*)
AND X401  (*AND input at address X401*)
ORB
OUT Y430  (*Output to address Y430*)
```

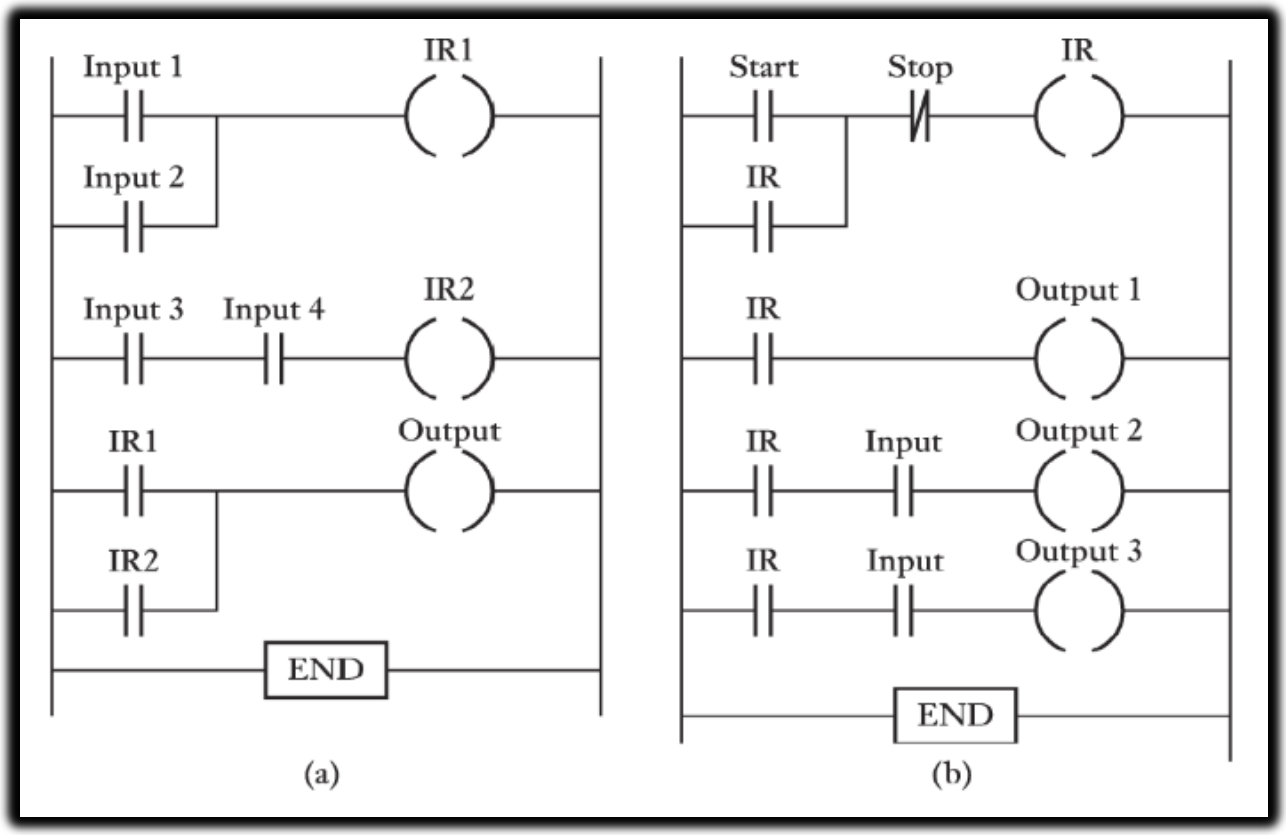


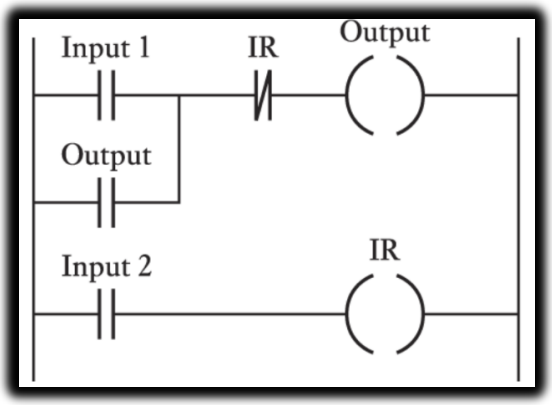
(b)

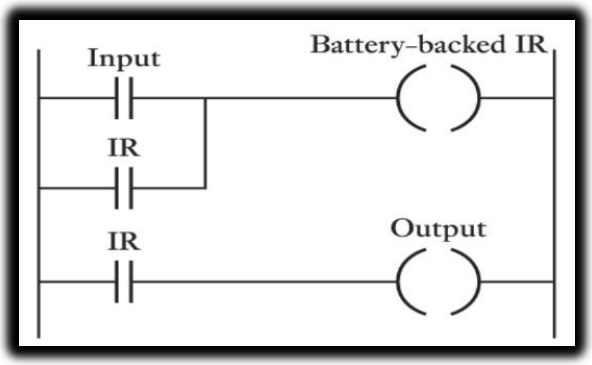
```
A(          (*Load the bracketed term*)
A   I0.0    (*Load input at address I0.1*)
AN  I0.1    (*AND input at address I0.1*)
)
O(          (*OR the bracketed term*)
AN  I0.0    (*Load NOT input at address I0.0*)
A   I0.1    (*AND input at address I0.1*)
)
=   Q2.0    (*Output to address Q2.0*)
```

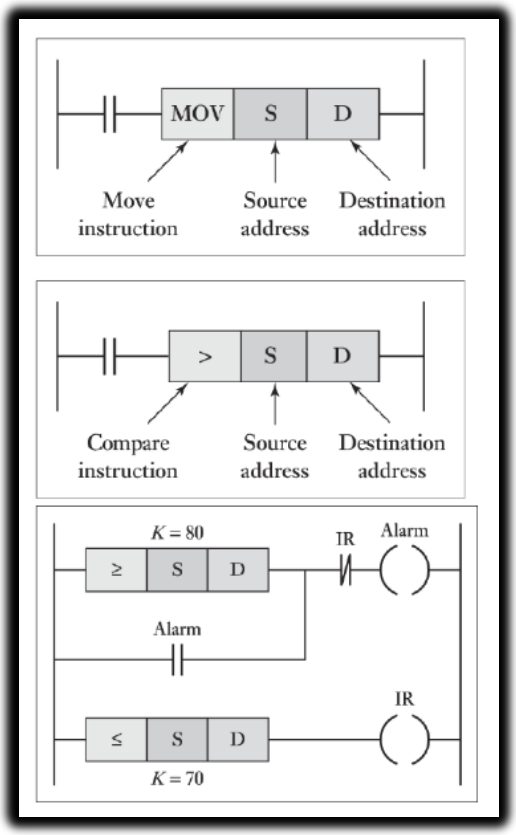

Input 1
Input 2
Output
Output

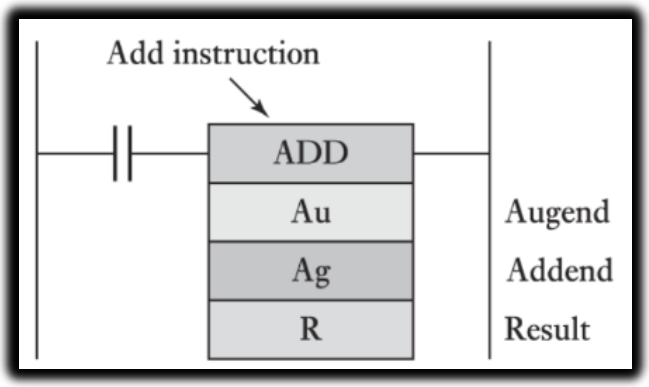

Motor control relay
Start
Stop
Motor
Start
Stop
PLC
Motor control relay
Motor

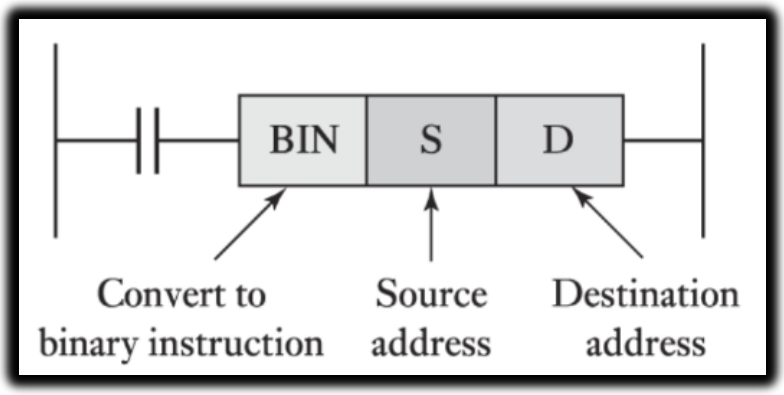

Input 1
IR1
Input 2
Input 3
Input 4
IR2
IR1
Output
IR2
END
(a)
Start
Stop
IR
IR
IR
Output 1
IR
Input
Output 2
IR
Input
Output 3
END
(b)


Input 1
IR
Output
Output
Input 2
IR


Input
Battery-backed IR
IR
IR
Output


MOV
S
D
Move instruction
Source address
Destination address
>
S
D
Compare instruction
Source address
Destination address
K = 80
≥
S
D
IR
Alarm
Alarm
IR
≤
S
D
K = 70


Add instruction
ADD
Au
Ag
R
Augend
Addend
Result


BIN
S
D
Convert to binary instruction
Source address
Destination address