INDUSTRIAL AUTOMATION

DIPLOMA WALLAH

EE/EEE

Unit 05

Program Control Instructions

1) Program Control Instructions: Jump, Subroutines, Calling with Parameters

Introduction

In PLC programming, beyond the basic rung-by-rung sequential logic, you often need to control the flow of program execution: skip certain rungs, call a separate routine (subroutine), reuse code blocks, or pass parameters to a routine. Program control instructions like **JUMP (JMP/LBL)**, **JSR (Jump to Subroutine / Subroutine Call)** enable structured, maintainable, modular PLC code. Using these instructions correctly helps you manage large programs, reduce repetition, and organize logic in a more engineering-grade way.

Definitions & Behaviour

- **JUMP (JMP) Instruction**: An instruction that causes the PLC to skip directly to another part of the program (identified by a label LBL) rather than sequentially executing each rung. Example: if a condition is met, jump over unused rungs. (Inst Tools)
- **Label (LBL) Instruction**: Used in conjunction with JUMP to mark the destination of the jump in the ladder logic. (SolisPLC)
- **Jump to Subroutine (JSR) / Subroutine Call**: A more structured form of flow control — when the JSR is executed, control transfers to a separate routine (subroutine), executes its logic, then returns to the calling point. Parameters may be passed (depending on PLC platform). (Rockwell Automation)
- **Calling a Subroutine with Parameters**: Many PLC brands allow passing input and output parameters to subroutine blocks, making code reusable. For example, in Allen-Bradley Logix5000, a program can call a routine with parameters. (SolisPLC)

Engineering Considerations

- Use subroutines to encapsulate repeating logic (e.g., motor start/stop sequence, fault handling) rather than duplicating rungs.

- Using JUMP instructions can improve performance by skipping large sections of logic when not needed — but may make program harder to read/troubleshoot if over-used. (Many sources advise caution) (SolisPLC)
- Ensure that parameters passed to subroutines are clearly documented and data types match.
- Be aware of scan timing: subroutine calls still fall under the same scan; large subroutines may increase scan time.
- Maintain clear naming and structure: e.g., _InitRoutine, _ProcessRoutine, _FaultRoutine.
- On first scan or restart, ensure initialization logic runs before any subroutine that assumes data is valid.

Summary in Hinglish

PLC me JUMP ya JMP instruction ka matlab hai "agar condition hai to upar wale rungs ko chhod ke seedha doosre part pe jao". Label (LBL) us destination ko mark karta hai. JSR ya subroutine call ka matlab hai "ek alag routine (code block) call karo jo alag file ya section me hai, logic execute karo, phir wapas main program pe aao". Engineering point of view se, subroutine use karne se code modular, maintainable aur repeat-useable hota hai; lekin excessive jumps ya subroutine misuse se program complex ho sakta hai.

---

2) Comparison Instructions in PLC Programming

Introduction

In control applications we often need to compare numeric values (counters, timers, analog conversions, setpoints) and take action based on the comparison result (equal, less than, greater than etc.). PLCs provide **comparison instructions** such as EQU, NEQ, LES, LEQ, GRT, GEQ and LIM. These instructions evaluate two (or more) values and produce a Boolean result (TRUE/FALSE) which can control ladder logic flow. Correct use of these ensures precise decision-making in automation systems.

Definitions & Behaviour

- **EQU (Equal)**: Tests whether Source A = Source B. If true, output = TRUE. (Inst Tools)
- **NEQ (Not Equal)**: Tests whether Source A ≠ Source B. If true, output = TRUE. (SolisPLC)
- **LES (Less Than)**: Tests whether Source A < Source B. TRUE if so. (Inst Tools)

- **LEQ (Less Than or Equal)**: Tests whether Source A ≤ Source B. TRUE if so. (Control.com)
- **GRT (Greater Than)**: Tests whether Source A > Source B. TRUE if so. (Ene)
- **GEQ (Greater Than or Equal)**: Tests whether Source A ≥ Source B. TRUE if so. (Inst Tools)
- **LIM (Limit Test)**: Tests whether Source A is within (or outside) a specified range defined by Low Limit and High Limit. (Inst Tools)

Engineering Considerations

- Ensure operands (Source A and Source B) are compatible data types (INT, DINT, REAL) according to your PLC platform. (SolisPLC)
- Comparison instructions should rarely be the last instruction in a rung; usually combined with contacts/coils to effect action. (Inst Tools)
- Use meaningful variable names (e.g., BottleCount, MaxLimit, FillLevel) so comparison logic is readable.
- For LIM instructions, clearly define low and high limits, and document whether the condition is "inside range" (Low ≤ A ≤ High) or "outside range".
- When using comparisons as part of recipe or parameter logic (e.g., fill weight threshold), make sure updates to operands are synchronized with scan/interrupt logic.
- Test boundary conditions (equal, just less, just greater) to ensure the correct branch triggers.
- Use comparisons to implement thresholds, alarms, counters overflow checks, etc.

Summary in Hinglish

Comparison instructions PLC me use hote hain numerical values ko compare karne ke liye: EQU matlab "barabar hai?", NEQ matlab "barabar nahi hai?", LES matlab "chhota hai?", LEQ matlab "chhota ya barabar hai?", GRT "bada hai?", GEQ "bada ya barabar hai?", aur LIM matlab ek range ke andar ya bahar hai? Engineering wise, ye instructions logic decisions ke liye bahut zaroori hain — jisse system jaanta hai "jab count 10 se bada ho gaya, stop karo", ya "jab level 50 se kam ho gaya, alarm do". Data type, edge conditions aur logic readability ka dhyan rakhna bahut zaroori hai.

---

3) Automatic Bottle Filling System using PLC

Here is a design sketch: sensors/switches, actuators, sequence and a simple ladder logic outline.

System Description

An automatic bottle filling system works as follows:

- Bottles arrive on a conveyor.
- When a bottle is detected at the filling station, conveyor stops.
- A filling valve/solenoid opens for a preset time (or until fill sensor indicates correct level).
- After fill, the valve closes, conveyor resumes to move the bottle to next station.
- Process repeats until stop button pressed.
  You may also include a counter for number of bottles filled, limit check, fault detection etc.

Sensors, Switches & Actuators

Inputs

- Start Push-Button (NO) → I0.0
- Stop Push-Button (NC) → I0.1
- Bottle Present Sensor at filling station (e.g., photoelectric sensor) → I0.2
- Fill Level Sensor (detects when bottle is full) → I0.3
- Bottle Count Limit Reached (optional) → I0.4

Outputs / Actuators

- Conveyor Motor → O0.0
- Filling Valve / Solenoid → O0.1
- Fill Complete Indicator Lamp → O0.2
- Fault/Alarm Lamp → O0.3

Internal Bits / Timers / Counters

- M_READY (system ready) → B0.0
- M_FILLING (valve open) → B0.1
- Timer T_FILL (to control fill time) → T1
- Counter C_BOTTLE_COUNT → C1

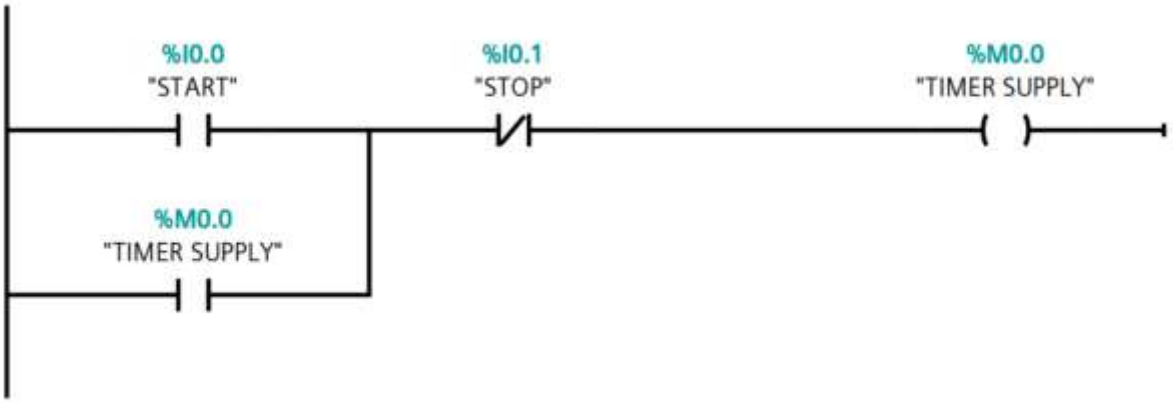
Sequence of Operation

1. Press Start → system ready, conveyor motor ON (assuming auto feed).
2. Bottle arrives at fill station (I0.2 true) → conveyor stops.
3. Fill valve opens (O0.1 ON) and timer T1 starts (or fill sensor monitors).
4. Either fill sensor indicates full or T1 reaches preset → valve closes, fill complete lamp ON momentarily, conveyor restarts.
5. Increment bottle count (C1). If count reaches limit (via comparison instruction, e.g., GEQ C1.PRE = MaxCount) → conveyor stop, alarm.
6. Stop push-button or fault input resets system.

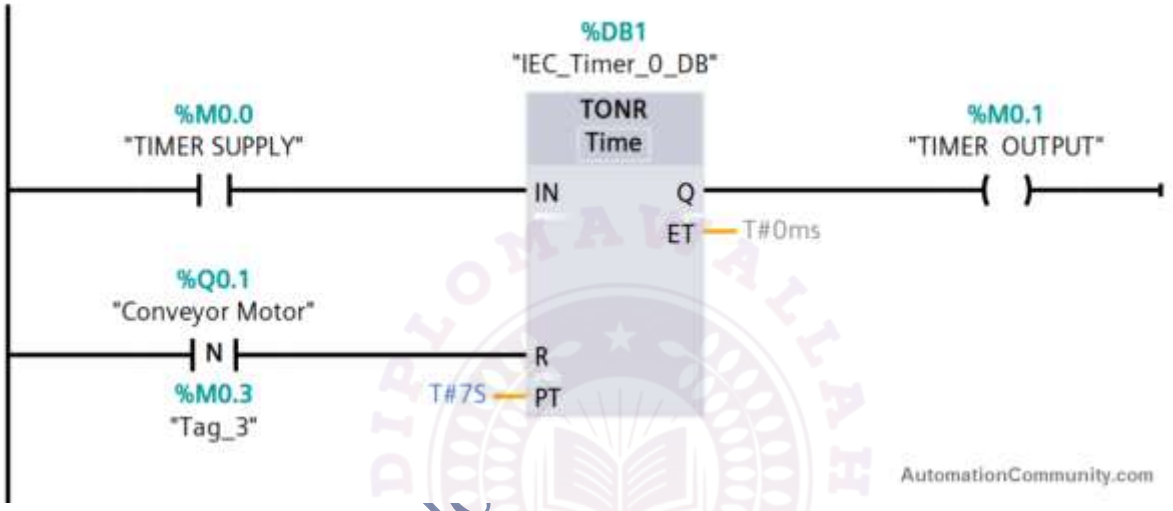
Additional Notes

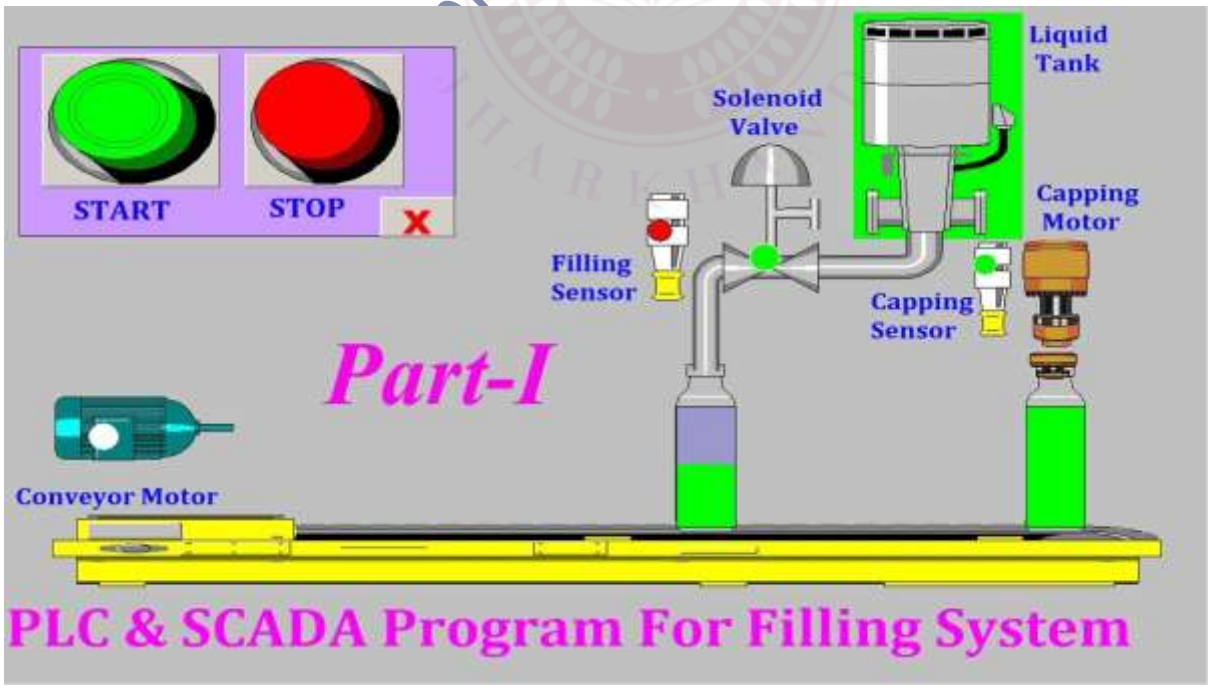
- Use proper fault / sensor missing detection logic.
- Include a subroutine for resetting count or maintenance routine (you could use JSR).
- Use comparison instructions (GEQ, etc) to evaluate counts or levels.
- Use jump/subroutine instructions if you want modular logic (e.g., a subroutine "FillValveControl" that is called each bottle).

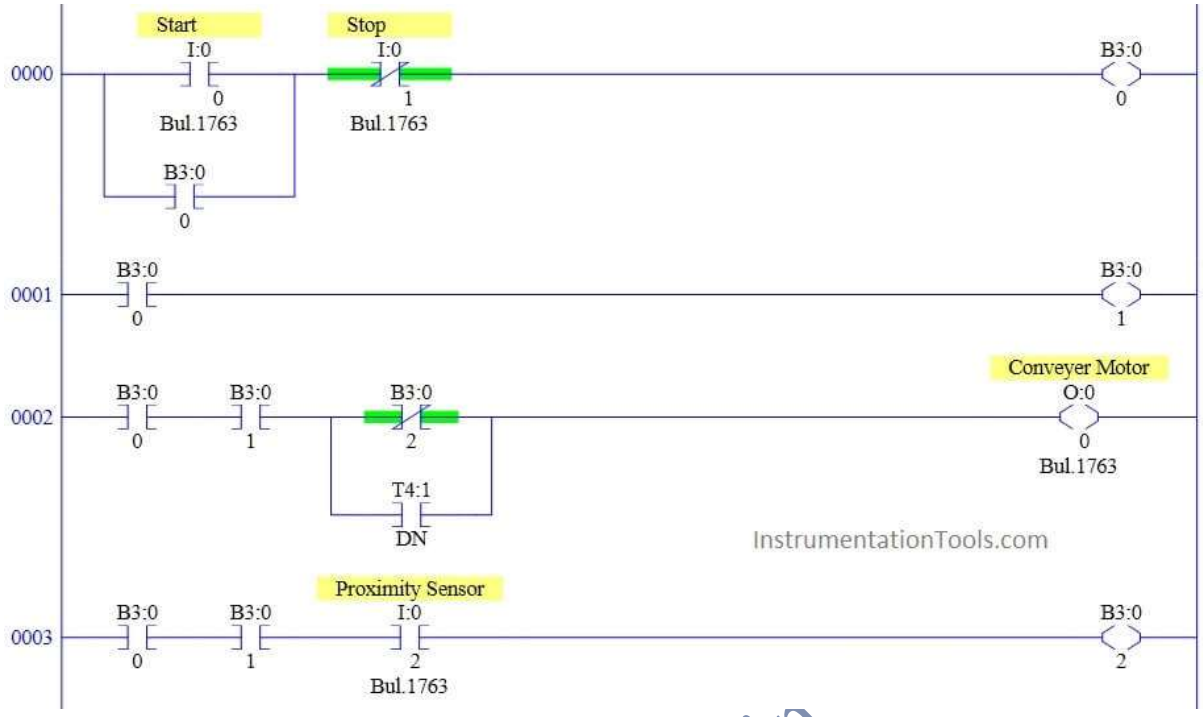
Network 1:

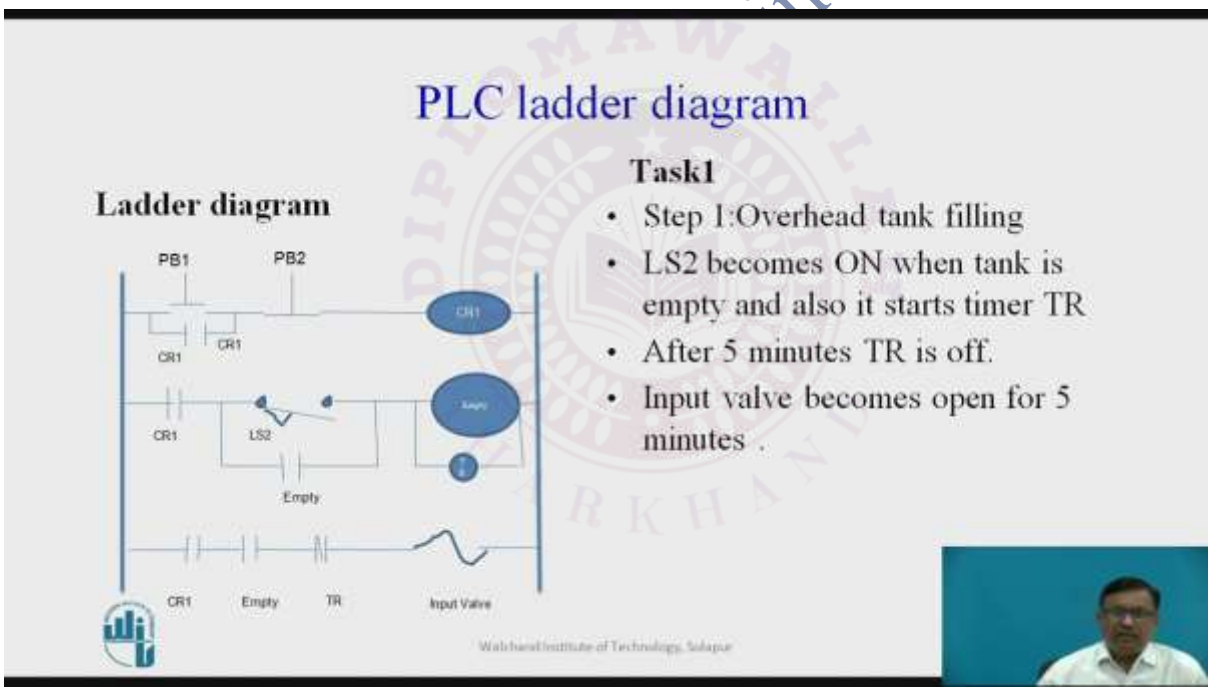


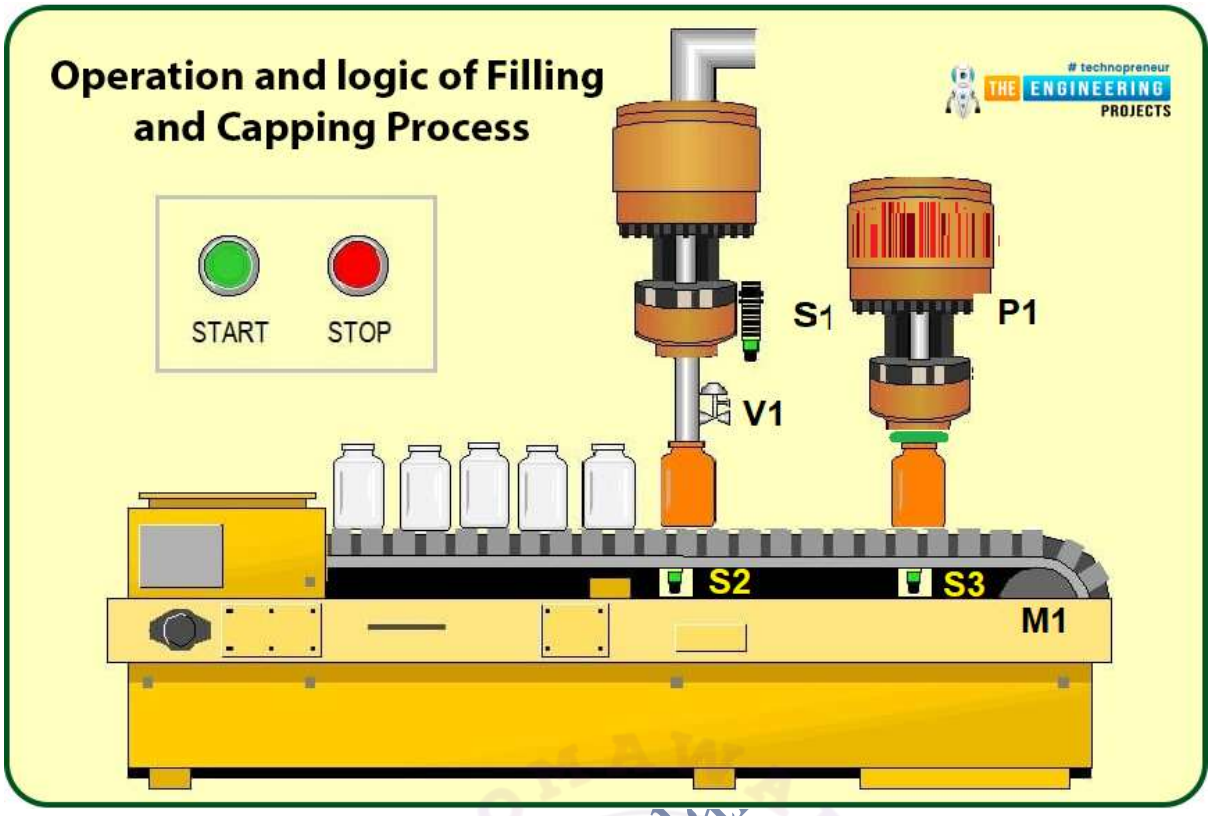
Network 2:

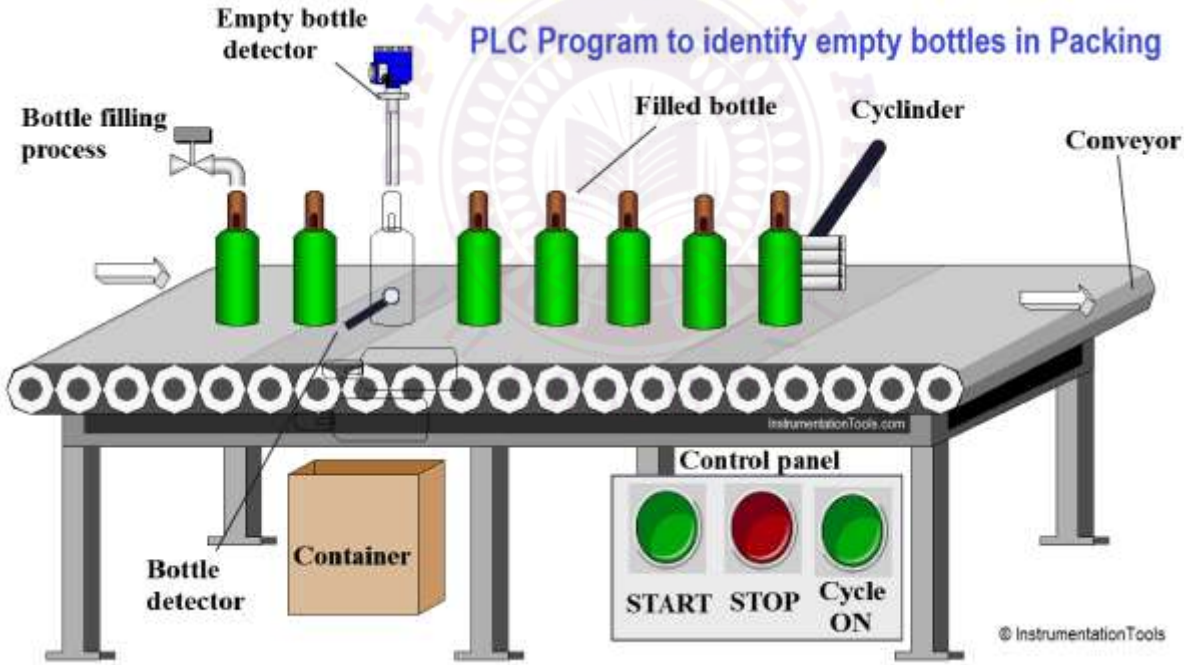












Topic 2: Comparison Instructions in PLC

Introduction

In the realm of industrial automation and PLC programming, numerical decisions often drive the logic: "Is this count above a threshold?", "Has the fill level reached the target?", "Is the temperature within acceptable range?" In such cases, basic ON/OFF contacts aren't enough; you need instructions that *compare* values and yield Boolean results that your ladder logic can use. These are known as **comparison instructions** (EQU, NEQ, LES, LEQ, GRT, GEQ, LIM, etc.). Proper usage of these instructions is foundational to reliable, flexible, and precise automation — especially for monitoring counters, timers, analog conversions, set-points, and limits.

Definitions & Behaviour

Here are the common comparison instructions and what they do:

- **EQU (Equal)**: Tests whether Source A = Source B. If true, output = TRUE (logic flows). (Inst Tools)
- **NEQ (Not Equal)**: Tests whether Source A ≠ Source B. If true, output = TRUE. (solisplc.com)
- **LES (Less Than)**: Tests whether Source A < Source B. If yes → TRUE. (kronotech.com)
- **LEQ (Less Than Or Equal)**: Tests whether Source A ≤ Source B. (LearnVern)
- **GRT (Greater Than)**: Tests whether Source A > Source B. (kronotech.com)
- **GEQ (Greater Than Or Equal)**: Tests whether Source A ≥ Source B. (ene.unb.br)
- **LIM (Limit Test)**: Tests whether a value ('Test') is between two limits: LowLimit ≤ Test ≤ HighLimit (or depending on configuration, outside those limits). (Inst Tools)

How They Work in Ladder Logic & Engineering Considerations

- These instructions are used in rungs to condition logic flow: e.g.,
- [LES SourceA, SourceB] → then Coil/Output
- Source A often must be an address (a register, counter accumulator, memory integer), and Source B can be a constant or another address. (Inst Tools)
- They seldom stand alone as the last instruction in a rung; typically they are followed by a contact or coil instruction. (Inst Tools)
- Examples of practical usage:

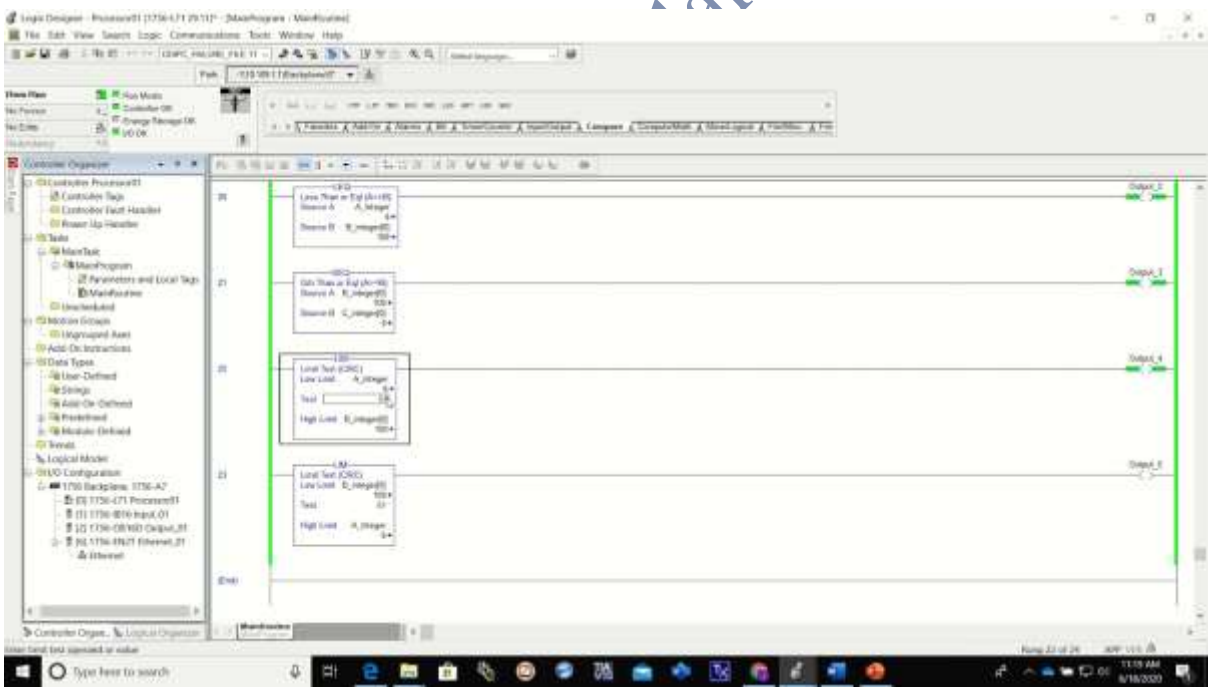
    - A counter has accumulated bottles filled. Use GEQ Counter.ACC, Target to determine when enough bottles filled.
    - A temperature value TestTemp is compared to a maximum limit using GRT TestTemp, MaxLimit.
    - A fill level detector value is compared inside a range using LIM: LowLimit, FillSensorValue, HighLimit.
- Engineering best-practices:
    - Ensure data types match (INT to INT, DINT to DINT, Real/Float if supported). (solisplc.com)
    - Test boundary conditions: For LEQ and GEQ, equal case must be considered true.
    - Use meaningful operand names (not vague registers) so logic is clear.
    - For range checking, use LIM rather than chaining LES + LEQ or similar – makes code cleaner.
- Example of chaining comparisons: You can place two compare instructions in series to check an upper and lower bound: e.g.,

---

Summary in Hinglish

Comparison instructions PLC me use hote hain jab numerical values ko compare karna hota hai — jaise "value barabar hai?", "value chhota hai?", "value badha ya barabar hai?", ya "value ek range ke andar hai?" In instructions ka sahi use control logic ko bahut precise aur robust banata hai. Agar data types ya boundaries galat set hon, to system ya to galat decision lega ya unpredictable behaviour karega.

Topic 3: Limit Test (LIM) Instruction



Introduction

In advanced PLC programming, there are many cases where you don't just want to check if a value is greater than or less than another, but you want to check if it is *within a range* or *outside a range*. The **Limit Test (LIM)** instruction (also called LIMIT in some platforms) allows you to compare a test value against both a lower limit and an upper limit in a single instruction. This makes range-checks simpler and more

efficient than chaining separate greater/less comparisons. For processes like temperature control, level monitoring, or count thresholds, using LIM instruction leads to cleaner and more maintainable logic.

Definition & Behaviour

- The LIM instruction takes three operands: **Low Limit**, **Test Value**, and **High Limit**. It evaluates if the Test Value falls between (or outside) those limits depending on how you set them. (SolisPLC)
- Basic behaviour: If Low Limit ≤ Test Value ≤ High Limit, the instruction outputs TRUE (logic passes). If Test Value is outside that range, output = FALSE. (Inst Tools)
- Special case: If Low Limit is *greater* than High Limit, the instruction logic is inverted — i.e., it becomes a "Test Value outside range" boolean. Some platforms allow this trick. (SolisPLC)
- Data types supported typically include integers (INT, DINT) and real/float types as test and limits. (The Automization)

How It Works in Ladder Logic & Engineering Considerations

- In a rung, you might see something like:
- [LIM LowLimit, TestValue, HighLimit] → Coil/Output

That rung energizes the coil only if TestValue lies between LowLimit and HighLimit.

- Because it is a single instruction, it simplifies logic compared to two separate instructions (e.g., GEQ + LES). This reduces scan load and avoids duplication.
- Engineering use cases:
    - Temperature between safe limits: e.g., 20 °C ≤ TempSensor ≤ 80 °C → run heater.
    - Production count within target band: e.g., 1000 ≤ PartCount ≤ 1200 → alarm if outside.
    - Level check in a tank: For example, fill only if level is between Low and High thresholds.
- Important considerations:
    - If limits are constants, ensure correct order (Low < High) unless you deliberately want the inverted logic.

  - In platforms like Rockwell Allen-Bradley, the LIMIT instruction may treat number space as circular when dealing with signed integers — meaning if Low > High, range wraps around. (Rockwell Automation)
  - Ensure data types match; mixing floats with ints may require conversion. (The Automization)
  - On first scan, if TestValue or limits aren't initialized, logic may behave unpredictably; include initialization.

- Example: If you want to check if a process pressure is between 50 psi and 150 psi:
- LIM 50, PressureValue, 150 → Coil "PressureInRange"

If PressureValue = 100 → coil ON. If PressureValue = 160 → coil OFF.

Summary in Hinglish

LIM instruction ka matlab hai ke aap ek value ko **do limit-points ke beech** check kar rahe ho — yani "value ≥ LowLimit aur ≤ HighLimit" ho tab logic TRUE ho. Agar value is range ke bahar ho, to logic FALSE ho. Engineering ke point of view se, agar aapko continuously chain karne ki bajay ek simple range-check karna hai (jaise temperature, level, count ke liye) to LIM instruction bahut useful hai. Bas dhyan rahe: LowLimit sahi set karo, data type match karo, aur agar limit order reverse kar rahe ho (Low > High) to logic invert ho sakta hai—ye carefully design karo.

Program Control Instructions: Jump / Subroutine / Calling a Subroutine with Parameters

---

Introduction

In complex automation systems, a PLC program isn't just a linear sequence of rungs. We often need modularity, reuse of logic blocks, conditional branching, efficient execution of common tasks, and better readability/maintainability. That's where program control instructions—like jumps (JMP), subroutines (JSR/SBR), labels (LBL), returns (RET) and parameter-passing to subroutines—come into play. These instructions allow programmers to direct the flow of the ladder logic in non-sequential ways, call subsidiary routines, pass data in and out of those routines, and thereby structure large programs more like software rather than simple relay logic.

Definitions & Behaviour

Jump Instructions (JMP / LBL)

- A **JMP (Jump)** instruction causes the PLC execution to skip from the location of the JMP to a labelled point (defined by **LBL (Label)**) within the same program or routine. (SolisPLC)
- Conditioned: typically placed with preceding logic (contacts) so that when the condition is true, the jump occurs. If false, logic proceeds normally. (SolisPLC)
- Label (LBL) marks the destination rung; the JMP references that label by name. (SolisPLC)
- Use case: Skip blocks of logic when not needed; implement mode selection (manual vs auto) by jumping past sections.
- Caution: Jumping over logic means that logic is **not executed** for that scan. That may leave outputs or bits in a stale state if not carefully designed. Also over-use makes program difficult to trace/debug. (SolisPLC)

Subroutine Instructions (JSR / SBR / RET)

- **JSR (Jump to Subroutine)**: This instruction invokes a separate routine (subroutine) when the preceding logic condition is true. Execution transfers to the subroutine and upon completion, returns to the instruction following the JSR. (Rockwell Automation)
- **SBR (Subroutine)**: In the called routine, an SBR instruction may appear at the beginning to define input parameters received. (Rockwell Automation)
- **RET (Return from Subroutine)**: Used inside the subroutine to end it and return control (and optionally return parameters) back to the caller. (Rockwell Automation)
- **Parameter Passing**: Many PLC platforms allow passing parameters (inputs and/or outputs) into subroutines via JSR and SBR/RET. For example, JSR may include input parameters, and the SBR in the subroutine should match those. (The Automization)
- Benefits: Subroutines allow reuse of logic blocks, better organization, maintenance ease. (Control.com)

Engineering Considerations

- **Modularization**: Break large programs into smaller logically-coherent subroutines (e.g., “FillCycle”, “FaultHandler”, “MotorControl”) so that each can be developed/debugged/tested independently. This is especially beneficial in large systems. (Control.com)

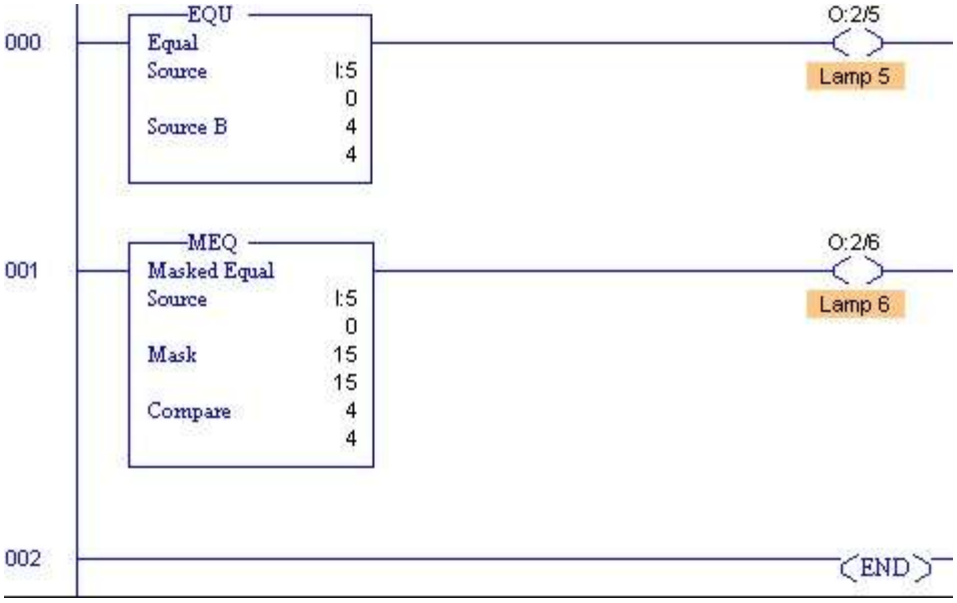
- **Scan time & CPU load**: Subroutines are executed when called; you must ensure that nested/sub-routine calls don't create unacceptably long scan times or watchdog faults. Conditional calls help reduce waste. (SolisPLC)
- **Readability & maintenance**: Overuse of JMP (jump around) can make logic unpredictable and hard to follow; use sparingly. Subroutines with clear names and parameter passing are preferred for clarity. (SolisPLC)
- **Parameter integrity**: When passing parameters into subroutines, ensure data types match (BOOL/INT/DINT etc). Mismatches may cause unexpected system behaviour. (Rockwell Automation)
- **Return logic**: A RET should be used if you need to exit a subroutine early or pass return parameters. If not used, many PLCs default to returning at the end of the routine automatically, but that may produce less predictable behaviour. (PLCtalk)
- **Safe design**: Ensure that essential safety logic is not placed inside a block that might be skipped via JMP or not called. All critical interlocks should always run.
- **Documentation & versioning**: Clearly annotate which routines are called from where, conditions of calls, parameter lists, and ensure consistent naming across project.
- **Debug/troubleshoot**: Because subroutines and jumps change the normal sequential execution, during debugging you must check which routines were executed (via trace/log) and ensure calls/returns occurred properly.

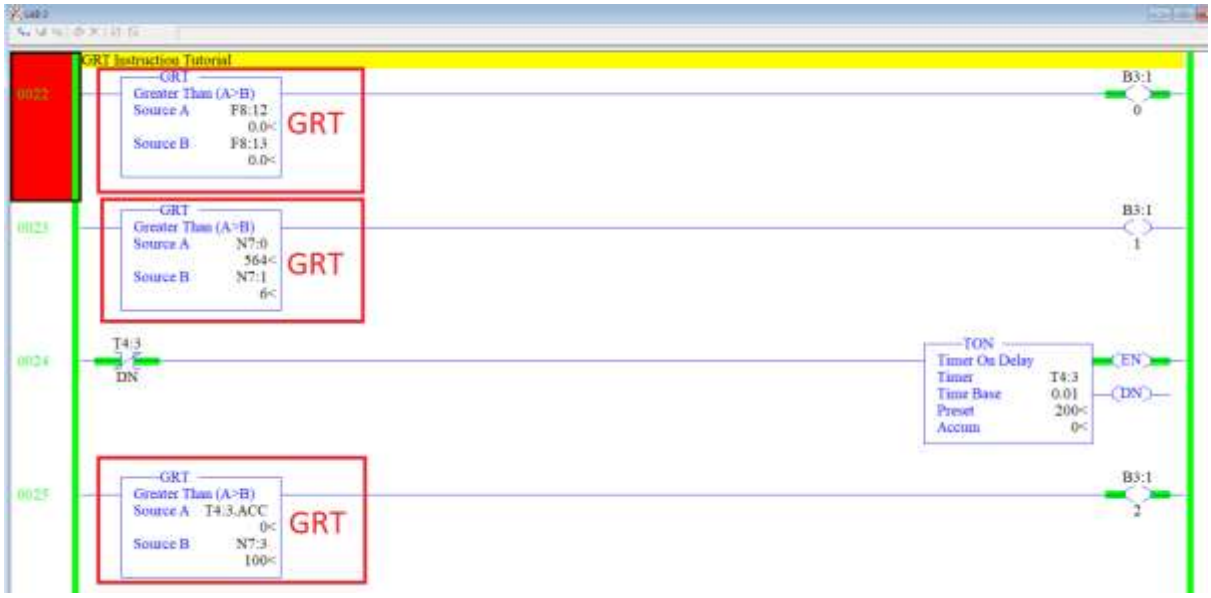
.

Summary in Hinglish

Jump instruction ka matlab hai program flow ko **ek label tak le jaana / skip karna** jab condition sach ho. Subroutine instruction ka matlab hai code ka ek separate block hai jise aap "call" kar sakte ho, logic execute karega, phir wapas aa jaayega. Parameters se aap data subroutine me bhej aur wapas la sakte ho. Engineering wise, yeh techniques large PLC programs ko modular, maintainable aur efficient banane ke liye bahut important hain—lekin misuse se debugging mushkil, scan time badh sakta hai, logic unpredictable ho sakta hai.

Topic 4: Comparison Instructions (EQU, NEQ, LES, LEQ, GRT, GEQ, LIM)





Introduction

In control systems using PLCs, most decisions are not simply "is this switch pressed?" but more complex: "has a counter reached its preset?", "is a temperature above or below the safe limit?", "is a value within a band or outside it?" For such tasks, the basic contact/coils logic is not sufficient; we need comparison instructions. These instructions compare two (or more) numeric values and produce a Boolean result (true/false) that can drive logic. Instruction sets like Allen-Bradley, Siemens or other PLC vendors include comparisons such as Equal (EQU), Not Equal (NEQ), Less Than (LES), Less or Equal (LEQ), Greater Than (GRT), Greater or Equal (GEQ) and Limit Test (LIM). Using them correctly is essential for precision, reliable decision-making, and predictable automation behaviour. (Inst Tools)

Definitions & Behaviour

Here are the key comparison instructions:

- **EQU (Equal)**: Checks whether Source A = Source B. If they are equal, the instruction yields TRUE. Source A must be an address; Source B can be a constant or address. (Inst Tools)
- **NEQ (Not Equal)**: Checks whether Source A ≠ Source B. If they differ, the instruction yields TRUE. (SolisPLC)
- **LES (Less Than)**: Checks whether Source A < Source B. If true, output = TRUE. (Control.com)
- **LEQ (Less Than or Equal)**: Checks whether Source A ≤ Source B. If yes, output = TRUE. (Inst Tools)
- **GRT (Greater Than)**: Checks whether Source A > Source B. If true, output = TRUE. (SolisPLC)
- **GEQ (Greater Than or Equal)**: Checks whether Source A ≥ Source B. If true, output = TRUE. (ene.unb.br)
- **LIM (Limit Test)**: Checks whether a Test value lies between a Low Limit and a High Limit (i.e., Low ≤ Test ≤ High). If yes, output = TRUE. Some platforms invert the logic if the Low Limit is greater than the High Limit (out-side range test). (Inst Tools)

How They Work in Ladder Logic & Engineering Considerations

- In ladder logic, the comparison instruction acts like a special contact; if the comparison condition is true, logic continues to the next instruction; if false, the rung doesn't pass. For example:
- [LES SourceA, SourceB] —(Coil)

- Source A is typically a variable address (counter accumulator, analog value, memory register). Source B may be a constant or another variable. (Inst Tools)
- For LES, GRT, LEQ, GEQ comparisons: data types matter (INT, DINT, REAL) and some platforms allow mixed types but you must check compatibility. (SolisPLC)
- For LIM instructions: Instead of chaining two comparisons (LES & GRT) you can use one LIM instruction to check for "within range" or "outside range" tests — this simplifies logic and improves readability. (Control.com)
- Engineering best practices:
    - Choose meaningful tag names for variables (e.g., BottleCount, MaxLimit, FillLevel) not generic registers.
    - Test boundary conditions: For LEQ/GEQ ensure you understand whether "equal" case should trigger.
    - Use comparison instructions to control branching, alarms, counters overflow, recipe limits, etc.
    - When the logic uses process parameters (e.g., recipe values), use variable operands rather than constants so you can change limits without rewriting logic.
    - Document the purpose of each comparison instruction clearly so future maintenance engineers understand the thresholds.
    - Use series comparisons when necessary (e.g., GEQ & LES in series to check "between"). Example:
    - [GEQ SourceA, LowLimit] – [LES SourceA, HighLimit] → Coil

This means LowLimit ≤ SourceA < HighLimit. This is discussed in many guides. (ene.unb.br)

Example Use Cases

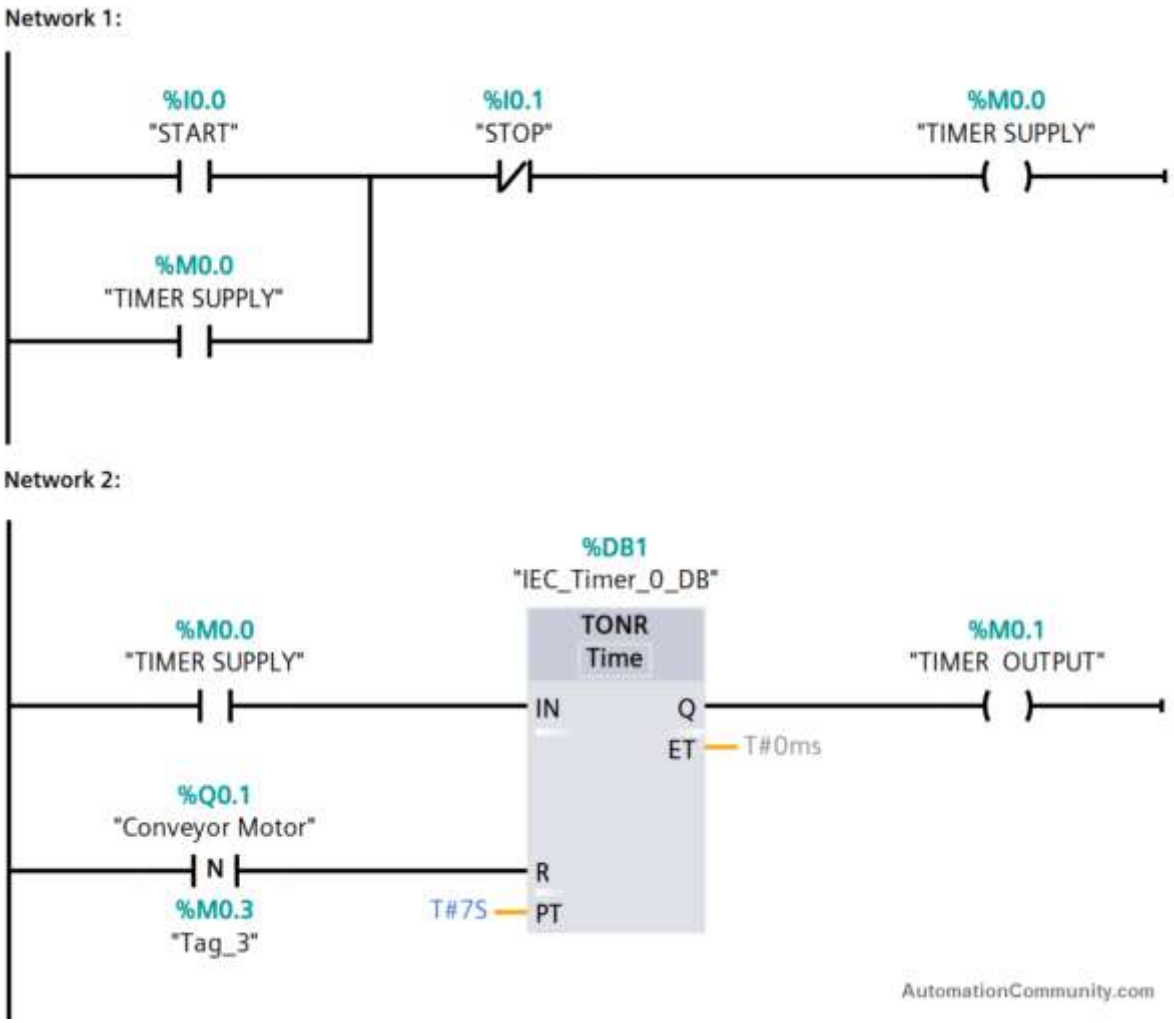
- A bottle-filling line: Count of bottles (C1.ACC) compared with preset target; use GEQ C1.ACC, Target to stop conveyor when filled.
- Temperature monitoring: Use GRT TempValue, MaxTemp to trigger alarm if above max.
- Batch range check: Use LIM LowLimit, FillWeight, HighLimit to ensure weight is between acceptable limits.
- Recipe selection: Use EQ RecipeNumber, 2 to decide which program branch to follow.

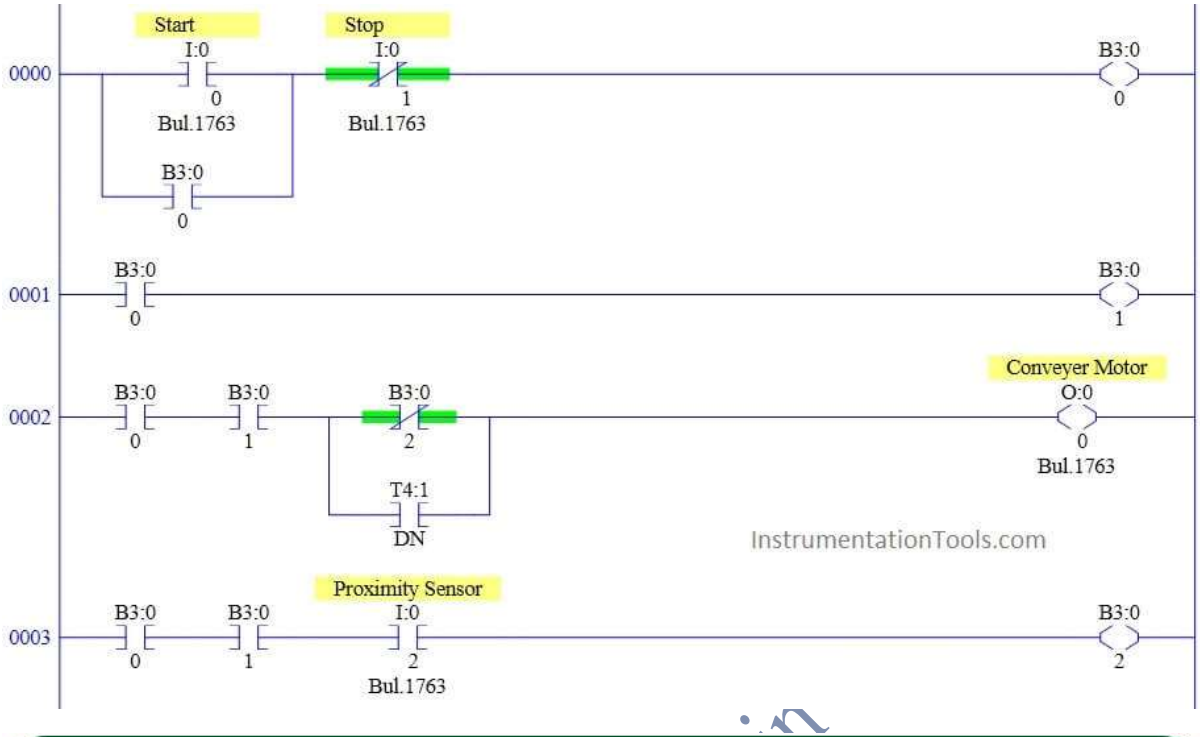
- Fault detection: Use NEQ ExpectedState, ActualState to detect mismatch and trigger fault logic.

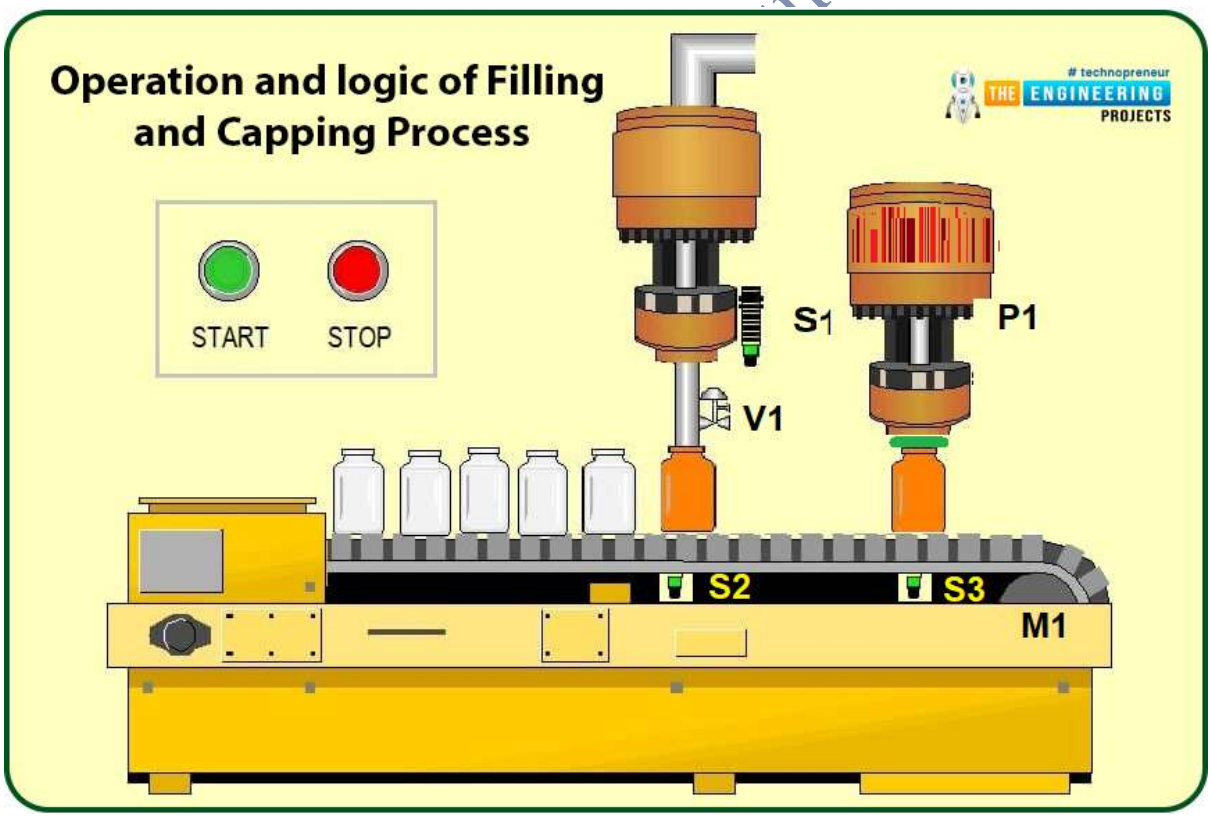
Summary in Hinglish

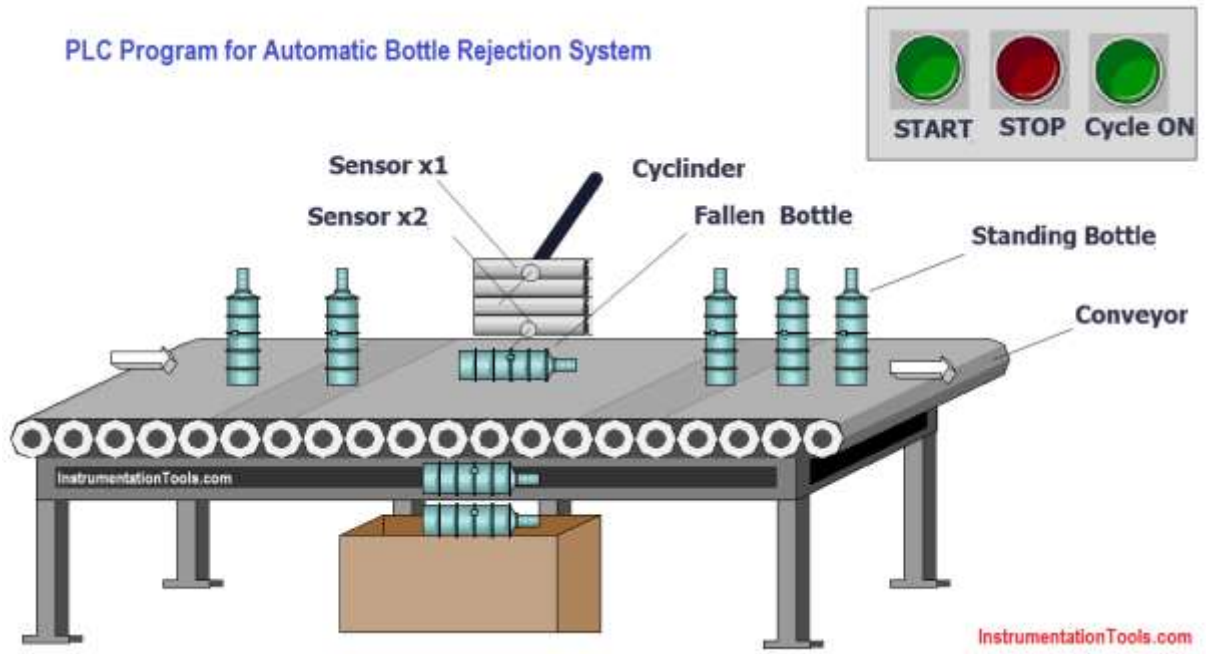
Comparison instructions PLC me zaroori hain jab numerical values ko compare karna hota hai — jaise “barabar hai?”, “chhota hai?”, “bada ya barabar hai?”, ya “value ek range ke andar hai?”. Agar logic ka limit ya threshold sach nahi hai, to process galat result de sakta hai. In instructions ka sahi use automation system ko accurate aur dependable banata hai.

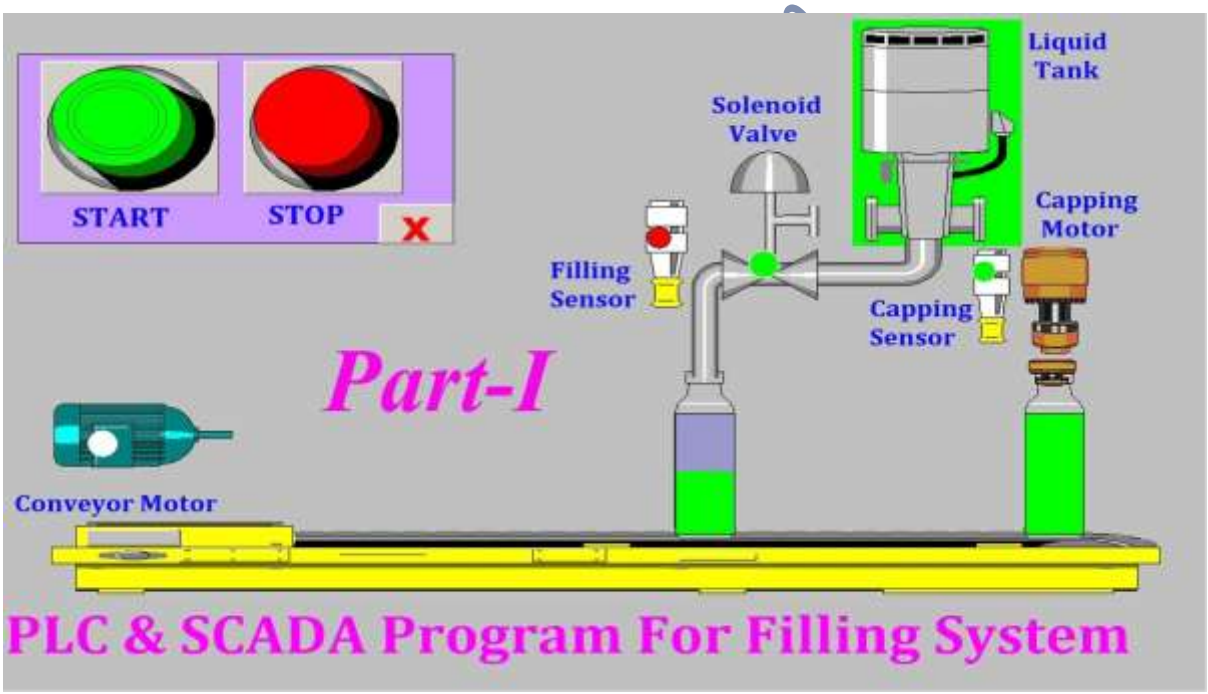
Next Topic: Automatic Bottle Filling System Using PLC

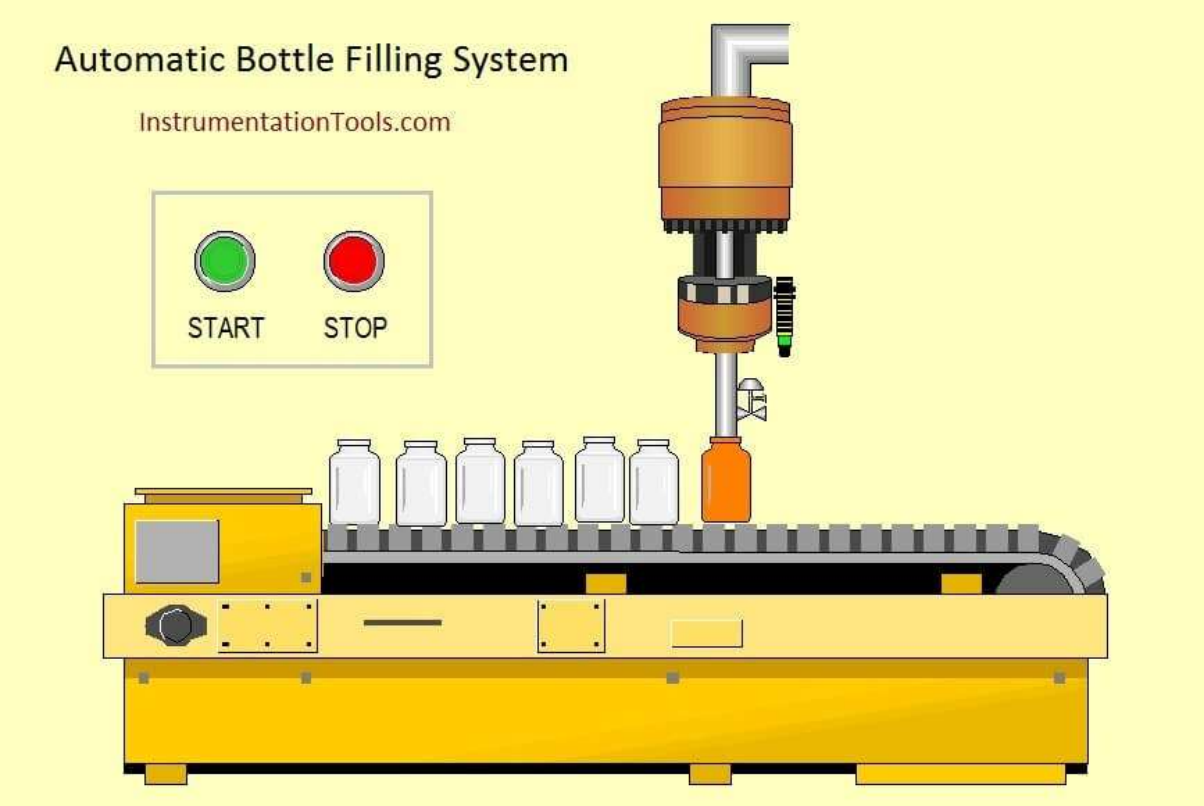



Start
I:0
0
Bul.1763
Stop
I:0
1
Bul.1763
B3:0
0
B3:0
0
0000
B3:0
0
B3:0
1
0001
B3:0
0
B3:0
1
B3:0
2
T4:1
DN
Conveyer Motor
O:0
0
Bul.1763
0002
InstrumentationTools.com
Proximity Sensor
B3:0
0
B3:0
1
I:0
2
Bul.1763
B3:0
2
0003


Operation and logic of Filling and Capping Process
technopreneur
THE ENGINEERING PROJECTS
START
STOP
S1
P1
V1
S2
S3
M1


PLC Program for Automatic Bottle Rejection System
START
STOP
Cycle ON
Sensor x1
Sensor x2
Cyclinder
Fallen Bottle
Standing Bottle
Conveyor
InstrumentationTools.com
InstrumentationTools.com


START
STOP
Liquid Tank
Solenoid Valve
Capping Motor
Filling Sensor
Capping Sensor
Part-I
Conveyor Motor
PLC & SCADA Program For Filling System



1. Introduction

An automatic bottle filling system is an industrial automation setup where bottles are moved via conveyor(s), detected by sensors, filled with liquid via valves or pumps, and transferred onward — all controlled by a programmable logic controller (PLC). This system integrates sensors, switches, actuators, timers, counters, and logic to handle operations like bottle detection, fill-control, conveyor motion, and batching. Such a system is typical in beverage, pharmaceutical, chemical, or packaging industries and provides consistency, speed, and reduced human intervention. (The Engineering Projects)

2. Process Flow & Key Steps

Here is a typical sequence for an automatic bottle filling system:

1. Operator presses **Start** push-button; system becomes ready.
2. Conveyor motor turns ON to move empty bottles into position under the filling station.
3. A **Bottle Present** sensor (proximity, photo-eye, or limit switch) detects an empty bottle in the correct position.
4. Conveyor stops, bottle is positioned.
5. A **Valve/solenoid** opens to allow fill-liquid into the bottle; or a fill mechanism starts.

6. A **Fill Level Sensor** or timer monitors fill time or fill level; when correct level reached, valve closes.
7. Conveyor restarts to move filled bottle to next stage (cap, label, removal).
8. Optional: A **Counter** increments bottle count; if preset target reached, system enters stop or batch complete mode.
9. Operator presses **Stop** or fault/safety sensor triggers stop; system halts. This sequence is described in a PLC-logic tutorial: the process continues until stop button pressed. (Inst Tools)

3. Inputs, Outputs, Sensors & Actuators

Typical Inputs (to PLC):

- Start Push-Button (NO)
- Stop Push-Button (NC)
- Bottle Present Sensor (proximity or photo-electric)
- Fill Level Sensor / Overflow Sensor
- Safety/Interlock Sensors (e.g., guard door, fault detection)

Typical Outputs (from PLC):

- Conveyor Motor Contactor
- Filling Valve / Solenoid
- Fill Complete Indicator Lamp
- Alarm/Buzzer if fault or batch end
- (Optional) Counter Reset or Batch Done Lamp

Sensors & Switches Selection Considerations:

- Bottle detection sensor should be fast and reliable (photo-eye, fibre optic) because bottles are moving.
- Fill level sensor must prevent overflow – whether via float, capacitive sensor, or timer fallback.
- Safety sensors must ensure no human access during operation; these are often NC type so failure triggers stop.

Actuators Selection:

- Valve or solenoid must have appropriate flow rate and response time for fill speed.

- Conveyor motor and drive must handle start/stop cycles smoothly; use frequent start-stop logic.
- Indicators and alarms help operator awareness and fault handling.

5. Testing & Simulation

- Use a PLC simulation software (e.g., Siemens TIA Portal, RSLogix Emulate, Factory I/O) to test the logic before implementing on real hardware.
- Simulate bottle arrival (sensor signals), fill time, level sensor, conveyor start/stop transitions.
- Test edge cases: no bottle arrives, sensor fails, fill timer fault, stop button pressed mid-fill.
- Validate counters and batch termination logic.
- Log states for debugging: show bottle count, fill states, fault flags.

6. Engineering Considerations

- **Safety and fail-safe wiring**: Critical sensors (stop button, guard door sensor) should be wired NC so that a failure/interruption causes stop.
- **Timing vs speed**: Bottle speed and fill time must be coordinated; if conveyor moves too fast, bottle may not be under fill station long enough.
- **Sensor reliability**: Use appropriate sensors for environment (liquid splash, reflections, transparent bottles).
- **Level detection vs overflow**: Ideally use a level sensor; fallback to a maximum timer to avoid overflow in sensor failure.
- **Batch handling**: If production runs in batches, integrate counting and batch end logic (e.g., pump shutdown, alarm, reset).
- **Maintenance mode**: Include manual/auto mode logic, emergency stop, and reset routines.
- **Documentation**: Provide I/O list, ladder logic comments, flow charts for maintenance engineers.
- **Hardware vs software division**: Some interlocks may be better implemented in hardware (e.g., safety circuits) to reduce software complexity.

Made with 💗 by Sangam