

PLC Training Material Compilation and Examples

Christopher Shu (shuchris@umich.edu)

Adrian Pinto (adrianp@umich.edu)

Winter 2017

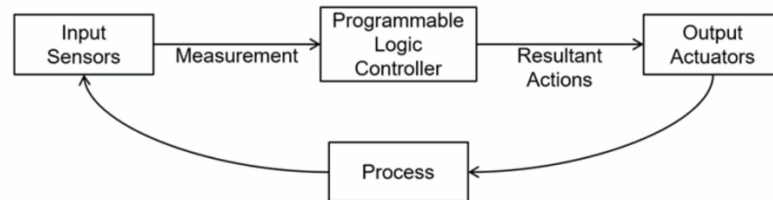
EDITS:

- 04/11: Initial compilation
- 09/07: Added links (may or may not be useful or relevant) – CS
- 10/07: Added TON bit descriptions

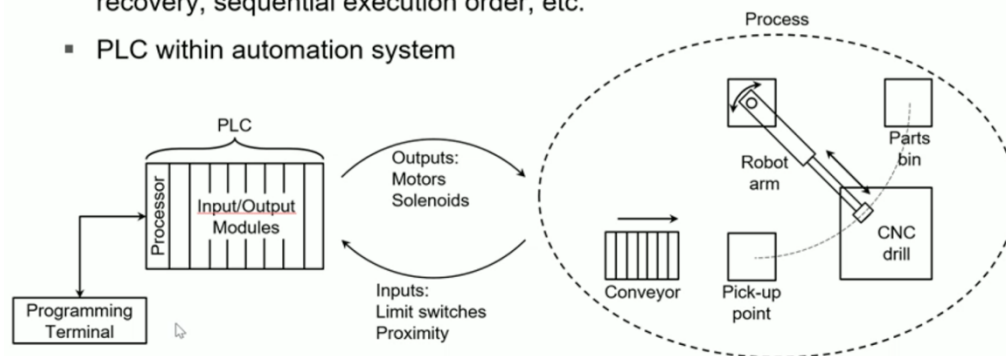
Introduction to PLCs

- **What are PLCs?**

- Programmable Logic Controllers (PLCs) were developed due to the need of reconfiguration in automotive manufacturing, which relied on wiring and [relay-based control systems](#) ([Converting Relays to Programmable Controllers](#)).
- “Workhorses of modern manufacturing automation”



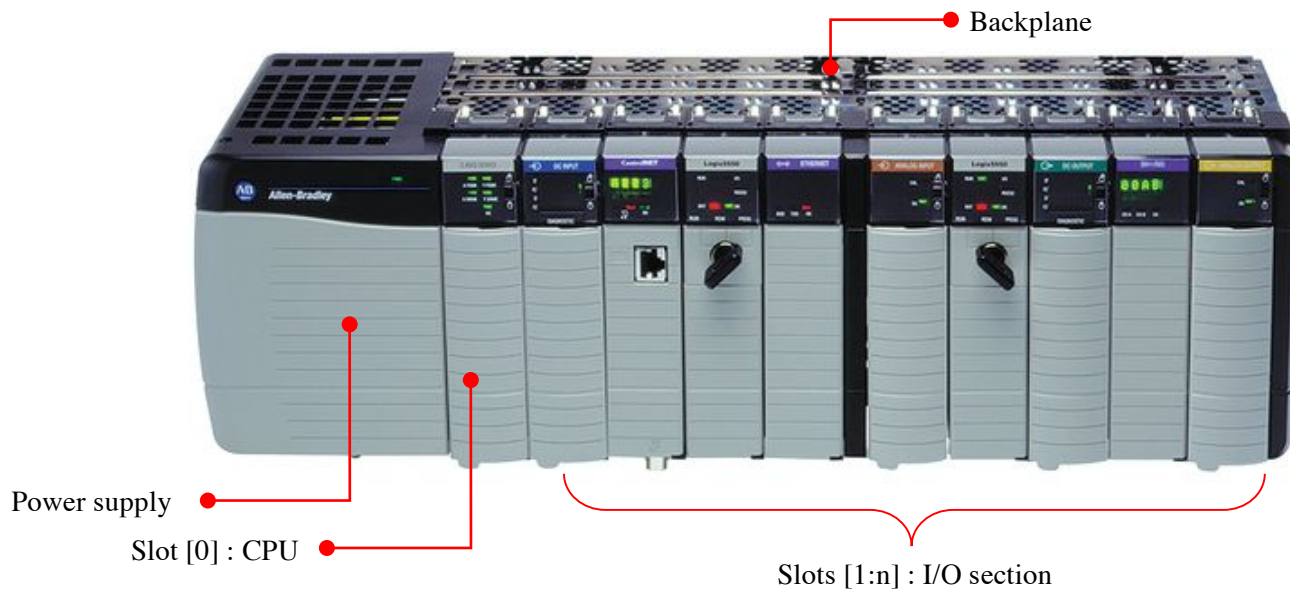
- Initial concept was developed in 1968
- Industrial environment, more reliable, easily maintained, quick power failure recovery, sequential execution order, etc.
- PLC within automation system



- Program PLC to receive signal and produce output
- PLCs are used not only in the automotive industry, but also in petroleum and chemical plants, CNC manufacturing, and more.
- There are many brands of PLCs. They are programmed with languages standardized in [IEC 61131-3](#). One of those languages is ladder logic.

- **PLC components**

- **CPU:** Main control center that evaluates the logic and changes outputs every 5 to 50 ms.
- **Power supply:** Supplies the power for the CPU and I/O sections.
- **Programming interface device:** A computer used to write a ladder program that has the capability to download (send to PLC), upload (read from PLC) or go online and monitor. The application software for programming is Studio 5000.
- **I/O modules:** Work with voltages higher than the CPU can handle. These modules “isolate” the PLC and protect its circuits.

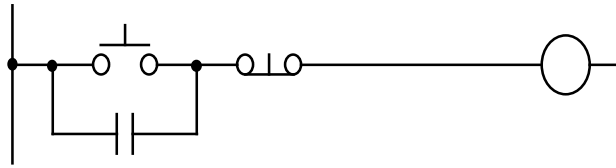


- **Addressing, memory, and scan routine**
 - Addressing tells the PLC what to look for and send out. Addressing involves three parts:
 - Data type (e.g., “I” for input and “O” for output)
 - Module location (e.g., 1 is for slot 1)
 - Specific terminal (e.g., terminal 0)
 Example: “I:1/0” refers to the input from slot 1 and terminal 0.
- **Tags and Data Type**
 - **BOOL**: 1-bit boolean, 0 = cleared 1 = set
 - **SINT**: 1-byte integer, -128 to 127
 - **COUNTER**: later in counter instruction
 - **TIMER**: later in timer instruction
 - Rules:
 - First character can’t be a digit
 - Cannot have more than two consecutive underlines
 - Last character is not an underline
 - Example tag name: RFID_N056:I.Channel[0].Busy
 - RFID_N056: RFID Interface Module (Dual Channel, specified by the Channel[] argument)
 - I: input
 - Busy: BOOL
 - Example tag name: RFID_N056:O.Channel[1].Data[10]
 - O: output; send out signal from PLC to RFID transceiver
 - Data[10]: SINT; 11th tag since RFID starts at 0 (total of 112)

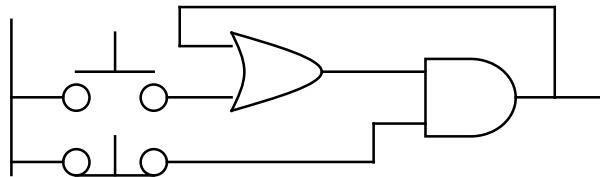
Basic ladder logic programming using Studio 5000Logix Designer (AB from Rockwell)

- **Origins:**
 - First and most popular programming language for PLC (functional block is another one)
 - The Ladder Diagram (LD) programming language originated from the graphical representation used to design an electrical control system

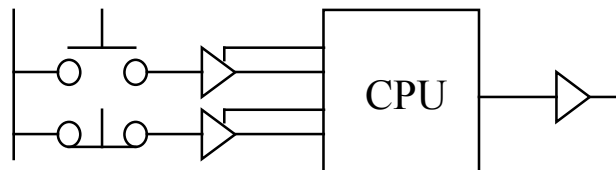
- Control decisions were made using relays



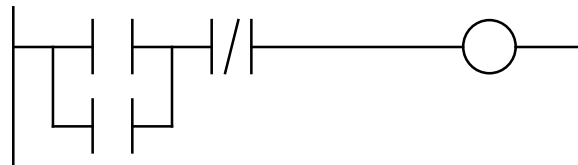
- After a while Relays were replaced by logic circuits
 - [Logic gates](#) used to make control decisions



- Finally, CPUs were added to take over the function of the logic circuits
 - I/O Devices wired to buffer transistors
 - Control decisions accomplished through programming

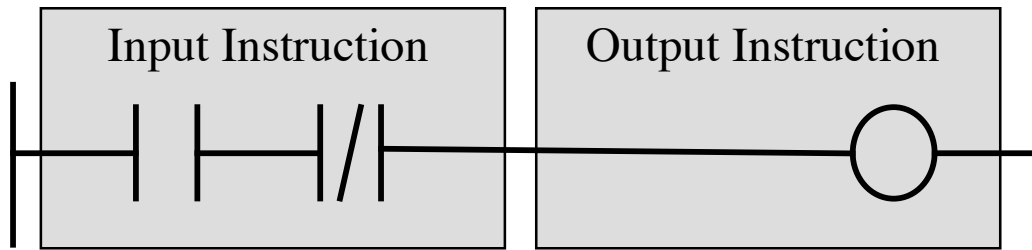


- Relay Logic representation (or LD) was developed to make program creation and maintenance easier
 - Computer based graphical representation of wiring diagrams that was easy to understand
 - Reduced training and support cost



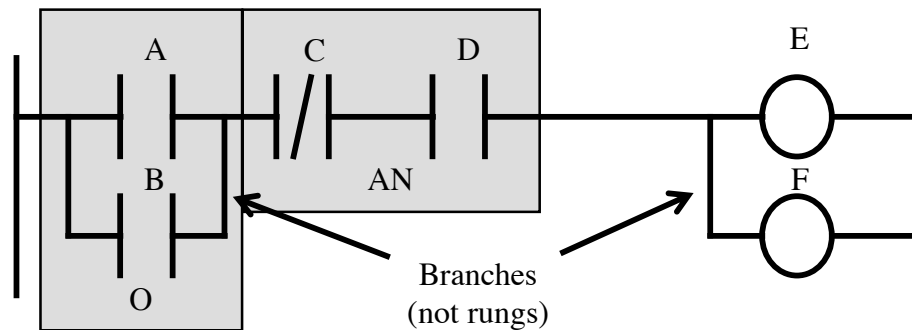
- **What is a rung?**

- A rung of ladder diagram code can contain both input and output instructions
 - Input instructions perform a comparison or test and set the rung state based on the outcome
 - Normally left justified on the rung
 - Output instructions examine the rung state and execute some operation or function
 - In some cases, output instructions can set the rung state
 - Normally right justified on the rung



- **Series vs parallel operations: parallel is better to trigger at the same time, series if sequential**

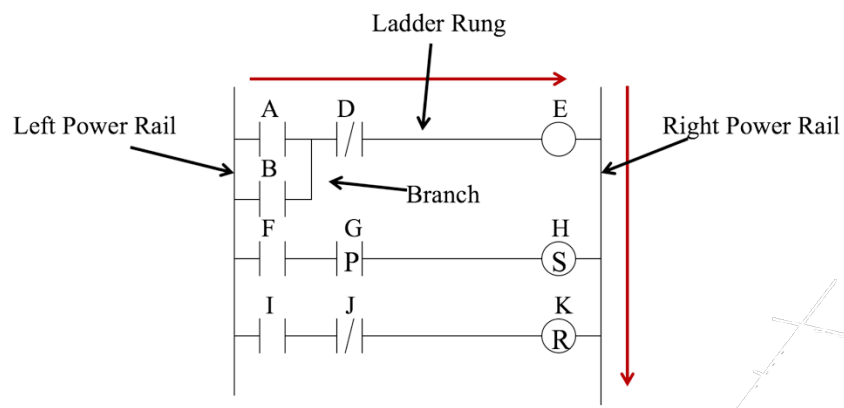
- Ladder Diagram input instructions perform logical AND and OR operations in an easy to understand format
 - If all Input Instructions in series must all be true for outputs to execute (AND)
 - If any input instruction in parallel is true, the outputs will execute (OR)
- Paralleling outputs allows multiple operations to occur based on the same input criteria



IF ((A OR B) AND (NOT C) AND D) THEN E=1;

- **Ladder logic execution**

- Rungs of Ladder diagram are solved from Left to right and top to bottom
- Branches within rungs are solved top left to bottom right

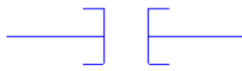



- **Retentive vs non-retentive operations**

- Definitions
 - Retentive values or instructions maintain their last state during a power cycle
 - Non-retentive values or instructions are reset to some default state (usually 0) after a power cycle
- IEC1131 permits values to be defined as retentive

- A contradiction to this is ladder diagram where 3 instructions are classified as retentive
- In most PLCs only timer and coil instructions operate as non-retentive


- **Contacts**

-  **-] [- Examine-if-closed (XIC)/Normally open contact:** Examines a bit for an ON (closed) condition to energize/enable rung. Uses: start push buttons, limit switches, lights, internal bits
-  **-]/[- Examine-if-open (XIO)/Normally closed contact:** Examines a bit for an OFF (open) condition to energize/enable rung. Uses: e-stops, same as XIC
- **-]P[- Positive transition contact or -]ONS[-:** Enables the right side of the rung for one scan when the rung on left side of the instruction is true
- **-]N[- Negative transition contact:** Enables the right side of the rung for one scan when the rung on left side of the instruction is false

- **Retentive coils: the referenced bit is unchanged when processor power is cycled**

- **-(L)- Output latch/set (OTL):** Turns a bit ON when the rung condition is TRUE. Bit retains its state when rung is not executed (does nothing when rung is FALSE)
- **-(U)- Output unlatch/reset (OTU):** Turns a bit OFF when the rung condition is TRUE. Bit retains its state when the rung is not executed (does nothing when rung is FALSE)

- **Non-retentive coils: the referenced bit is reset when processor power is cycled**

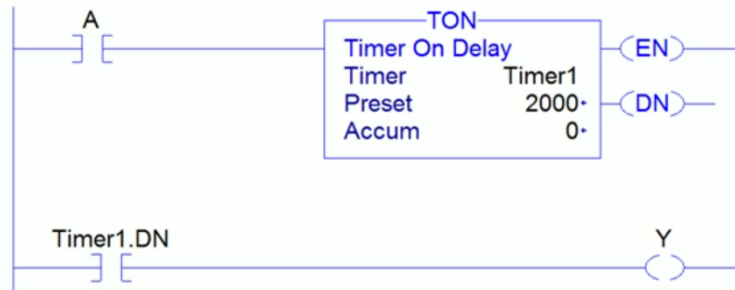
-  **-()- Output enable/energize (OTE):** Turns a bit ON if rung condition is TRUE (rung energized), or OFF if no continuous power on rung. Uses: lights, motor run signal, internal bits
- **-(/)- Negative coil:** Sets a bit when the rung is false(0) and resets the bit when the rung is True(1). Not commonly supported because of potential for confusion

- **Transition Sensing Coils**

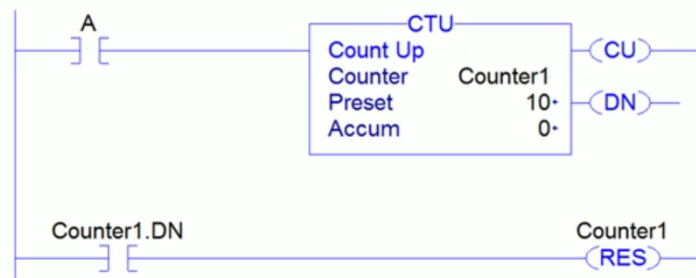
- **-]P[- One-shot rising (OSR)/Positive transition-sensing coil:** Triggers a one-time event. Sets the bit (1) when rung to the left of the instruction transitions from off(0) to on(1). The bit is left in this state.
- **-]N[- One-shot falling (OSF)/Negative transition-sensing coil:** Triggers a one-time event. Resets the bit (0) when rung to the left of the instruction transitions from on(1) to off(0). The bit is left in this state.

- **Timers and counters**

- **TON (“on-delay” timer):** Enabled when rung conditions are true (one can check the enabler with the .EN bit). The “done bit” .DN is set to 1 when the counter finishes its count. When the time input is on, the accumulator is counted up once each millisecond. When the accumulator equals the preset value, the timer is “timed out” and the .DN bit is set to on. If the time input turns off during the timing interval, the accumulator is set to zero and timing recommences when the input turns on
 - Example: once A is on, timer begins until 2 seconds and then triggers; make sure the conditions will stay enabled to actually use the timer

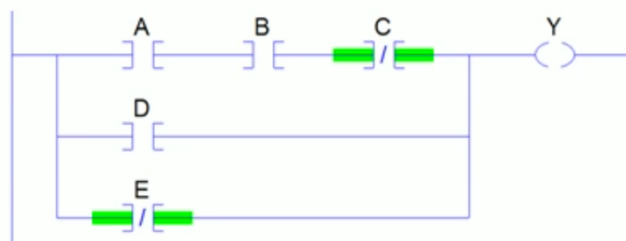


- Bit 13 (“done” bit/“DN”). On when the timer finished its time count
- Bit 14 (“timer-timing” bit/“TT”). On when the timer is initiated and begins timing. Off when the “DN” bit is set
- Bit 15 (“enable” bit/“EN”). On when the timer initiates and begins timing (like “TT”), but stays on until rung conditions preceding the TON instruction transition back to “false”, resetting the “EN” to “0” and the “accumulated”/“ACC” back to its start value
- **TOF (“off-delay” timer):** Enabled when rung conditions are false. The “done bit” **DN** is set to 0 when the counter finishes its count. Only accumulated if condition input is off
- **RTO retentive On-Delay Timer:** no matter if the input is continuous or not, it’ll keep accumulating and can change rapidly on/off
- **CTU (Up Counter) or CTD (Down Counter):** counter accumulator increments/decrements by one for every off-to-on transition of the counter input (NOT MILLISECONDS)
- Use timer if input A is off, not closed anymore, timer refreshes automatically but counter has to be manually set/reset (need condition to reset)
 - Example: once .DN enabled, resets:

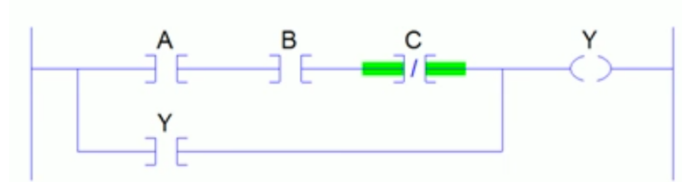


- Multiplication is AND; addition is OR:

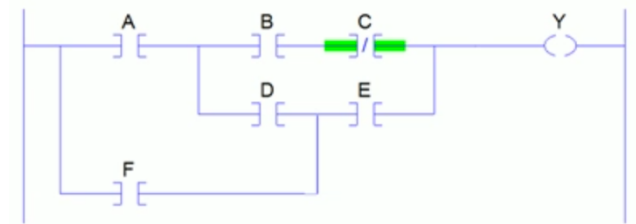
$$Y = ABC\bar{C} + D + \bar{E}$$



- Output coil can also be a contact: if Y is energized, will always be energized (similar to a Latch Coil, but not possible to unlatch it):



- Another (theoretical, not possible) example: can never go back (right to left or bottom rung upwards, only left to right or top rung downwards)



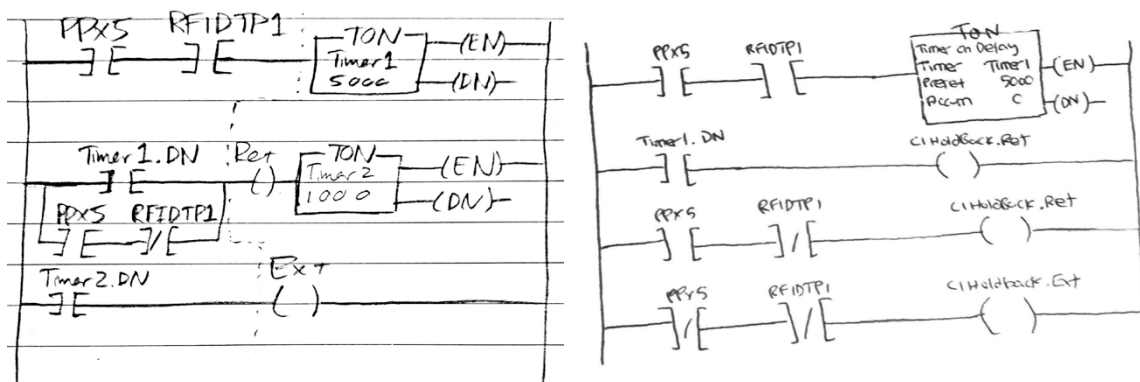
- **DO NOT** repeat normal output coils or latch/unlatch
- Use latch/unlatch coil **TOGETHER**

Examples

• Objective 1

- Try to create a simple program that controls Cell 1 hold back. When an empty pallet comes, retract the hold back and let the pallet go. When a pallet with a part comes, wait for 5 seconds, then retract the hold back and let the pallet go. The hold back should be extended most of the times.
- Useful input/output tags are listed below. You need to create some user-defined tags, for example: a timer, to achieve the objective.
- Input Tags
 - PPX5: The proximity sensor at Cell 1 holdback. Turns on if it is blocked by a pallet; turns off if not.
 - RFIDTP1: RFID transceiver 1 tag present. Turns on if a RFID tag presents; turns off if not.
- Output Tags
 - C1HoldBack.Ret: When this tag is on, Cell 1 hold back will be retracted.
 - C1HoldBack.Ext: When this tag is on, Cell 1 hold back will be extended.

| Conditions (input) | Commands (output) |
|--|---|
| Pallet “PPX5] [“ with no part “RFIDTP1] [“ | Retract “C1HoldBack.Ret ()” |
| Pallet “PPX5] [“ with part “RFIDTP1] [“ | Hold for 5 seconds “TON ‘T1’ 5000” > retract (additional input “T1.DN] [“ for retract output) |
| Otherwise: after 5 seconds of hold back retracted “T1.DN] [“ | Extend “C1HoldBack.Ext ()” |

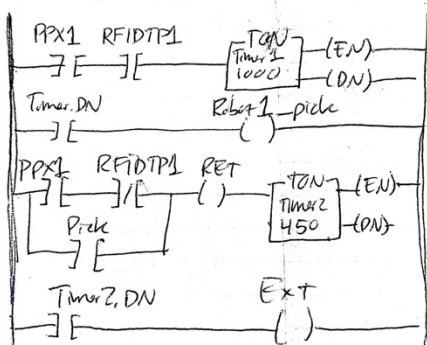


• Objective 2

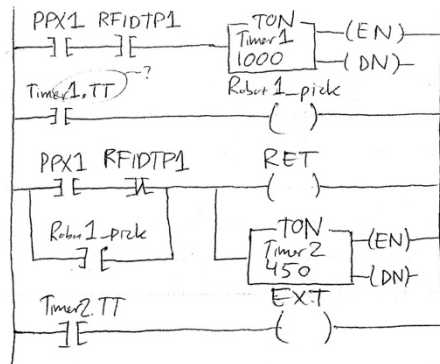
- Try to create a program that controls Cell 1 robot stop. When an empty pallet comes, retract the hold back for 0.45 seconds and let the pallet go. When a pallet with a part on it comes, wait for 1 seconds, then trigger an output called “Robot1_Pick”. This output will trigger the robotic in Cell 1 to pick up a part from conveyor to CNC. After the part has been picked up, the empty pallet should go.
- Useful input/output tags are listed below. You need to create some user-defined tags, for example: a timer, to achieve the objective.
- Input Tags
 - PPX1: The proximity sensor at Cell 1 holdback. Turns on if it is blocked by a pallet; turns off if not.
 - RFIDTP1: RFID transceiver 1 tag present. Turns on if a RFID tag presents; turns off if not.
- Output Tags
 - C1HoldBack.Ret: When this tag is on, Cell 1 hold back will be retracted.
 - C1HoldBack.Ext: When this tag is on, Cell 1 hold back will be extended.
 - Robot1_Pick: This output will trigger the robotic in Cell 1 to pick up a part from conveyor to CNC.

| Conditions (input) | Commands (output) |
|---|---|
| Pallet “PPX1] [“ with no part “RFIDTP1] [“ | Retract “C1HoldBack.Ret ()” for 0.45 seconds “TON ‘T2’ 450” > extend “C1HoldBack.Ext ()” (additional input “T2.DN] [“ for extend output) |
| Pallet “PPX1] [“ with part “RFIDTP1] [“ | Hold for 1 second “TON ‘T1’ 1000” > trigger output “Robot1_Pick” |
| After part picked up “Robot1_pick] [“ | Retract “C1HoldBack.Ret ()” for 0.45 seconds “TON ‘T2’ 450” > extend “C1HoldBack.Ext ()” (additional input “T2.DN] [“ for extend output) |
| After retracting for 0.45 seconds “T2.DN] [“ | Extend “C1HoldBack.Ext ()” |

Version 1:



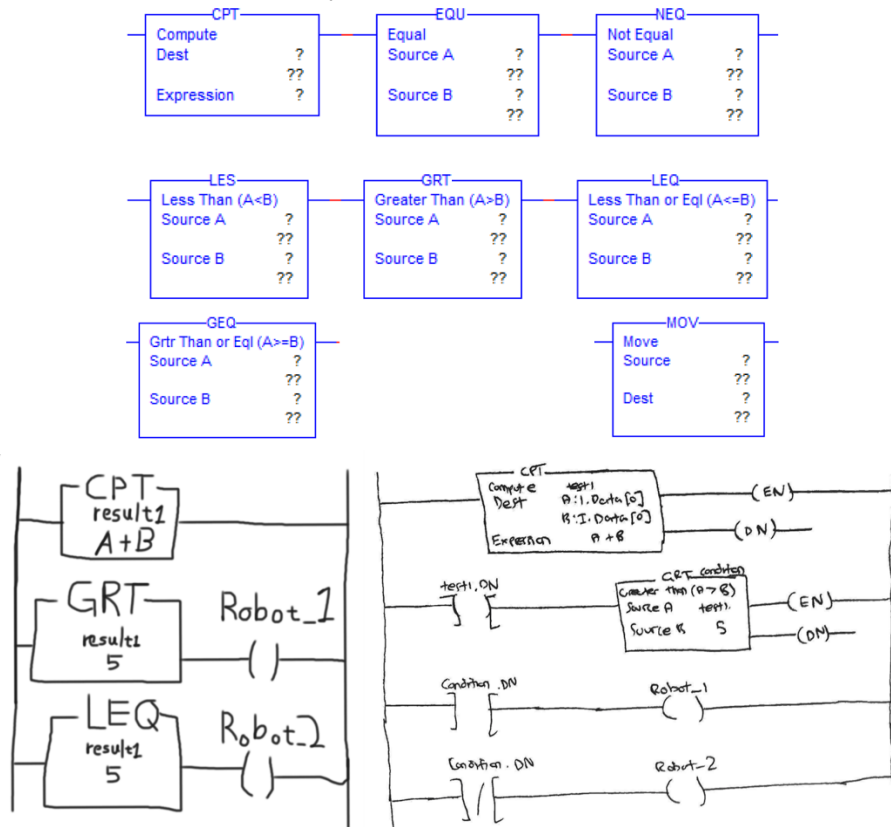
Version 2:



- For timers, use .DN (Version 1) and always put timers and output in parallel instead of in series (Version 2)

• Objective 3

- Create a program that does some simple calculation and comparison. If the sum of A and B is larger than 5, latch Robot_1, else, latch Robot_2.
- Useful input/output tags are listed below. You need to create some user-defined tags to achieve the objective.
- Input Tags
 - A:I.Data[0]: A numeric input. Data Type: DINT.
 - B:I.Data[0]: A numeric input. Data Type: DINT.
- Output Tags
 - Robot_1: Arbitrary Robot movement 1.
 - Robot_2: Arbitrary Robot movement 2.



| Conditions (input) | Commands (output) |
|--|---|
| None | Sum A and B “CPT Dest ‘result1’ Expression ‘A+B’” |
| If sum of A and B greater than 5 “GRT Source A ‘result1’ Source B ‘5’” | Latch “Robot_1 ()” |
| Else: If sum of A and B less than or equal to 5 “LEQ Source A ‘result1’ Source B ‘5’” | Latch “Robot_2 ()” |

- **Objective 4:**

- Think about the logic for a part entering a cell. Write a program controlling two pallet stops, one in front of a cell, and the other one inside the cell. Make sure only one pallet is allowed to go inside the cell at a time. If there is currently no pallet inside the cell, the first pallet stop should let the next coming pallet go. If there is already a pallet inside the cell, the first pallet stop should hold the next coming pallet until the current pallet has left the cell.
- Useful input/output tags are listed below. You need to create some user-defined tags, for example: a timer, to achieve the objective.
- Input Tags
 - PPX5: The proximity sensor at Cell 1 holdback (the pallet stop in front of Cell 1). Turns on if it is blocked by a pallet; turns off if not.
 - PPX1: The proximity sensor at Cell 1 robot stop (the pallet stop inside Cell 1). Turns on if it is blocked by a pallet; turns off if not.
- Output Tags:
 - C1HoldBack.Ret: When this tag is on, Cell 1 hold back will be retracted.
 - C1HoldBack.Ext: When this tag is on, Cell 1 hold back will be extended.
 - C1RobotStop.Ret: When this tag is on, Cell 1 robot stop will be retracted.
 - C1RobotStop.Ext: When this tag is on, Cell 1 robot stop will be extended.
- User defined:
 - Robot_busy: Need additional condition for if a pallet has already been released/is moving to cell 1 but not there yet; turns on if hold back releases a pallet and turns off when the robot releases the pallet from the cell
 - Robot_done: Additional input for when robot done with part to release pallet from cell

| Conditions (input) | Commands (output) |
|--|--|
| If no pallet inside cell and no pallet in transit to cell: robot not busy “Robot_busy]/[” | Retract hold back “C1HoldBack.Ret ()” for 1 second “TON ‘T1’ 1000” > latch Robot_busy |
| If 1 second passed after pallet released from hold back: “T1.DN] [” | Latch tag for Robot_busy (L) |
| If pallet inside cell or pallet in transit to cell: robot busy “Robot_busy] [” | Extend hold back “C1HoldBack.Ext ()” |
| If robot done with part “Robot_done] [“ | Retract robot stop “C1RobotStop.Ret ()” for 1 second “TON ‘T2’ 1000” > extend “C1RobotStop.Ext ()” (additional input “T2.DN] [” for extend output) |
| If robot stop retracted for 1 second “T2.DN] [” | Extend robot stop “C1RobotStop.Ext ()” and unlatch tag for “Robot_busy (U)” |

REFERENCES:

- Kelvin T. Erickson - *Programmable Logic Controllers: An Emphasis on Design and Application*
-