SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

INFORMATION SYSTEMS TECHNOLOGY AND DESIGN

51.508 – Secure Cyber Physical Systems

Week 2 – PLC Programming

Created by **Jianying Zhou** (2018)

Last updated: 12 Jan 2019

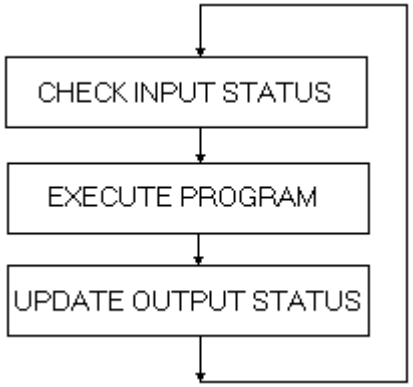
Learning Objectives

- Understand basic programmable logic controller (PLC) languages – ladder diagram (LD) and structured text (ST).
- Able to program and test PLCs.

What is PLC ?

- **PLC = Programmable Logic Controller**, first introduced in 1960's.
- A device that was invented to replace the necessary sequential relay circuits for machine control.
- Look at its inputs and depend upon their state, turn on/off its outputs.
- Continually scan a program.
- **PLCs available in iTrust lab:** Allen-Bradley, National Instruments, Schneider Electric, Siemens.

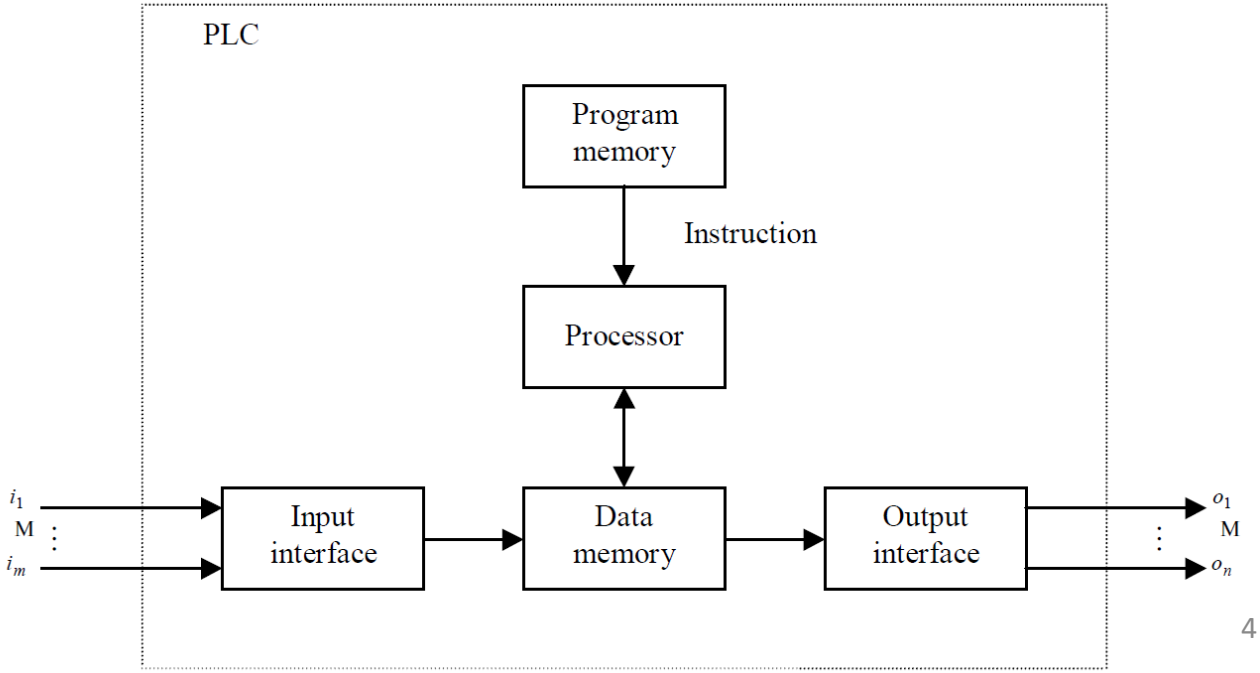
Typical PLCs



Discussion: Give more examples of PLC manufacturers.

PLC Hardware Architecture

- PLC is a miniaturized version of a general-purpose computer.
- Get **input signals** from the system of processes to be controlled, and store in PLC's **data memory**.
- Processor executes instructions from PLC's **program memory**.
- The **output interface** communicates the final result back to the system of processes under the PLC's control.



PLC vs RTU

- **RTU = Remote Terminal Unit**
- The functions of RTUs and PLCs overlap – Performs on-site control and transfers the data to the central SCADA system.
- RTUs are more suitable for wider geographical telemetry, and can withstand harsh conditions, for example, oil and gas pipelines.
- PLCs are more suitable for local control, for example, assembly lines in a factory.

Part 1: PLC Programming Languages

PLC Programming Languages – IEC61131-3

Defined in IEC 61131-3 (1996):

1. **Ladder Diagram (LD)** – graphical, based on Relay Ladder Logic
2. **Structured Text (ST)** – textual, resembling Pascal/C
3. Function Block Diagram (FBD) – graphical, resembling electronic circuit
4. Sequential Function Chart (SFC) – graphical, derived from Petri Nets
5. Instruction List (IL) – textual, resembling assembler

Discussion:

- What PLC languages are you familiar with?
- Why would you prefer those PLC languages?

PLC Programming Languages – LD

- **Ladder Diagram (LD):**
  1. Invented in the U.S. decades ago.
  2. Used in probably 95% of all applications in U.S.
  3. Resembles a series of control circuits, with a series of **inputs** needing to be "made" or "true" in order to activate one or more **outputs**.
- **Pros:**
  1. Easy to start writing a program in Ladder Diagram, especially for a non-programmer with an electrical background.
  2. Allow a program to be organized into folders or subprograms that are downloaded to the PLC, allowing for easy segmentation.
- **Cons:**
  1. Ideal for a simple material handling application (with timers , or some basic comparisons or math), but there are no complex functions involved.
  2. As program size grows, the ladder can become very difficult to read and interpret.

PLC Programming Languages – ST

- **Structured Text (ST):**
  1. A textual programming language that uses **statements** to define what to execute.
  2. Closely resembles a high level computer programming language such as Pascal/C.
  3. Best embraces the growing complexity of PLC programming, such as the process control functions.
  4. Seen the greatest increase in adoption of all the IEC61131-defined programming languages.
- **Pros:**
  1. Decision loops and pointers (variables used to do indirect addressing) allow for a more compact program implementation.
  2. Easy to insert comments throughout a program, and to use indents and line spacing to emphasize related sections of code.
  3. **Runs much faster than Ladder.**
- **Cons:**
  1. Unsuitable for troubleshooting for non-programmer with an electrical background.
- **LD + ST:**
  1. Use Structured Text "behind the scenes".
  2. Encapsulate a Structured Text program inside an instruction called on in Ladder.

PLC Programming Languages – FBD

- **Function Block Diagram (FBD):**
  1. The second most widely used PLC programming language.
  2. A graphical language which resembles a wiring diagram even more so than Ladder code.
  3. The blocks are “wired” together into a sequence that’s easy to follow.
- **Pros:**
  1. Easy to follow – just follow the path!
  2. Ideal for simpler programs consisting of digital inputs and outputs.
- **Cons:**
  1. Not ideal for large programs using special I/O and functions.
  2. The large amount of screen space required by this style of programming can quickly make a program unwieldy.
  3. More difficult to make corrections of the codes later.

PLC Programming Languages – SFC

- **Sequential Function Chart (SFC):**
  1. Resembles the computer flowcharts.
  2. An initial step “action box” is followed by a series of transitions and additional action steps.
  3. Code inside an action box can be written in any language of the programmer’s choice.
- **Pros:**
  1. Ideal for applications which have a repeatable multi-step process or series of repeatable processes.
  2. Very friendly to maintenance engineers for troubleshooting.
- **Cons:**
  1. The structure that is forced on a program could add unneeded complexity.
  2. The overhead required for this type of program causes it to **execute slower than the other languages.**
  3. **The inability to convert to other languages.**

PLC Programming Languages – IL

- **Instruction List (IL):**
  1. Consists of many lines of code, with each line representing exactly one operation.
  2. Very popular in Europe.
- **Pros:**
  1. A program written in this language can be moved easily between hardware platforms.
  2. A low level language and executes much faster in the PLC than a graphical language.
  3. Much more compact and consumes less space in PLC memory.
- **Cons:**
  1. Hard to enter complex functions.
  2. Not in any form of structured programming, limiting its usefulness for implementing large programs.

PLC Programming Languages – comparison

- **Choosing an appropriate language**

Requirement	Choices
Ease of maintenance by the final user	SFC
Universal acceptance of language	LD
Speed of execution by the PLC	ST or IL
Applications mainly using digital I/O and basic processing	LD or FBD
Ease of changing code later	LD
Ease of use by engineers with IT background	ST
Ease of implementing complex mathematical operations	ST
Applications with repeating processes or processes requiring interlocks and concurrent operations	SFC

- **In this course, we choose LD + ST.**

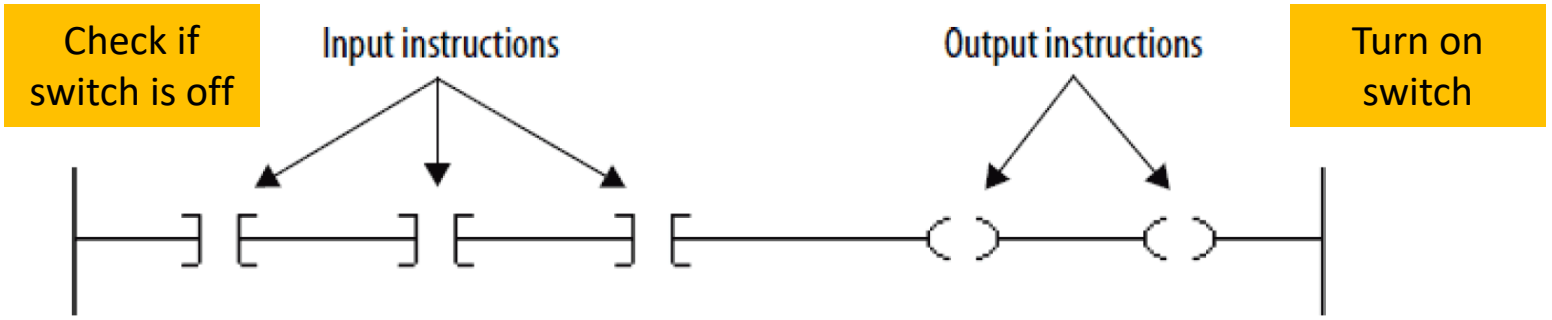
Part 1: PLC Programming Languages

Ladder Diagram

Ladder Diagram – instruction

Instruction

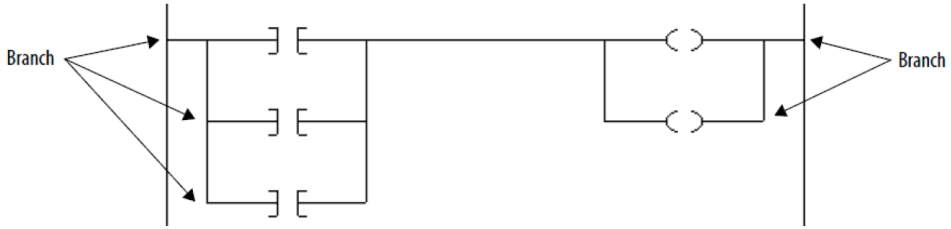
- Ladder Diagram (LD) is organized as rungs on a ladder and put instructions on each rung.
- Two basic types of instructions:
  - **Input instruction**: An instruction that *checks*, *compares*, or *examines* specific conditions in your machine or process.
  - **Output instruction**: An instruction that *takes* some action, such as turn on/off a device, copy data, or calculate a value.



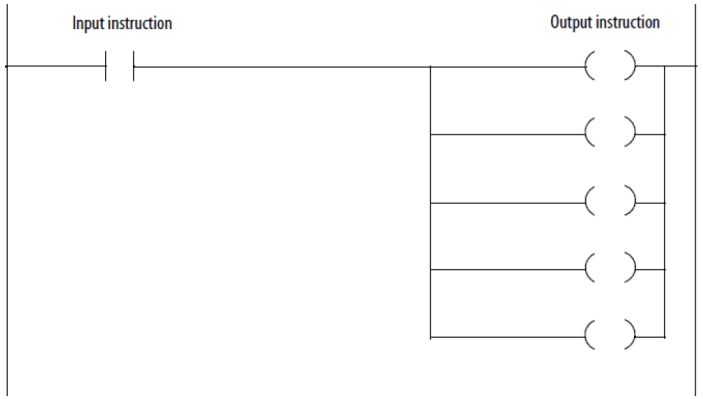
Ladder Diagram – branch

Branch

- Two or more instructions in parallel.



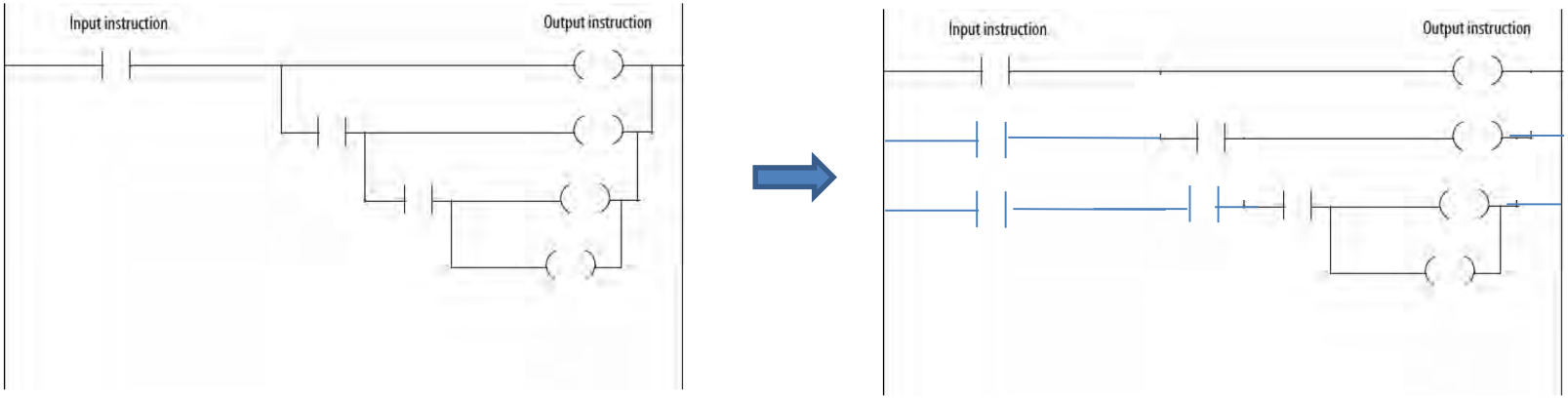
- There is *no limit* to the number of parallel branch levels that you can enter.



Ladder Diagram – branch

Branch

- You can nest branches to as many as *6 levels*.
- To make it easier to maintain, divide the logic into *multiple smaller* rungs.

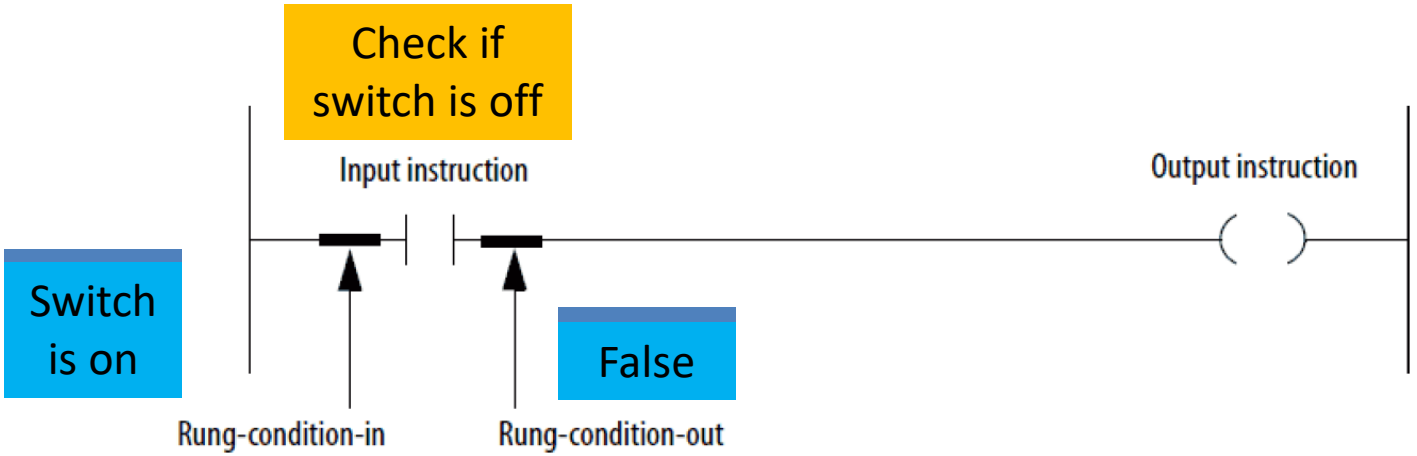


Discussion: Further divide the logic into smaller rungs.

Ladder Diagram – rung condition

Rung condition

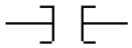
- The controller *evaluates* ladder instructions based on the rung condition preceding the instruction (rung-condition-in).
- If the rung-condition-in to an input instruction is *true*, the controller evaluates the instruction and sets the rung-condition-out to *match the results* of the evaluation.



Ladder Diagram – ladder logic

Write ladder logic

- Identify the *conditions* to check, and separate them from the *action* to take for the rung.
- Choose the appropriate input instruction for each *condition* and the appropriate output instruction for each *action*.

Symbol	Name	Mnemonic	Description
	Examine If Closed	XIC	An input instruction that looks at one bit of data.

If the bit is	Then the instruction (rung-condition-out) is
On (1)	True
Off (0)	False

Symbol	Name	Mnemonic	Description
	Output Energize	OTE	An output instruction that controls one bit of data.

If the instructions to the left (rung-condition-in) are	Then the instruction turns the bit
True	On (1)
False	Off (0)

Switch is off

True

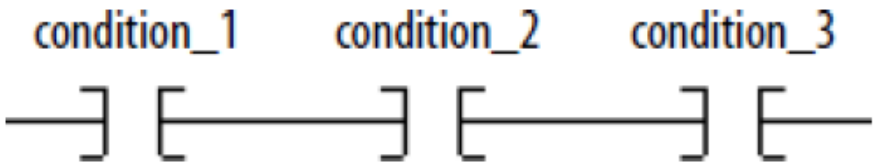
True

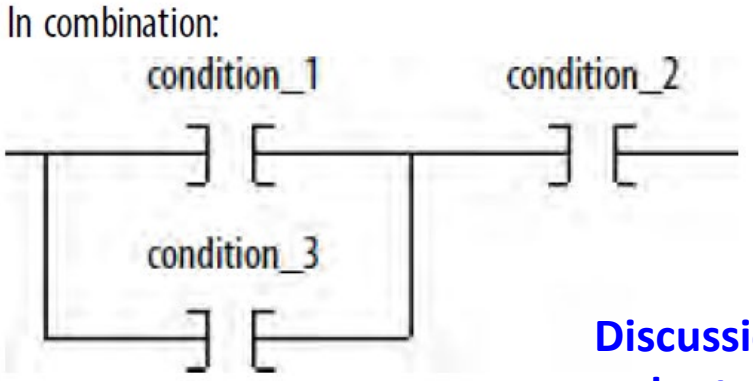
Switch is on

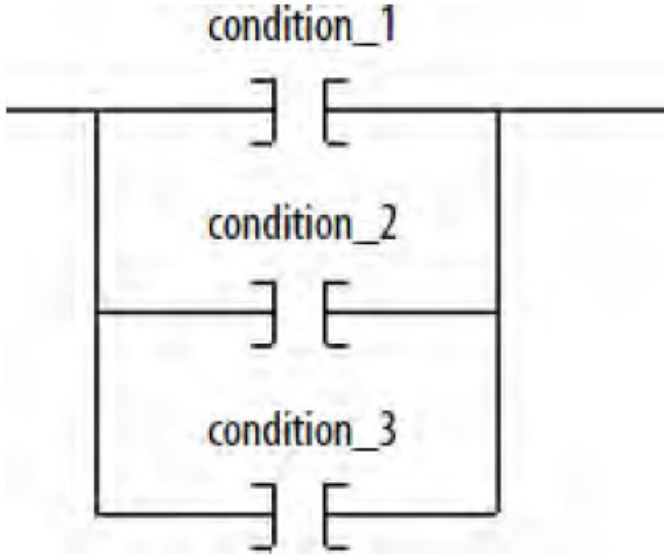
Ladder Diagram – input instruction

Arrange input instructions

- In series. *All conditions* must be met in order to take action.
- In Parallel. *Any one* of several conditions must be met in order to take action.





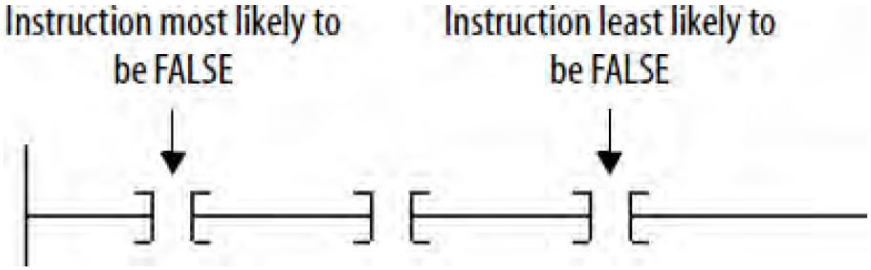


Discussion: What conditions should be met in order to take action in this hybrid mode?

Ladder Diagram – input instruction

Arrange input instructions

- The controller executes all instructions on a rung regardless of their rung-condition-in.
- For optimal performance of a series of instructions, sequence the instructions from most likely to be false on the left to least likely to be false on the right.
- When the controller finds *a false instruction*, it executes the remaining instructions in the series with their *rung-condition-in* set to *false*.
- Typically, an instruction executes *faster* when *its rung-condition-in is false* rather than true.

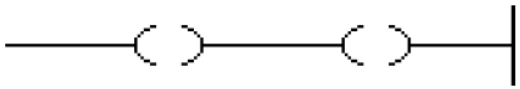


Ladder Diagram – output instruction

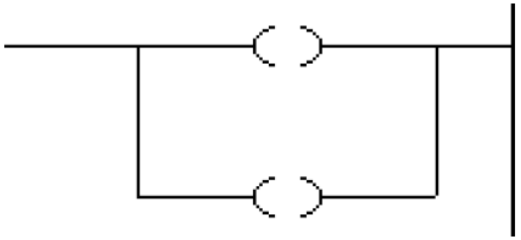
Arrange output instructions

- Place *at least one* output instruction to the right of the input instructions.
- You can enter *multiple output* instructions on a rung of logic.

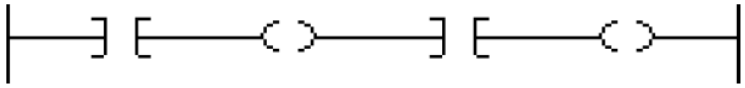
Place the output instructions in sequence on the rung (serial).



Place the output instructions in branches (parallel).



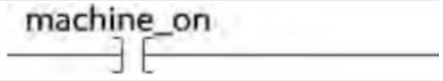
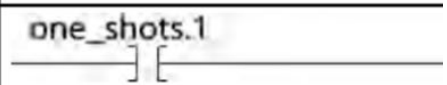
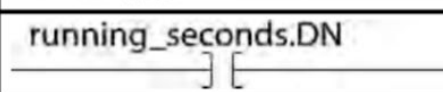
Place the output instructions between input instructions. *The last instruction on the rung must be an output instruction.*



Ladder Diagram – tag name

Choose a tag name

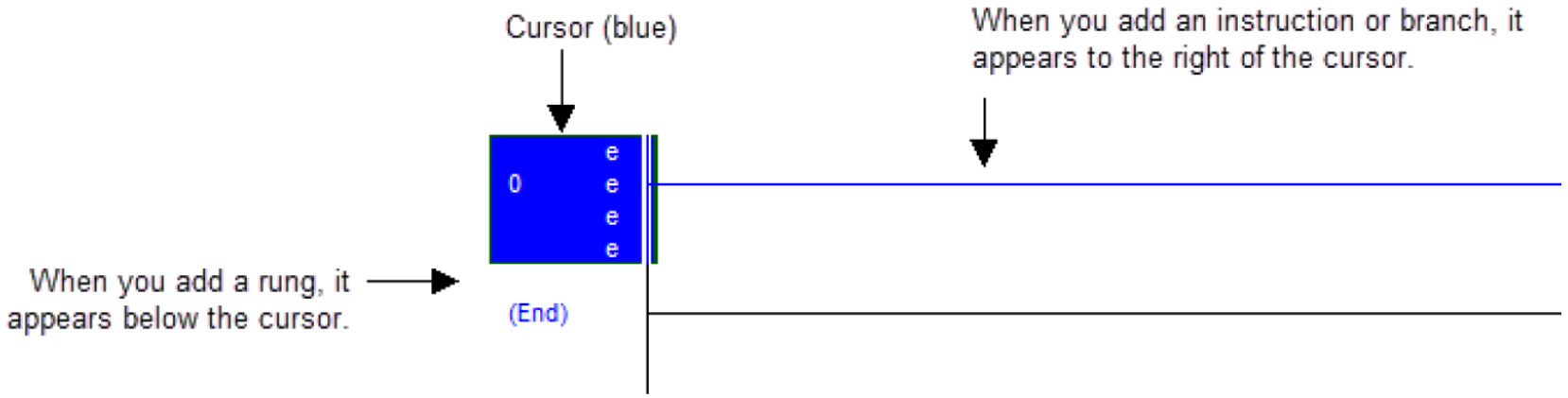
For a:	Specify:
Tag	*tag_name*
Bit number of a larger data type	*tag_name.bit_number*
Member of a structure	*tag_name.member_name*
Element of a one dimension array	*tag_name[x]*
Element of a two dimension array	*tag_name[x,y]*
Element of a three dimension array	*tag_name[x,y,z]*
Element of an array within a structure	*tag_name.member_name[x]*
Member of an element of an array	*tag_name[x,y,z].member_name*

To Access:	The tag name looks like this:
machine_on tag	
bit number 1 of the one_shots tag	
DN member (bit) of the running_seconds timer	

- *x* is the location of the element in the first dimension.
- *y* is the location of the element in the second dimension.
- *z* is the location of the element in the third dimension.

Ladder Diagram – language element

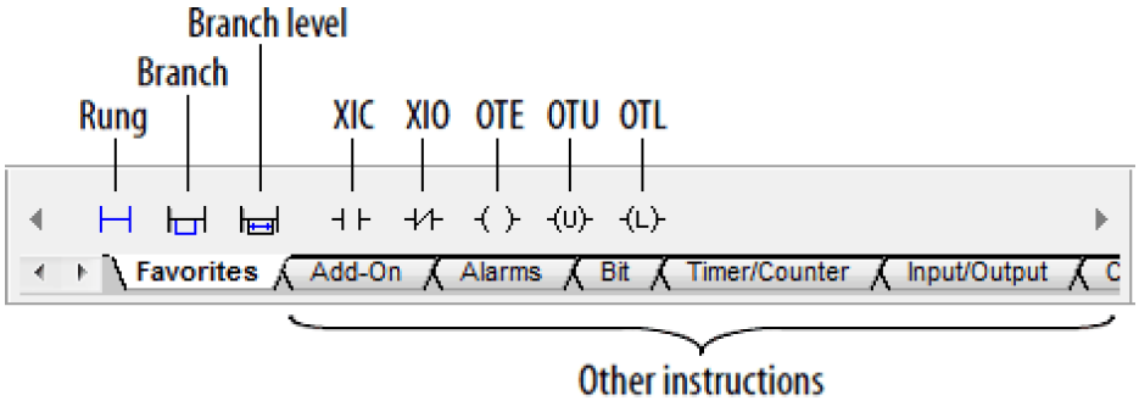
Enter ladder logic (Will use Allen-Bradley Studio 5000 in iTrust lab.)



Use **Language Element** toolbar to add a ladder diagram element.

Input:
- XIC - Examine If Closed
- XIO - Examine If Open

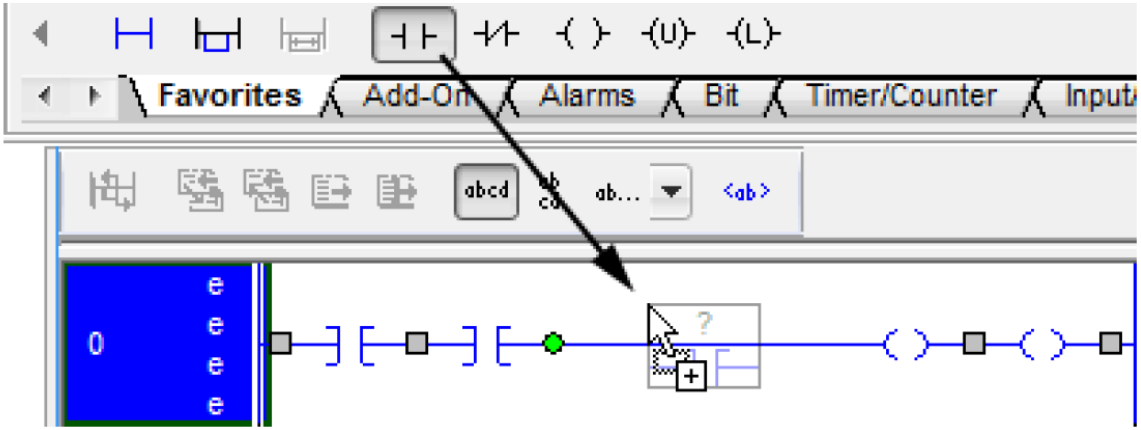
Output:
- OTE - Output Energize [set a bit]
- OTU - Output Unlatch [clear bit (retentive)]
- OTL - Output Latch [set a bit (retentive)]



Ladder Diagram – language element

Enter ladder logic

- Click to select the instruction, branch, or rung that is above or to the left where you want to add an element.
- On the Language Element toolbar, click the button for the element that **you want to add.**
- Drag the button for the element directly to the desired location. A **green dot** shows a valid placement location (drop point).

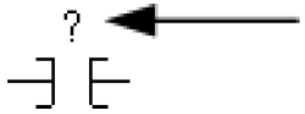


Ladder Diagram – instruction operands

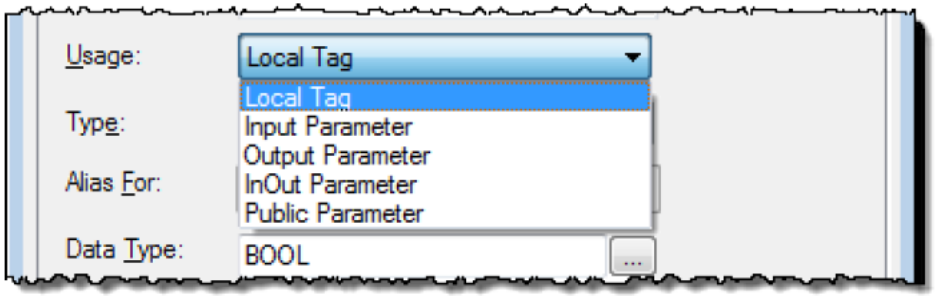
Assign instruction operands

- After adding an instruction to a ladder rung, assign tags to the instruction operands.
- You can create a new tag, use an existing tag, or assign a constant value.

Create and assign a new tag



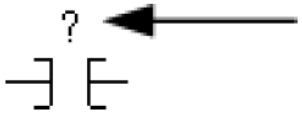
1. Click the operand area of the instruction.
2. Type a name for the tag and press **Enter** key.
3. Right-click the tag name and then click **New "tag_name"**.
4. In **New Parameter or Tag** dialog box, choose **Usage**, **Type**, and **Data Type** for the tag.



Ladder Diagram – instruction operands

Assign instruction operands

Choose a name or an existing tag



1. Double-click the operand area of the instruction, and then click  .
2. The Tag Browser window appears, then select name or tag.

To select a:	Do this:
Label, routine name, or similar type of name	Click the name.
Tag	Double-click the tag name.
Bit number	A. Click the tag name. B. To the right of the tag name, click  . C. Click the required bit.

Ladder Diagram – instruction operands

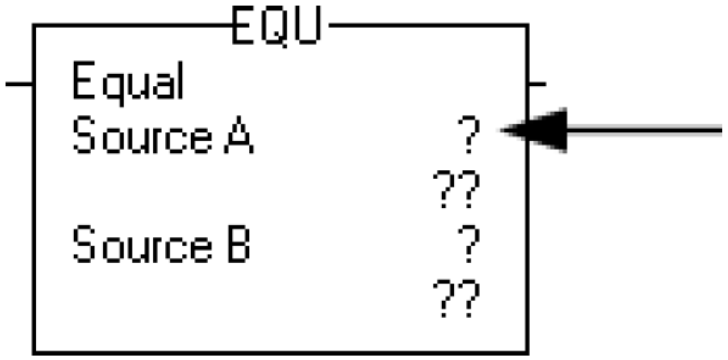
Assign instruction operands

Drag and drop a tag from the Tags window

1. Find the tag in **Controller Tags** or **Program Parameters and Local Tags window**.
2. Double-click the tag to select it.
3. Click and drag the tag to its location on the instruction. A green dot appears to show you where you can drop the tag.

Assign an immediate (constant) value

1. Click the operand area of the instruction.
2. Type the value and press **Enter** key.

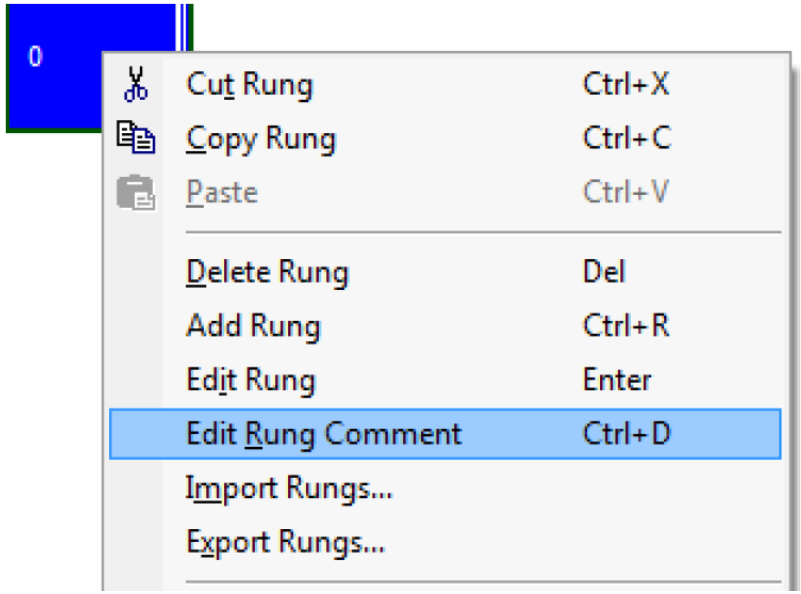


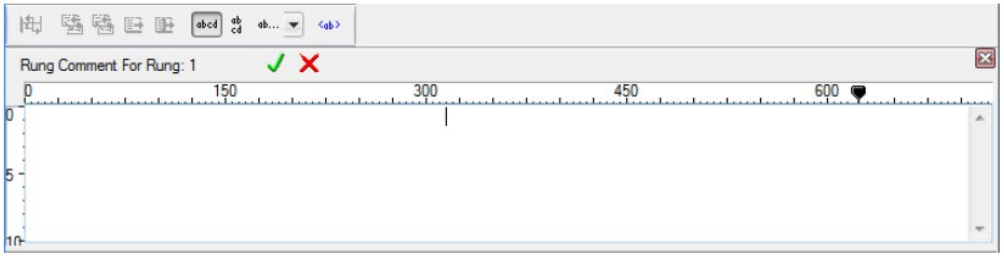
Ladder Diagram – rung comment

Enter a rung comment

When entering a rung of ladder logic, you can add comments that explain the purpose of your rung.

1. Right-click the rung number of your ladder logic and then **click Edit Rung Comment**.
2. The **Rung Comment** dialog box appears.
3. Type your rung comment, and then click the **green check** to save your changes, or click the **red X** to discard your changes.





Ladder Diagram – verify routine

Verify the routine

As you program your routine (s), it is a good idea to **periodically verify** your work.

1. In **Standard** toolbar click  .
2. Errors are listed in **Output** window on **Errors** tab at the bottom of the application.
   - To go to the first error or warning, press **F4** key.
   - Correct the error according to the description in **Errors** tab.
   - Repeat until you have corrected all errors.
3. To close **Output** window, press **Alt+1** keys.

To be practiced at the next session in the lab.

Ladder Diagram – programming example

Example – PLC Program with On-Delay Timer

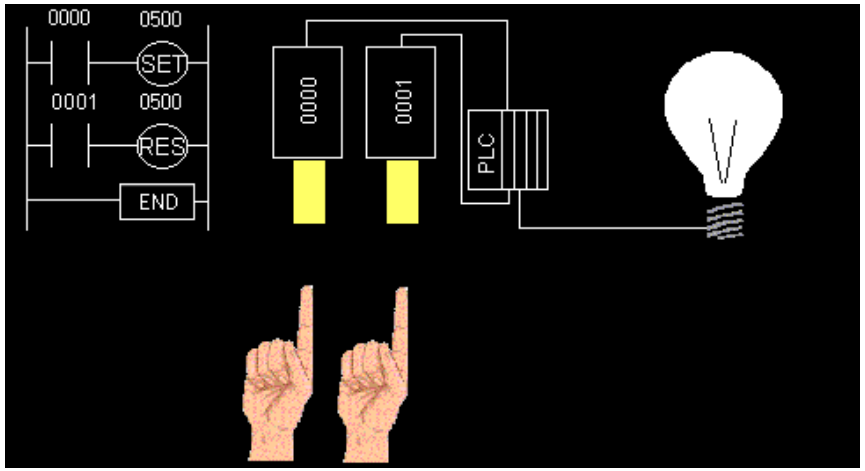
- As soon as the on-delay timer gets a signal at the input, the timer starts to count down. When the preset time is up, the output of the on-delay timer will turn on. If the input is turned off before the count down finishes, the time will reset.
- **Demo:** https://youtu.be/jyOcwPbCoKw
- The **enable output (EN)** is the first output and it is on when the timer is energized. So, as long as the input is true or on, the enable output will be true.
- Second output is the **done output (DN)**. This output in an on-delay timer is only on, when the timer has counted down the preset time.

We use Ladder Diagram to encapsulate Structured Text programs.

PLC Control Animation – latch

Latch Instructions

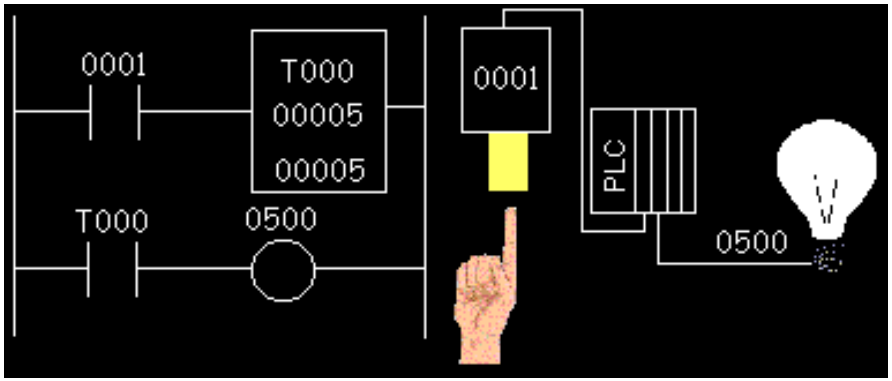
- Latch: SET or OTL (output latch) – Pushes button (0000), light on
- Unlatch: RES (reset) or OTU (output unlatch) – Pushes button (0001), light off



PLC Control Animation – timer

Timer On-Delay (TON) – delays turning on

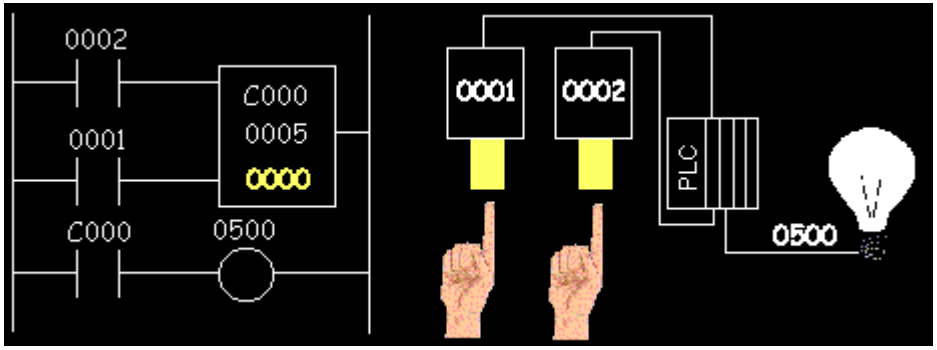
- When the operator pushes button (0001), the timer starts ticking.
- When the accumulated value reaches 0, the timer contacts (T000) will close and output 0500 becomes true.
- When the operator releases button (0001), the accumulated value changes back to the preset value (i.e. it resets).



PLC Control Animation – counter

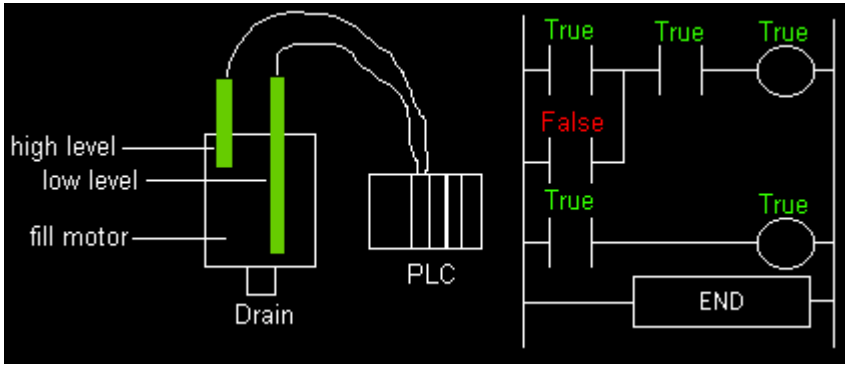
Up-Counters (CTU) – count up

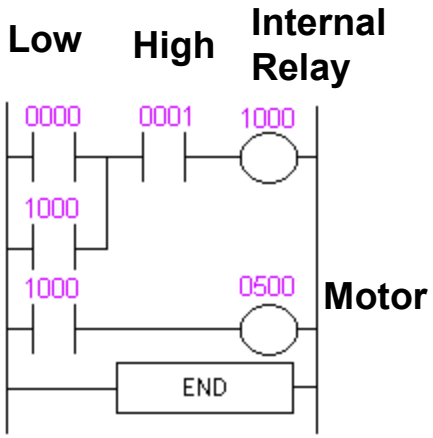
- Each time the operator pushes button (0001), the counter accumulated value increases by one. Note that it increases only when the button first turns on. (i.e. the off to on transition).
- When the accumulated value equals the preset value (i.e. 5), the C000 contacts turn on and output 0500 becomes true.
- When the operator pushes the reset button (0002), the accumulated value changes back to 0000.

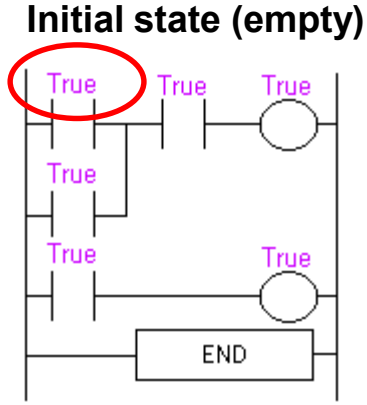


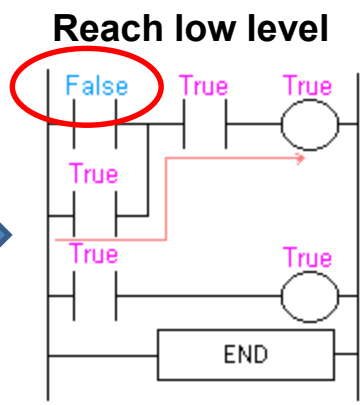
PLC Control Animation – tank

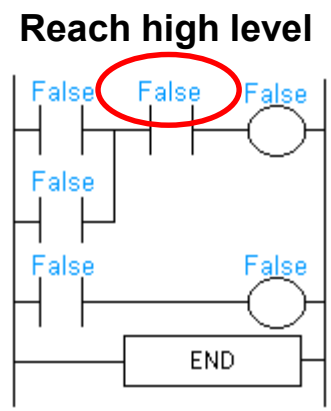
- Fill motor pumps oil into the tank until the reaching high level.
- At that point, turn off the motor until the level falls below the low level.

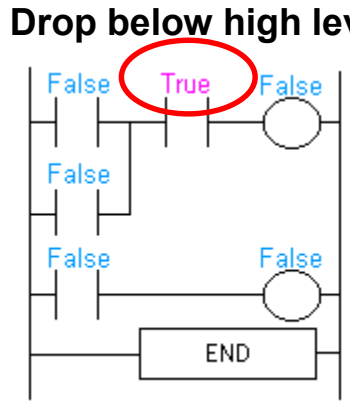












Part 1: PLC Programming Languages

Structured Text

Structured Text – syntax

Syntax

Structured text is a *textual* programming language that uses *statements* to define what to execute.

- Structured text is *not case sensitive*.
- Use *tabs* and *carriage returns* (separate lines) to make your structured text easier to read.

Structured Text – comments

Comments

- Comments let you use *plain language* to describe how structured text works.
- Comments *do not affect the execution* of the structured text.

Format	Example

//comment	**At the beginning of a line**
	`//Check conveyor belt direction`
	`IF conveyor_direction THEN...`
	At the end of a line
	`ELSE //If conveyor isn't moving, set alarm light`
	`light := 1;`
	`END_IF;`

(*comment*)	`Sugar.Inlet[:=]1;(*open the inlet*)`
	`IF Sugar.Low (*low level LS*)& Sugar.High (*high level LS*)THEN...`
	`(*Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.*)`
	`IF tank.temp > 200 THEN...`

/*comment*/	`Sugar.Inlet:=0;/*close the inlet*/`
	`IF bar_code=65 /*A*/ THEN...`
	`/*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/`
	`SIZE(Inventory,0,Inventory_Items);`

Structured Text – assignments

Assignments

- Use an assignment to change the value stored within a tag.
- Syntax: **tag := expression;**
- The ***tag*** retains the assigned value until another assignment changes the value.
- The ***expression*** can be simple, such as an immediate value or another tag name, or can be complex and includes several operators and functions.

Structured Text – expressions

Expressions

- An expression can be a tag name, equation, or comparison:
  - Tag name that stores the value (variable)
  - Number that you enter directly into the expression (immediate value)
  - String literal that you enter directly into the expression
  - Functions, such as: ABS, TRUNC
  - Operators, such as: +, -, <, >, And, Or

- When writing expressions, follow these **general rules**:
  - Use any combination of upper-case and lower-case letter. For example, these three variations of "AND" are acceptable: AND, And, and.
  - For more complex requirements, use parentheses to group expressions within expressions. This makes the whole expression easier to read and ensures that the expression executes in the desired sequence.

Structured Text – expressions

Expressions

- **BOOL expression:** Produces either the BOOL value of 1 (true) or 0 (false).
  - A bool expression uses bool tags, relational operators, and logical operators to compare values or check if conditions are true or false. For example, tag1 > 65.
  - A simple bool expression can be a single BOOL tag.
  - Typically, you use bool expressions to condition the execution of other logic.

- **Numeric expression:** Calculates an integer or floating-point value.
  - A numeric expression uses arithmetic operators, arithmetic functions, and bitwise operators. For example, tag1 + 5.
  - May nest a numeric expression within a BOOL expression. For example, (tag1 + 5) > 65.

- **String expression:** Represents a string
  - A simple expression can be a string literal or a string tag.

Structured Text – operators & functions

Operators and Functions

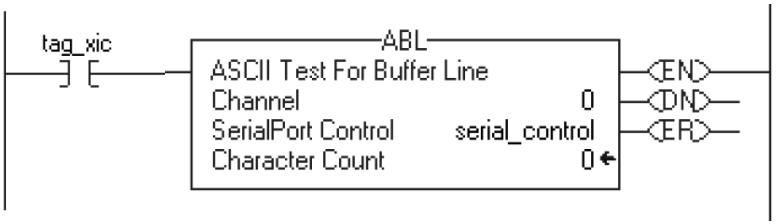
- Operators calculate new values:
  - Arithmetic operators: **+**, **-**, *, ** (exponent), **/**, **MOD**
  - Relational operators: **=**, **<** , **<=**, **>**, **>=**, **<>**
  - Logical operators, yielding a Boolean value of TRUE or FALSE: **AND**, **OR**, **XOR**, **NOT**
  - Bitwise operators, for each bit of two numbers: **AND**, **OR**, **XOR**, **NOT**

- Functions perform math operations: **LOG**, **SQRT**, …

Structured Text – instructions

Instructions

- In structured text, instructions will execute each time they are scanned unless you pre-condition the execution of the structured text instruction.
- This differs from relay ladder instructions that use rung-condition-in to trigger execution.



Case 1: The ABL instruction will execute every scan if tag_xic is set, not just when tag_xic transitions from cleared to set.

```
IF tag_xic THEN ABL(0,serial_control);
END_IF;
```

Case 2: If you want the ABL instruction to execute only when tag_xic transitions from cleared to set, you have to condition the structured text instruction. **Use one-shot rising with input (OSRI) to trigger execution.**

```
osri_1.InputBit := tag_xic;
OSRI(osri_1);
IF (osri_1.OutputBit) THEN
ABL(0,serial_control);
END_IF;
```

Structured Text – constructs

Constructs

If you want to	Use this construct
Do something if or when specific conditions occur	IF. . . THEN
Select what to do based on a numerical value	CASE. . . OF
Do something a specific number of times before doing anything else	FOR. . . DO
Keep doing something as long as certain conditions are true	WHILE. . . DO
Keep doing something until a condition is true	REPEAT. . . UNTIL

Structured Text – construct (IF…THEN)

Construct:

IF…THEN: to do something if or when specific conditions occur.

```
IF bool_expression1 THEN
   <statement>;
     .
     .
ELSEIF bool_expression2 THEN
 <statement>;
     .
     .
ELSE
 <statement>;
     .
     .
END_IF;
```

optional

optional

Statements to execute when *bool_expression1* is true.

Statements to execute when *bool_expression2* is true.

Statements to execute when both expressions are false.

If you want this	Enter this structured text
If conveyor direction contact = forward (1) then	IF conveyor_direction THEN
light = off	light := 0;
Otherwise light = on	ELSE
	light [:=] 1;
	END_IF;

Structured Text – construct (CASE…OF)

Construct:

CASE…OF: to select what to do based on a numerical value.

- Executes only the statements that are associated with **the first matching selector value**.
- Execution always breaks after the statements of that selector and goes to the END_CASE statement.

```
CASE numeric_expression OF
    selector1: <statement>;
    .
    .
   selector2: <statement>;
    .
    .
ELSE
                <statement>;
    .
    .
END_CASE;
```

optional

Specify as many alternative selector values as you need.

If you want this	Enter this structured text
If recipe number = 1 then Ingredient A outlet 1 = open (1) Ingredient B outlet 4 = open (1)	CASE recipe_number OF
	1: Ingredient_A.Outlet_1 :=1; Ingredient_B.Outlet_4 :=1;
If recipe number = 2 or 3 then Ingredient A outlet 4 = open (1) Ingredient B outlet 2 = open (1)	2,3: Ingredient_A.Outlet_4 :=1; Ingredient_B.Outlet_2 :=1;

Structured Text – construct (FOR…DO)

Construct:

FOR…DO: to do something a specific number of times before doing anything else.

optional

```
FOR count := initial value
    TO final_value
    BY increment
    DO
        <statement>;
        .
        .
END_FOR;
```

If you don't specify an increment, the loop increments by 1.

If you want this	Enter this structured text

Clear bits 0…31 in an array of BOOLs:
Initialize the subscript tag to 0.
Clear i . For example, when subscript = 5, clear array[5].
Add 1 to subscript.
If subscript is ≤ to 31, repeat 2 and 3.
Otherwise, stop.
	For subscript:=0 to 31 by 1 do
	array[subscript] := 0;
	End_for;

Structured Text – construct (WHILE…DO)

Construct:

WHILE…DO: to keep doing something as long as certain conditions are true.

```
WHILE bool_expression DO
   <statement>;
    .
    .
END_WHILE;
```

Statements to execute while *bool_expression* is true.

If you want this	Enter this structured text	
The WHILE…DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop. This differs from the REPEAT…UNTIL loop because the REPEAT…UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. The statements in a REPEAT…UNTIL loop are always executed at least once. The statements in a WHILE…DO loop might never be executed.	pos := 0;	
	While ((pos <= 100) & structarray[pos].value <> targetvalue)) do	
		pos := pos + 2;
		String_tag.DATA[pos] := SINT_array[pos];
	end_while;	

Structured Text – construct (REPEAT…UNTIL)

Construct:

REPEAT…UNTIL: to keep doing something until conditions are true.

```
REPEAT
   <statement>;
    .
    .
UNTIL bool_expression
END_REPEAT;
```

Statements to execute while *bool_expression* is false.

If you want this	Enter this structured text

The REPEAT…UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. This differs from the WHILE…DO loop because the WHILE…DO The WHILE…DO loop evaluates its conditions first.
If the conditions are true, the controller then executes the statements within the loop. The statements in a REPEAT…UNTIL loop are always executed at least once. The statements in a WHILE…DO loop might never be executed.
	pos := -1;
	REPEAT
	pos := pos + 2;
	UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))
	end_repeat;

Structured Text – exercise

Constructs:

- Give the example codes using the construct of **CASE…OF**.
- Rewrite using the construct of **IF…THEN** which has the same effect.
- Explain the difference between **CASE…OF** and **IF…THEN**.

Structured Text – exercise

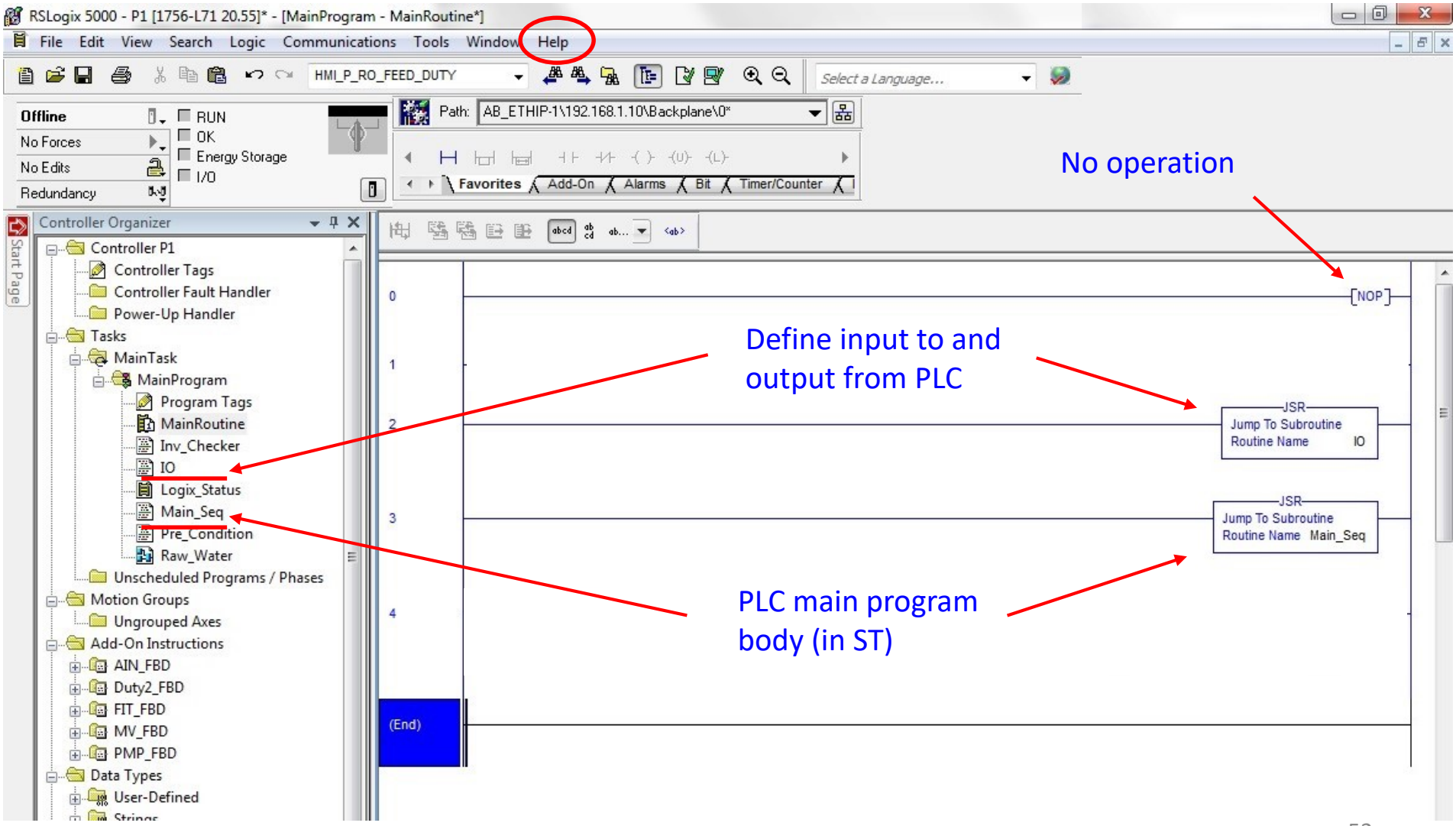
```
FUNCTION Ave_REAL : REAL
        VAR_INPUT
                Input1, Input2 : REAL;
        END_VAR
        Ave_REAL := (Input1 + Input2) / 2;
END_FUNCTION

Average1 := Ave_REAL(4.0, 6.0);
Average2 := Ave_REAL(Input2 := 6.0);
```

What are the values of Average1 and Average2?

Average1 = 5.0

Average2 = 3.0 (default value of Input1 = 0)

PLC Programming – example (LD + ST)



Summary – Part 1

1. Introduction of basic PLC programing languages, especially about ladder diagram (LD) and structured text (ST).
2. You are expected to be able to use LD and ST to program PLCs.

3. Will continue the topic on PLC programming in the next session (Security Tools Lab), using iTrust PLC training skid.
   - **Time:** Tue, 12 Feb (Group 1-5); Fri,15 Feb (Group 6-10)
   - **Venue:** iTrust lab @ Building 2, Level 7
   - Please bring your own laptop.

4. Week 3: Modeling CPS (Aditya)
   - **Time:** Tue, 19 Feb

Reading Materials – Part 1

1. Ted Thayer. “Understanding the IEC 61131-3 Programming Languages”. Control Engineering, 2009.
2. PLC Programming & Automation Online. “Ladder Logic Examples and PLC Programming Examples” – http://www.plcacademy.com/ladder-logic-examples/
3. Info PLC. “Structured Text Programming” – http://www.infoplc.net/files/descargas/rockwell/infoplc_net_plc_st.pdf

Part 2: PLC Training Skids & Lab Exercise

PLC Training Skids in iTrust Lab

Four different types of PLC training skids are available in iTrust Lab:

1. **Allen-Bradley – Rockwell Automation**
2. National Instruments
3. Schneider Electric
4. Siemens

PLC Training Skid – NI

- **NI cRIO-9063**, an embedded real-time controller with reconfigurable FPGA
- Runs on NI Linux Real-Time OS
- Remote digital/analogue I/Os
- A secure router and wireless client
- Milliamp meter, potentiometer, key switches, light push buttons, ultra mini photo sensor

PLC Training Skid – Schneider

- **Modicon Quantum CPU**, a special-purpose computing system with digital processing capabilities
- Remote digital/analogue I/Os
- A secure router and wireless client
- Milliamp meter, potentiometer, key switches, light push buttons, ultra mini photo sensor

PLC Training Skid – Siemens

- **SIMATIC S7-1500 controller**, compatible with Siemens Step 7 and TIA software
- Remote digital/analogue I/Os
- A secure router and wireless client
- Milliamp meter, potentiometer, key switches, light push buttons, ultra mini photo sensor

PLC Training Skid – Allen Bradley

PLC Training Skid – AB (HW & SW)

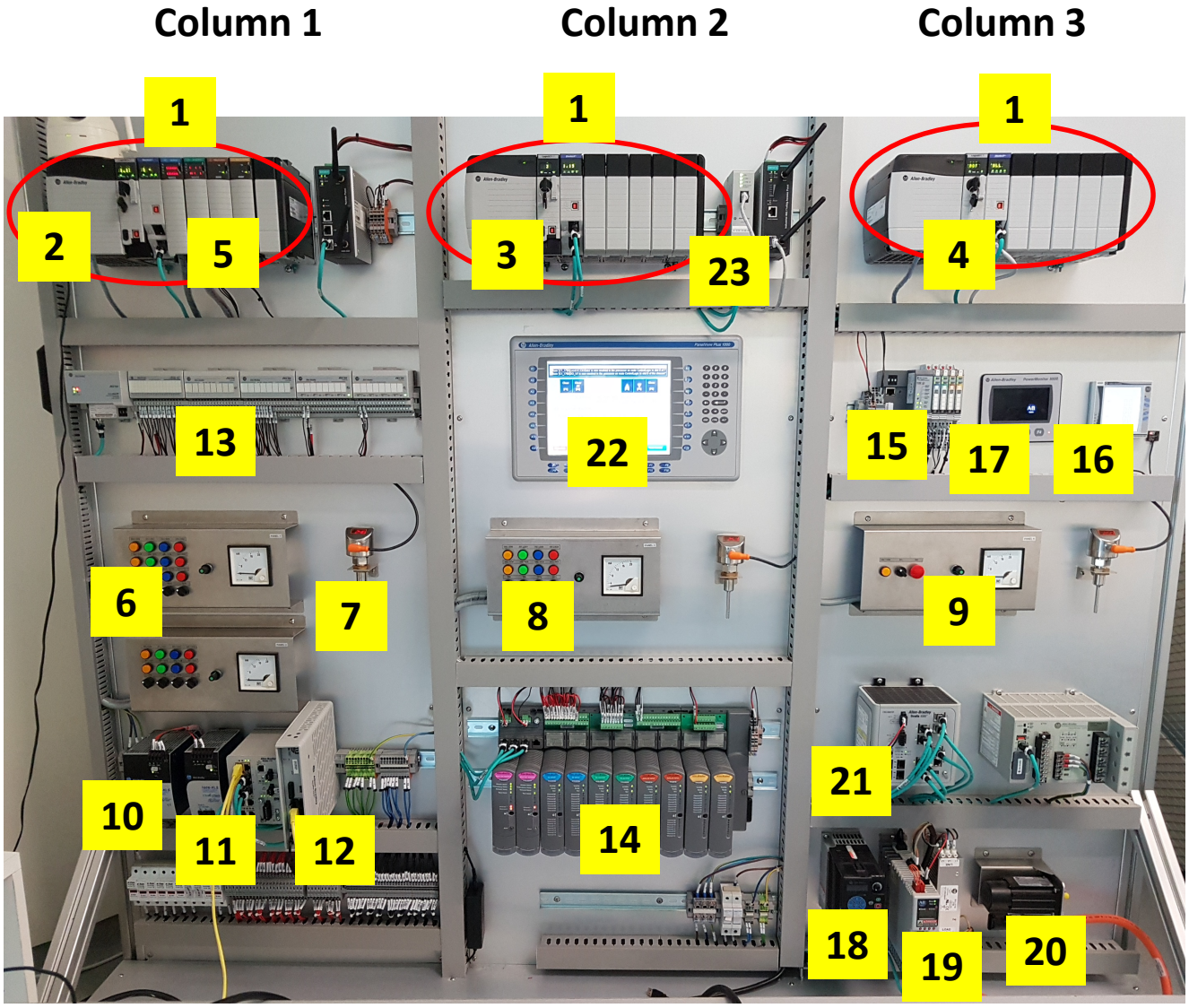
Hardware (3 separate columns of Allen Bradley PLCs)

- AC power supply
- CPU, communication processor with digital & analogue I/Os
- A set of control instruments – temperature probe, analogue ammeter, potentiometer and illuminated push buttons.

Software

- Studio 5000

PLC Training Skid – AB (components)



1. PLC setup rack
2. AC power unit
3. CPU
4. Communication processor
5. Analog and digital I/O
6. Illuminated push buttons
7. Temperature probe
8. Potentiometer
9. Analog ammeter
10. Power supply
11. Ethernet switch
12. Service router
13. Flex I/O
14. Redundant I/O
15. Point I/O
16. Power meter
17. Power quality meter
18. Variable speed drive
19. Servo drive
20. Servo motor
21. 10 ports switch
22. Panel view plus terminal
23. Wireless access point

PLC Training Skid – AB (components)

1. **PLC setup rack** = A layout of PLC with Power supply, CPU, communication processor and optional I/O module.
2. **AC power unit** = Power up PLC with required direct current.
3. **CPU** = Central processing unit, which carries out instructions with the program setup.
4. **Communication processor** = A small sized computers for interfacing with networks (connecting to remote I/O: No. 13, 14, 15).
5. **Analog and digital I/O** = Input to and output from PLC.
6. **Illuminated push buttons** (DI/DO)
7. **Temperature probes** (AI) = Temperature measuring device.
8. **Potentiometer** (AI) = A resistive instrument to increase or decrease resistivity.
9. **Analog ammeter** (AO) = A meter showing the readings of the measure current.
10. **Power supply** = Power supply to instruments which converts AC to DC.
11. **Ethernet switch** = Network access
12. **Service router** = Connects PLCs
13. **Flex I/O** (remote) = Offers flexibility for applications with digital, analog and specialty I/O.

PLC Training Skid – AB (components)

14. **Redundant I/O** (remote) = Has fault tolerance and redundancy for critical process applications.
15. **Point I/O** (remote) = For applications requiring flexibility and low cost of ownership.
16. **Power meter** = An instrument that measures the power through the lines.
17. **Power quality meter** = An instrument that measure the power quality by analyzing the sag, dip, swell, interruptions and impulses in a power system.
18. **Variable speed drive** = A type of adjustable-speed drive used in electro-mechanical drive systems to control AC motor speed and torque by varying motor input frequency and voltage.
19. **Kinetic 5500 servo drive** = A drive that controls servo and induction motors via a single cable. Optimization of power and energy.
20. **Servo motor**
21. **10 ports switch** = A hub that receives, processes data to destination device.
22. **Panel view plus terminal** = Human Machine Interface visually through touch screen.
23. **Wireless access point** = Converts hard-wired access to wireless (providing wireless connection between PLCs, or with remote I/O).

Connect to AB Training Skid

1. Connect to WiFi network: **Singtel8800(5G)-ABC5**. It will connect you to **10.0.1.96-98** network.

2. Remote Desktop (RDP) into VM: **10.0.1.91**. It will connect you to Allen Bradley training skid.

3. Then, you can start using **Studio 5000,** and download/upload program to the training skid.

Connect to AB Training Skid – WiFi

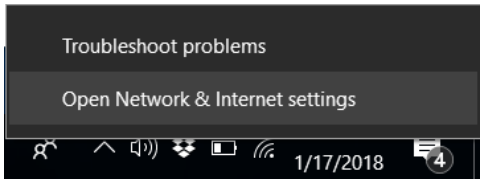
1. Connect to WiFi network: **Singtel8800(5G)-ABC5**. It will connect you to **10.0.1.96-98** network.

   Password: 0001289157

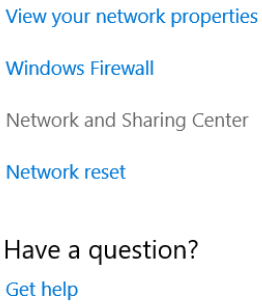
Connect to AB Training Skid – VM

2. Remote Desktop (RDP) into VM: **10.0.1.91**. It will connect you to Allen Bradley training skid.

   a) Right click on the Wifi icon

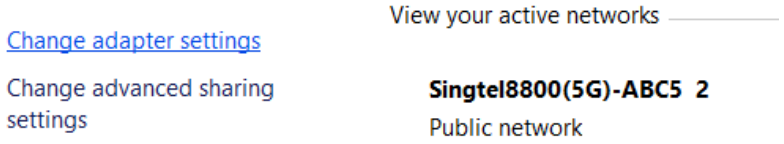


   b) Select “Open Network & Internet settings”



   c) Click on “Network and Sharing Center”

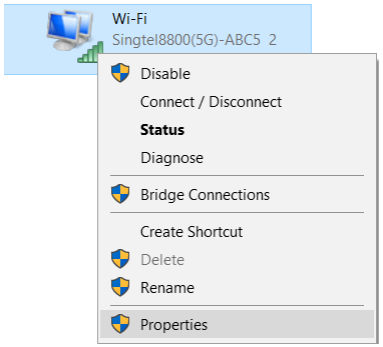
   d) Click on “Change adapter settings”



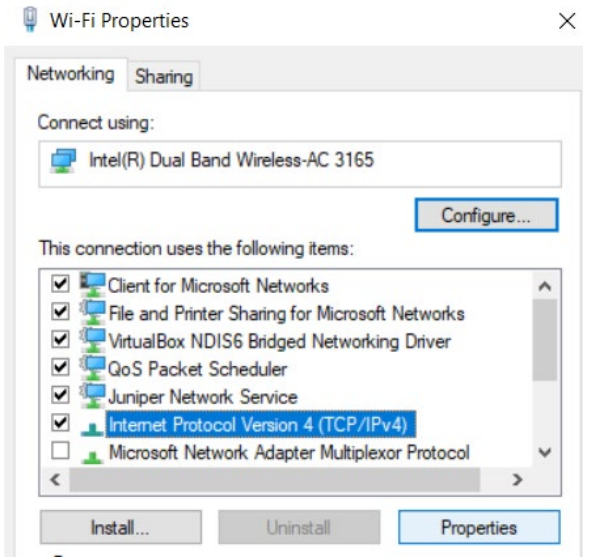
Connect to AB Training Skid – VM

2. Remote Desktop (RDP) into VM: **10.0.1.91**. It will connect you to Allen Bradley training skid.

e) Right click on the Wi-Fi connection and click on “Properties”



f) Select “Internet Protocol Version 4 (TCP/IPv4)”



g) Click on “Properties”

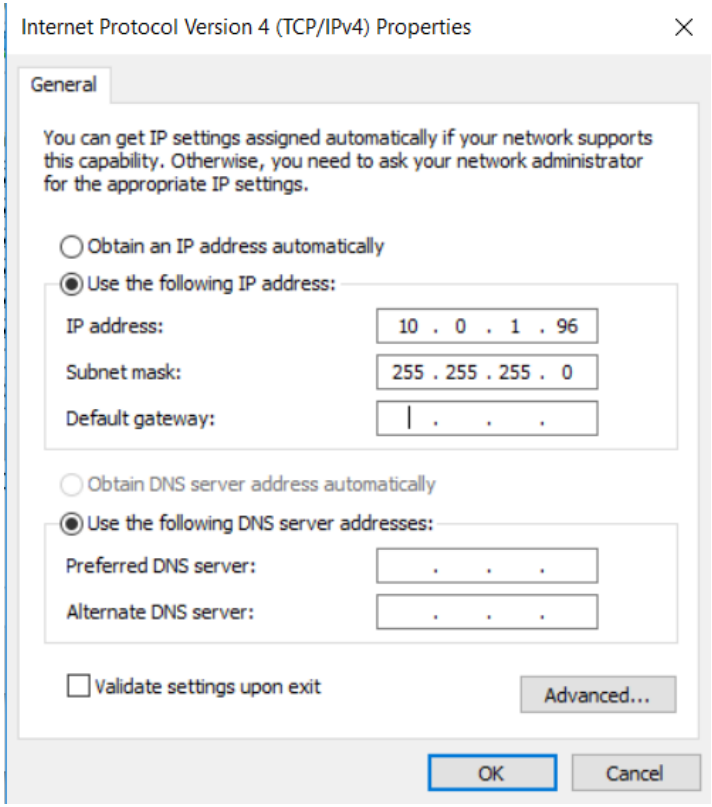
Connect to AB Training Skid – VM

2. Remote Desktop (RDP) into VM: **10.0.1.91**. It will connect you to Allen Bradley training skid.

   h) Set your IP address (**10.0.1.96-98**) and Subnet mask as listed and click OK

   i) Click “OK”

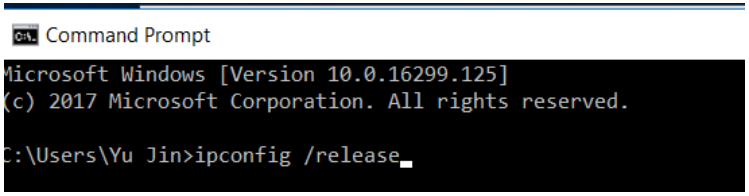
   j) Click “Close”



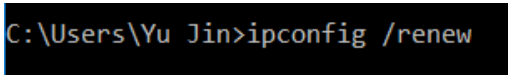
Connect to AB Training Skid – VM

2. Remote Desktop (RDP) into VM: **10.0.1.91**. It will connect you to Allen Bradley training skid.

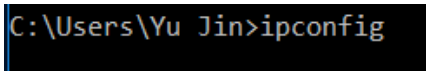
   h) Go to the command prompt and type “ipconfig /release”

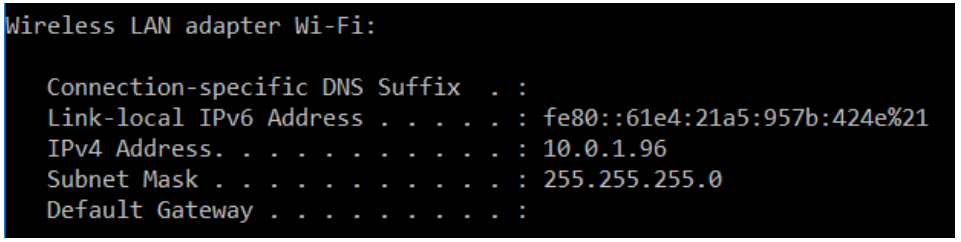


   i) Type “ipconfig /renew”



   j) Type” ipconfig” and check if your ip address has changed

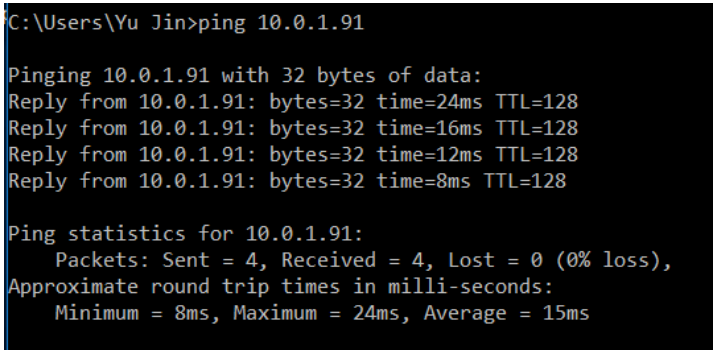


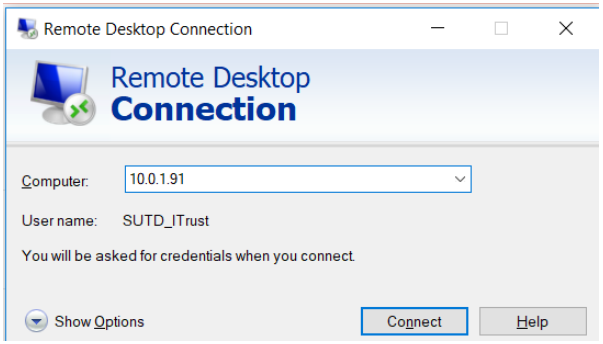


Connect to AB Training Skid – VM

2. Remote Desktop (RDP) into VM: **10.0.1.91**. It will connect you to Allen Bradley training skid.

k) Ping **10.0.1.91** to see if you can connect to the virtual machine





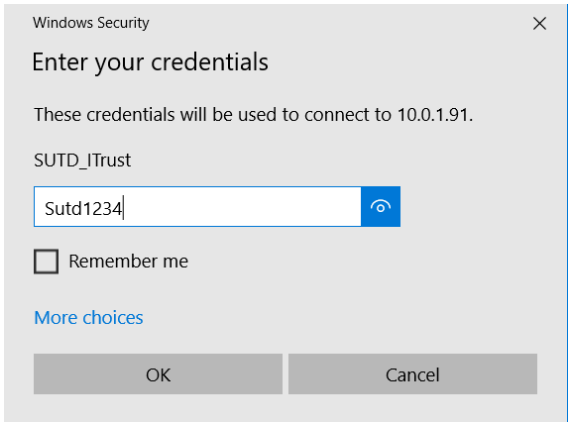
l) If the ping is successful, open **Remote Desktop Connection** and connect to **10.0.1.91**

Connect to AB Training Skid – VM

2. Remote Desktop (RDP) into VM: **10.0.1.91**. It will connect you to Allen Bradley training skid.

m) You will be prompted to Login

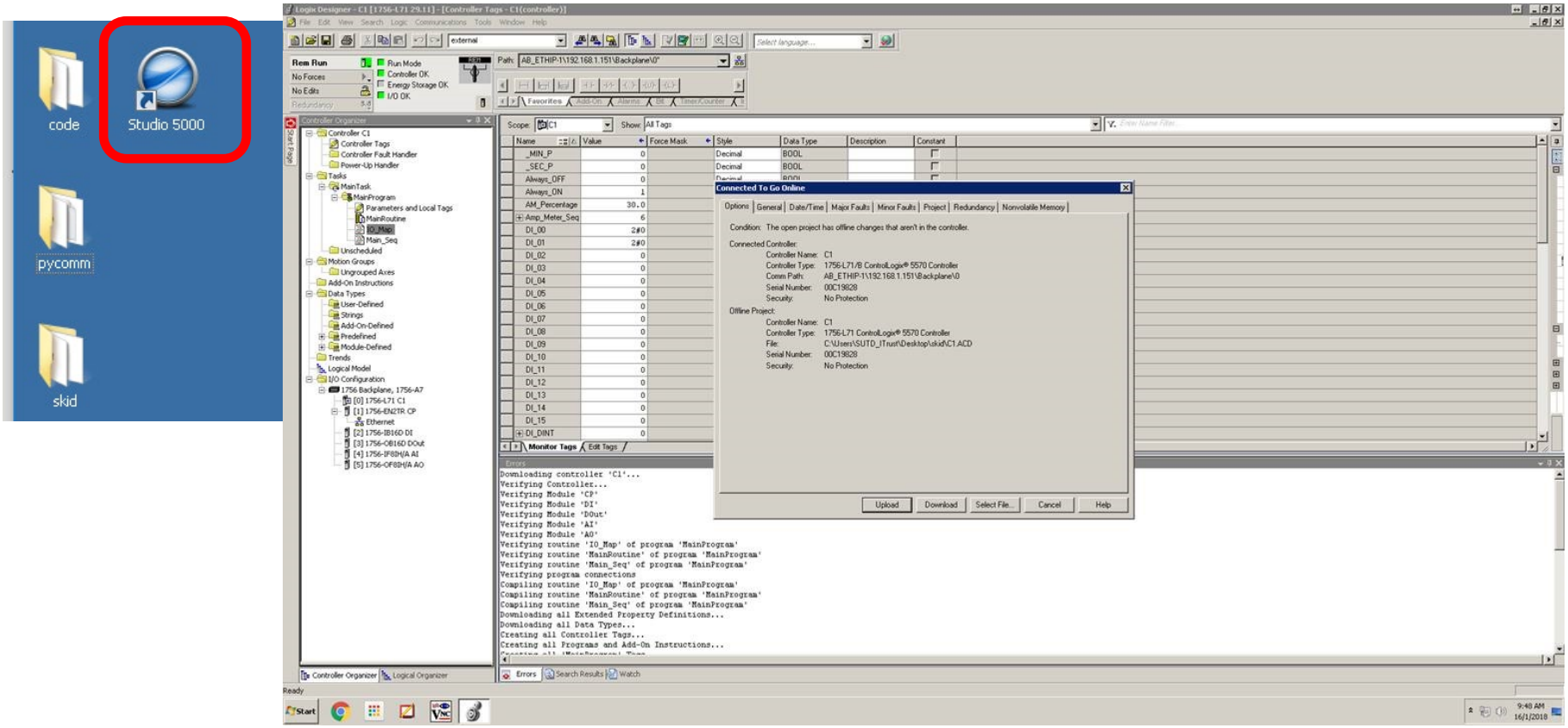
- Username: **SUTD_ITrust**
- Password: **Sutd1234**



n) You have successfully connected to the remote desktop (VM)

Connect to AB Training Skid – Studio 5000

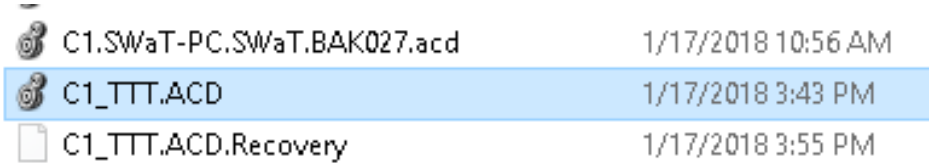
3. Then, you can start using **Studio 5000,** and download/upload program to the training skid.

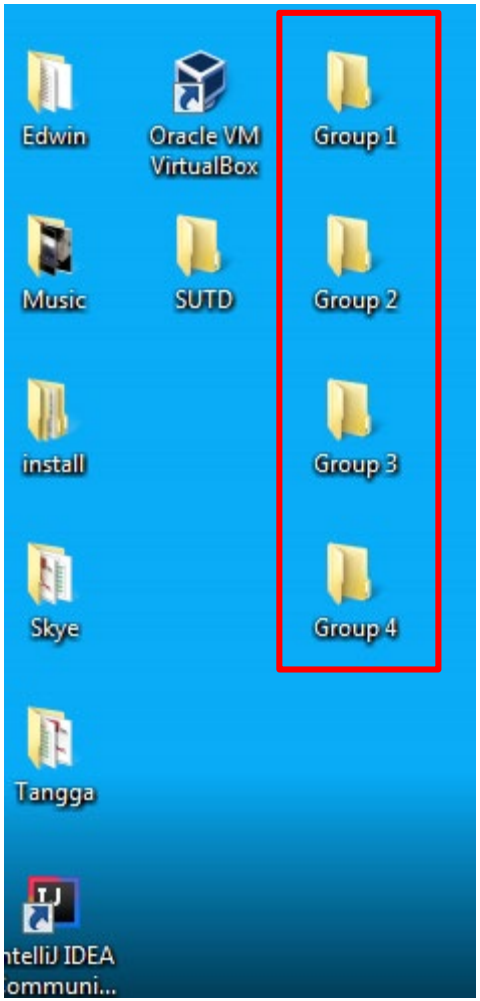


Studio 5000 – view a program

1. Click on folder with your group number on the remote desktop.

2. Search for **C1_TTT.ACD** in the folder and click on it.

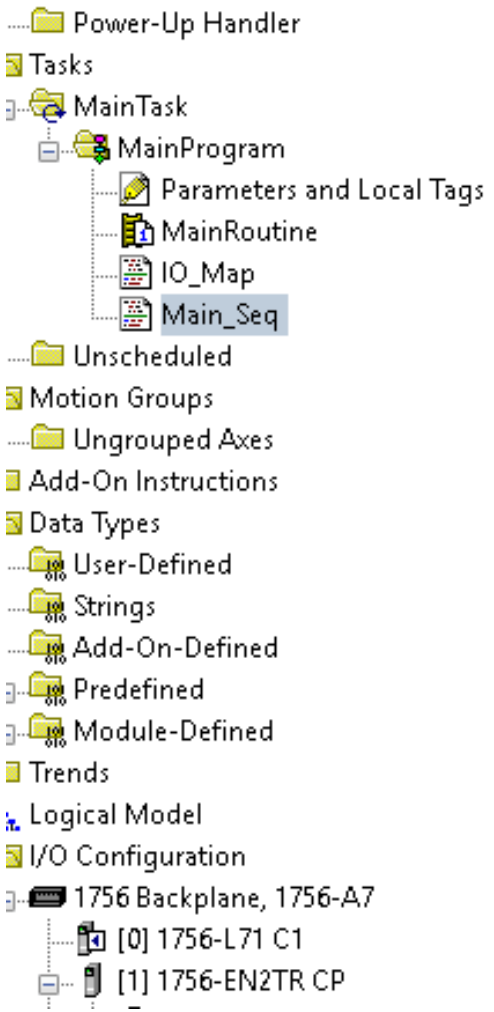




Studio 5000 – view a program

3. Navigate to “MainTask > MainProgram > Main_Seq”.

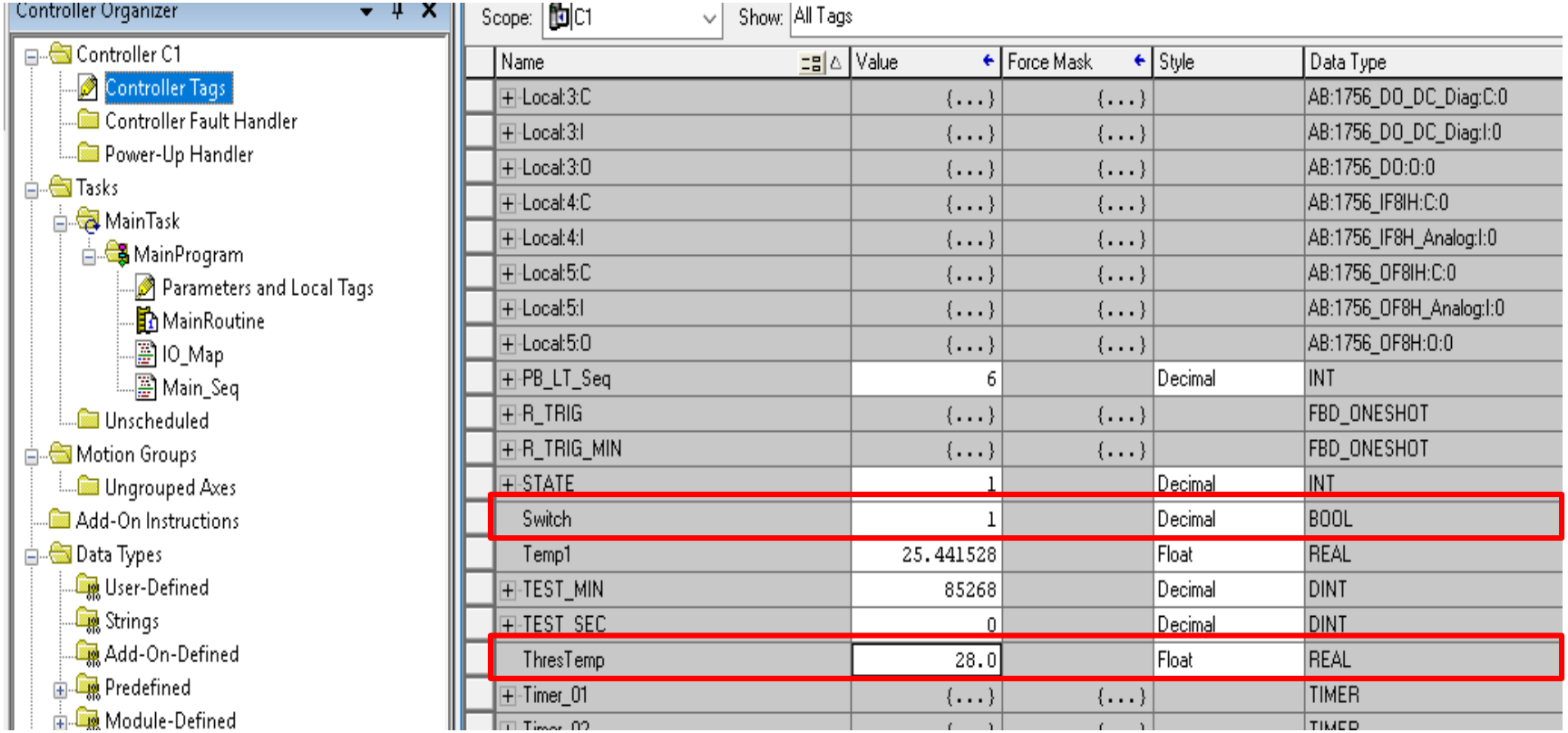
4. You should be able to view the program.




```
CASE STATE OF
    1:(* *)
        IF DI_01 THEN
            IF Switch THEN
                Switch := 0;
            ELSE
                Switch := 1;
            END_IF;

        END_IF;

        DO_02 := Switch;

        DO_06 := 0;

        IF Temp1 > ThresTemp THEN

            DO_10 := 1;
        ELSE
            DO_10 := 0;
        END_IF;

        IF _SEC_P THEN
            STATE := 1;
        END_IF;

ELSE
    STATE:=1;
END_CASE;
```

Studio 5000 – view a program

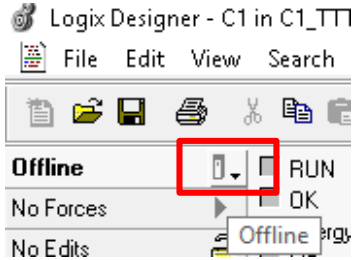
5. Navigate to “Controller C1 > Controller Tags”.



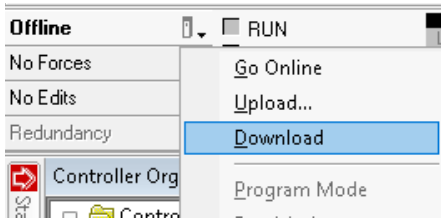
6. You can view the tags used in the program here such as **Switch** and **ThresTemp**. Make sure they have these values as stated.

Studio 5000 – run a program

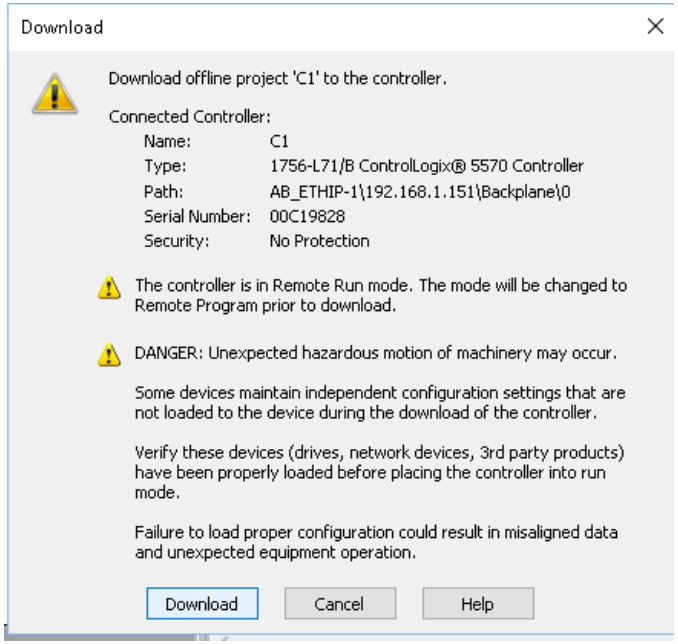
1. If the program is offline, click on the button next to the words “Offline”.



2. A drop down should appear.
   Select “Download”

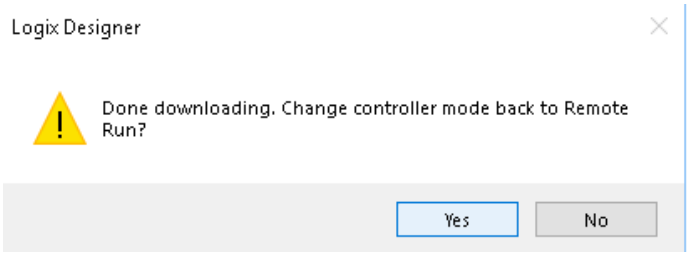


3. Click “Download”.

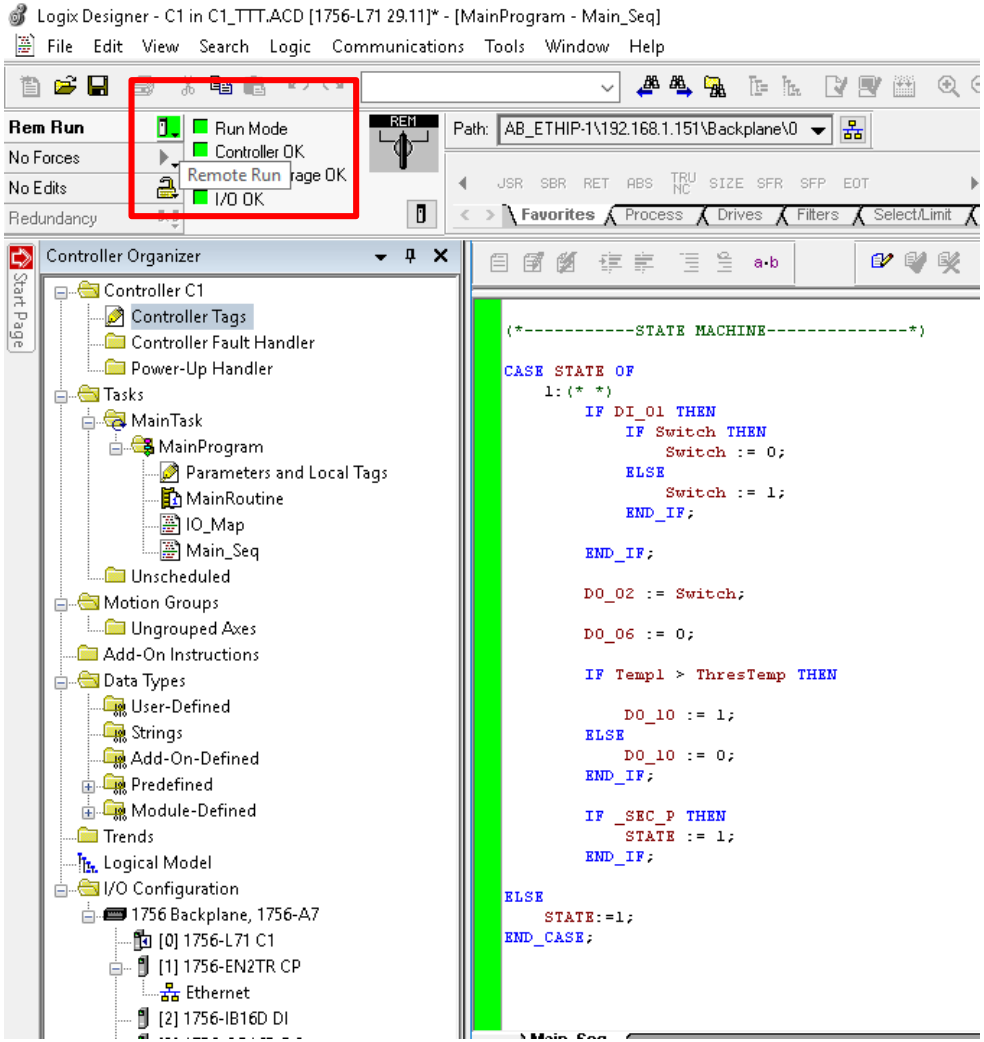


Studio 5000 – run a program

4. Click “Yes”.



5. The program should be running.



Studio 5000 – program demo

The current program is written such that the if DI_00 is pressed, the button DO_02 will turn on if it is not originally lit up and turn off if it is lit up.

1. Press DI_00.

2. Observe that DO_02 is lit up.

3. Press DI_00.

4. Observe that DO_02 is no longer lit up.

Studio 5000 – program demo

The current program is written such that the if the temperature sensor reads above 28 degrees, DO_10 button will light up.

1. Hold the temperature sensor to raise the temperature readings of the sensor.

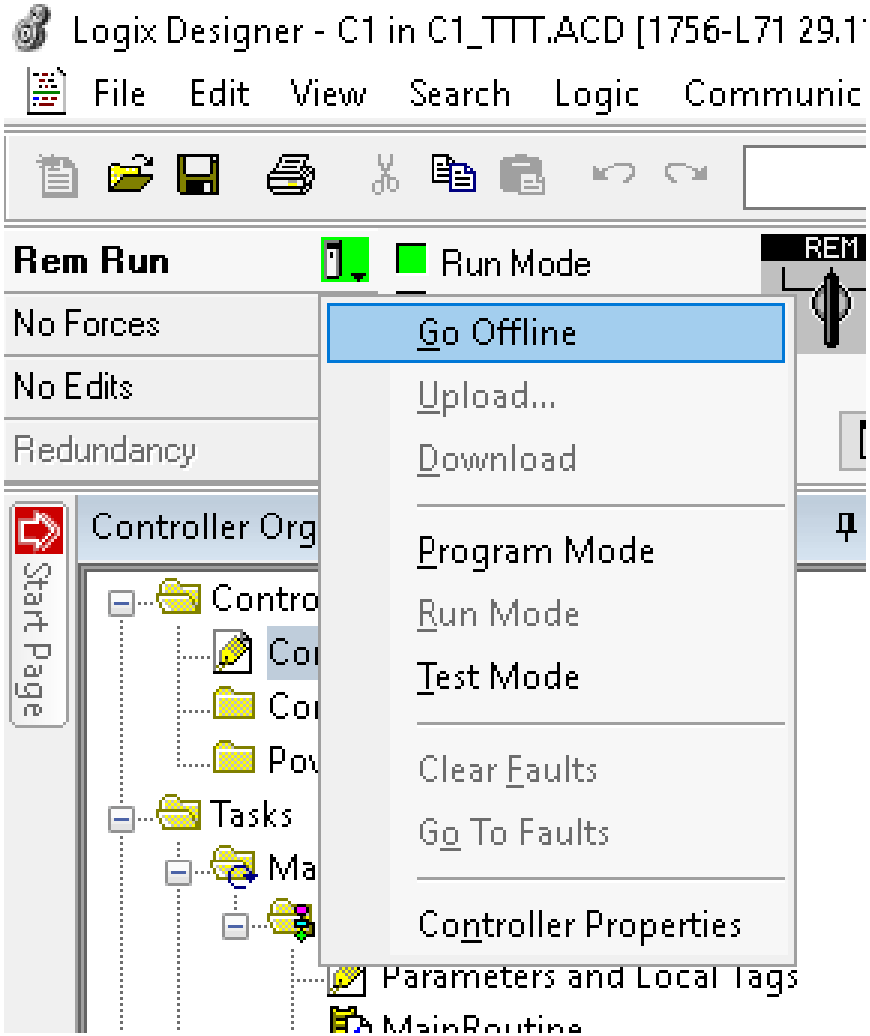
2. Wait till the temperature reads above 28 degrees.

3. Observe that DO_10 is lit up.

Studio 5000 – edit a program

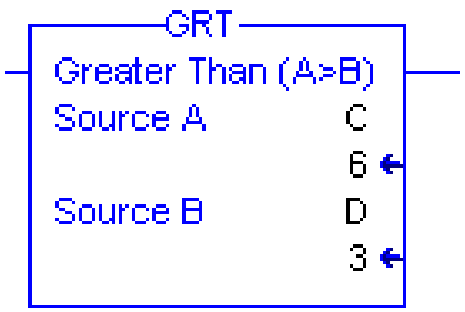
- If you would like to edit the main sequence, you must go offline first.

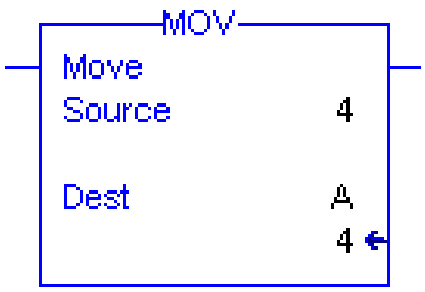
- Remember to click download and run the program as instructed previously after making changes to the program.

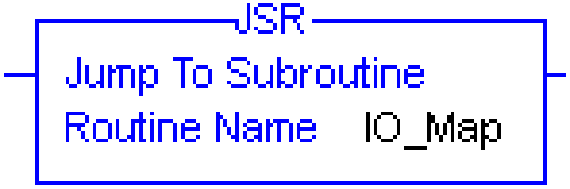


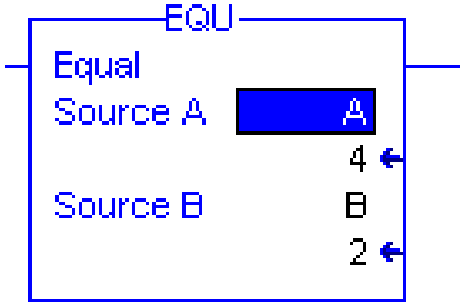
Studio 5000 – programming in LD/ST

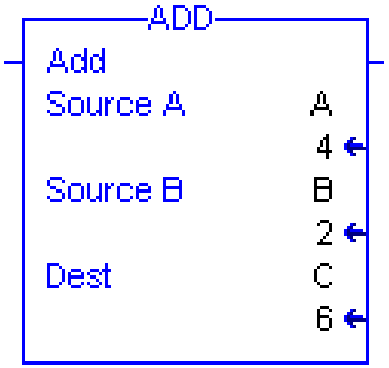
Some Graphical Representations:

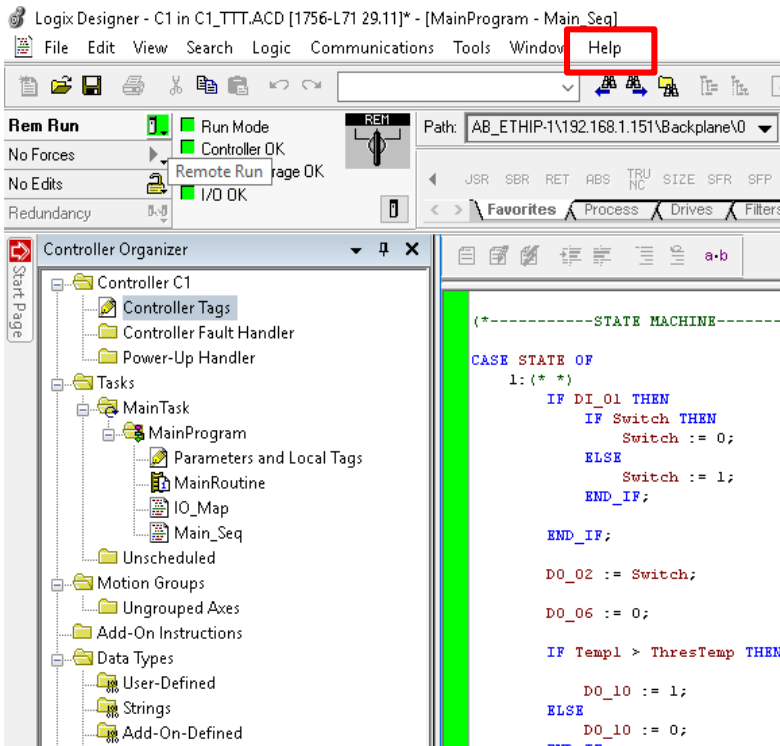








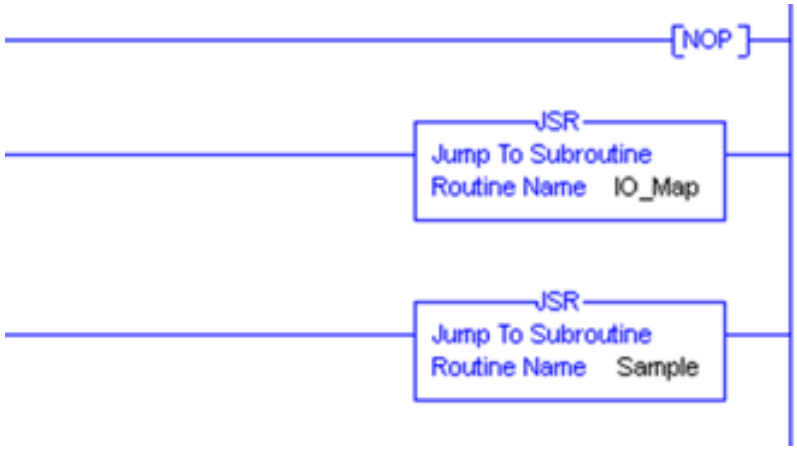




Get help from “Help > Instruction Help” in Studio 5000.

Studio 5000 – programming in LD/ST

A simple PLC program:



Subroutine IO_Map:

```
Local:3:0.Data.0  := DO_00;
Local:3:0.Data.1  := DO_01;
Local:3:0.Data.2  := DO_02;
Local:3:0.Data.3  := DO_03;
Local:3:0.Data.4  := DO_04;
Local:3:0.Data.5  := DO_05;
Local:3:0.Data.6  := DO_06;
Local:3:0.Data.7  := DO_07;
Local:3:0.Data.8  := DO_08;
```

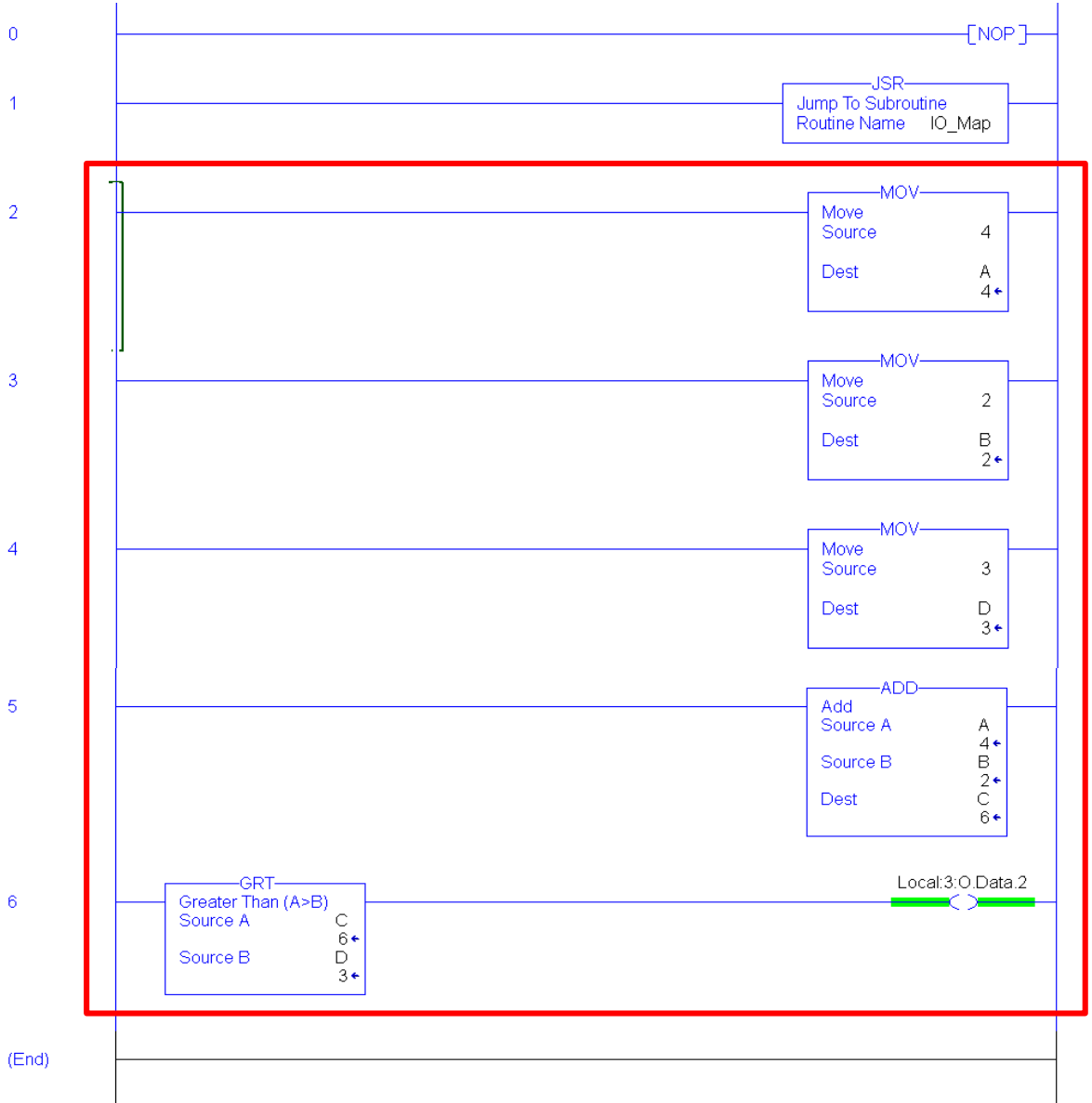
Subroutine Sample (in ST):

```
A := 6;
B := 2;
C := A+B;
IF C > 3 THEN
DO_02 := 1;
END_IF;
```

Discussion: Convert the Subroutine Sample written in ST to LD.

Studio 5000 – programming in LD/ST

A converted PLC program:



Studio 5000 – exercise

Perform the following 5 exercises using Studio 5000 separately:

1. If DI_01 is pressed, the button DO_06 will light up if it is not originally lit up and turn off if it is lit up.
2. If DI_02 is pressed, followed by DI_03 is pressed, the button DO_02 and DO_06 will light up if it is not originally lit up and turn off if it is lit up
3. Push DI_04 four times and DO_02 will light up and DO_02 will stay on till DI_04 is push for the fifth time.
4. Push DI_06 and DO_06 will remain lighted up for three seconds and will go off.
5. If the temperature sensor reads above 30 degrees, DO_02 and DO_10 buttons will light up.

Please include photographs and screenshots of your complete program in your assignment report.

Summary – Part 2

1. Introduction of PLC training skids in iTrust lab.
2. Practice of PLC programing with ladder diagram (LD) and structured text (ST) on Allen-Bradley training skid.
3. You are expected to be able to program and test PLCs.

4. Contact TA Edwin to make reservation (by project group) for PLC programming on training skid (Week 3 – Week 8).

Reading Materials – Part 2

1. Rockwell Software Studio 5000
2. Programming with ControlLogix

For reference only, available at eDimension.

Assignment – Week 2

- Download the assignment from eDimentsion.
- Submit your solution/report to eDimension by the deadline below.
- Each group only needs one submission.

- This assignment is due on **Tue Feb 19, 6:59 PM**.

End of Slides for Week 2