Introduction:

This exercise begins the programming portion of this experiment during which you will utilize the RSLogix 5000 software to develop a program that can be downloaded into the PLC in order to automate the proposed motor control system.

You will begin by configuring a new "project" within the RSLogix environment for the Compact Logix Controller (PLC) and other addressable devices that will be utilized within the control system. Ladder logic will then be used to create a program for the PLC that will provide an operator with basic stop-start control of Motor #1, which is energized by a contactor whose field coil is wired to output #0 of the PLC. Once the program is complete, you will download the program into the PLC and test the operation of the system, troubleshooting the program as necessary.

Note – If you have not yet completed parts B and C of this experiment because you haven't been able to access a lab bench that contains one of the physical motor control systems, you can begin part D (steps 1-47 on pages 1-12) on any lab bench since all of the desktop computers contain the RSLogix software. Just be sure to save your programs on a memory stick. Then, when one of the physical systems become available, you can go back and complete parts A-C, after which you can load your project into RSLogix software on that bench and continue on with the final steps of part D as well as part E of this experiment.

Procedure:

If you do not have access to a physical motor control system, you can still complete the first 12 pages of this procedure using any computer that contains the RSLogix software, after which you can complete the remainder once a physical system is available for use.

Log-In to the Desktop Computer

Log-in to the desktop computer using the following credentials:

Username: **.\student**
Password: **student**

RSLOGIX 5000 SOFTWARE

RSLogix 5000 is the software that you will be utilizing to program the Compact Logix PLCs. Software Revision 20 is loaded on the laboratory computers. Note that the software revision must match the firmware revision that is loaded into the PLCs, which is currently version 20.04.

Execute the **RSLogix 5000** software by double-clicking on the **RSLogix 5000** shortcut present on the desktop of the computer. If a shortcut is not available on desktop, the **RSLogix** software can be found by ***clicking*** on the **Start** button and choosing:

All Programs→Rockwell Software→RSLogix 5000 Enterprise Series→ RSLogix 5000.

Once the software loads, you can begin the process of creating a ladder-logic program for the PLC. To do this, you must first create a "**project**".

Note that, once you save your software project on a memory stick, you can also restart the software and reload your project simply by double-clicking the project file located on your memory stick.

Do NOT save your project on the desktop computer's hard-drive. The instructor will regularly delete any projects that are saved on the hard-drives of the lab computers.

Creating a New Project & Specifying the Controller, its Modules, and the VFD

Whenever you create a new project, you must first specify and characterize the controller (PLC) that will be utilized and the modules (if any) that are directly connected to the controller. This allows the RSLogix software to preconfigure its environment for all of the operational parameters of those devices, in-turn allowing the programmer to directly control or interface with those devices without necessarily knowing the technical aspects of their operation.

For this exercise, you will utilize a modular **PLC** that is composed of a:

- **1769-L32E Compact Logix 5332E Controller** (**E** ≡ includes an Ethernet port) with a:
  - **1769-IQ16 16-port, 24V$_{DC}$, Digital Input Module**, and a
  - **1769-OW16 16-port, Relay Output Module**.

Note – the input and output modules are directly connected to and communicate with the controller via its "Local" communication bus (***CompactBus***). The bus is characterized by positional "slots", such that the controller is considered to reside in slot 0, and the module (**IQ16**) that is directly plugged into the controller's CompactBus is considered to reside in slot 1. The module (**OW16**) that is plugged into CompactBus port that is available on the opposite side of the slot 1 module is considered to reside in slot 2, and so forth.

Additionally, when creating the project, you must specify any remote devices that will communicate with or be controlled by the PLC via a communication network. The proposed system also includes a **PowerFlex 40-E** VFD that contains a **22-COMM-E** (Ethernet) communications port.

Since an Ethernet network will be utilized for communication between the PLC and the VFD, the previously-assigned IP addresses must also be specified when configuring the project.

Create a New Project – Select & Configure the PLC's Controller (CPU):

1. Run the ***RSLogix 5000*** Software and open a "**New Project**" from the ***Quick Start*** window.

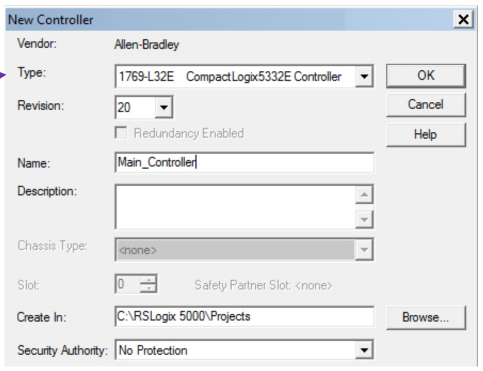
A "***New Controller***" dialog box will appear.

2. Locate and select the ***1769-L32E*** controller in the ***Type*** field.

3. Configure the controller as shown to the right and ***click*** "**OK**".

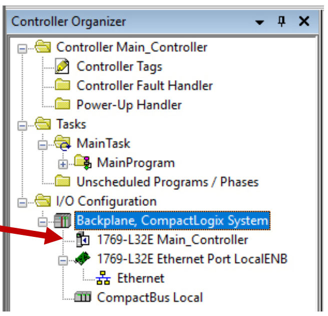
Note – you may replace the word "**Main**" in the ***Name*** field with your last name.

(I.e. – "**Lastname_Controller**")

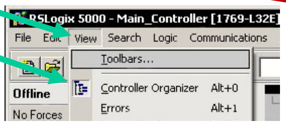

New Controller

Vendor: Allen-Bradley
Type: 1769-L32E CompactLogix5332E Controller
Revision: 20
☐ Redundancy Enabled
Name: Main_Controller
Description:
Chassis Type: <none>
Slot: 0 Safety Partner Slot: <none>
Create In: C:\RSLogix 5000\Projects
Security Authority: No Protection

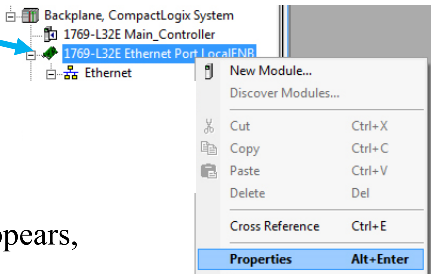
The ***1769-L32E*** controller should now appear under the ***I/O Configuration*** folder in the ***Controller Organizer*** section on the left-side of the RS Logix 5000 window.


Controller Organizer
Controller Main_Controller
Controller Tags
Controller Fault Handler
Power-Up Handler
Tasks
MainTask
MainProgram
Unscheduled Programs / Phases
I/O Configuration
Backplane, CompactLogix System
1769-L32E Main_Controller
1769-L32E Ethernet Port LocalENB
Ethernet
CompactBus Local

If the ***Controller Organizer*** section is not visible, ***click*** **View** in the main window and highlight the ***Controller Organizer*** option.

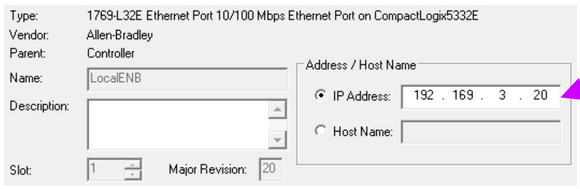


4. ***Right-click*** on the ***1769-L32E Ethernet Port LocalENB*** and choose **Properties**.



This will allow you to specify the ***IP Address*** that was previously assigned to the PLC in part C of this lab.

5. In the ***Module Properties Report: Controller:1*** window that appears, configure the ***IP Address*** as shown below and ***click*** **OK**.

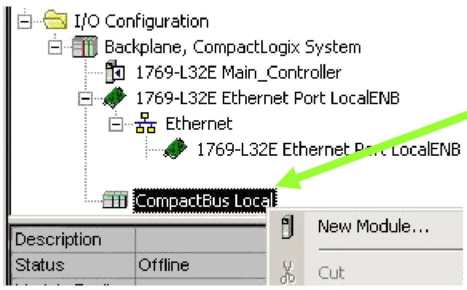


Type: 1769-L32E Ethernet Port 10/100 Mbps Ethernet Port on CompactLogix5332E
Vendor: Allen-Bradley
Parent: Controller
Name: LocalENB
Description:
Slot: 1 Major Revision: 20

Address / Host Name
● IP Address: 192 . 169 . 3 . 20
○ Host Name:

192.169.3.20

ADD & CONFIGURE THE INPUT/OUTPUT MODULES ATTACHED TO THE CONTROLLER:


I/O Configuration
Backplane, CompactLogix System
1769-L32E Main_Controller
1769-L32E Ethernet Port LocalENB
Ethernet
1769-L32E Ethernet Port LocalENB
CompactBus Local
Description
Status Offline
New Module...
Cut

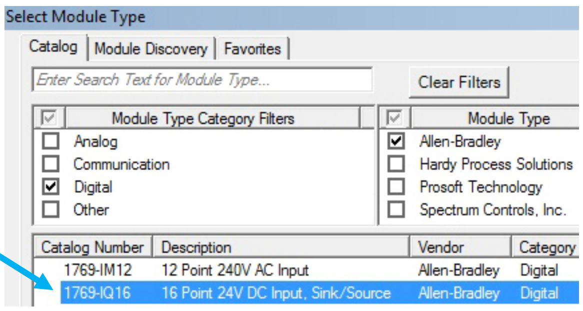
6. In the ***Controller Organizer*** window, locate **CompactBus Local** under ***I/O Configuration***. You may need to ***click*** ⊞ to expand the contents.

7. Right-click on ***CompactBus Local*** and select **New Module…** in order to add the **input module**.

8. When the ***Select Module Type*** window appears, ***uncheck*** all of the filter option boxes <u>except</u> for "**Digital**" and "**Allen-Bradley**".

Locate and select a ***1769-IQ16*** from the list of modules in the lower field and ***click*** **Create**.

A ***New Module*** pop-up window will appear.



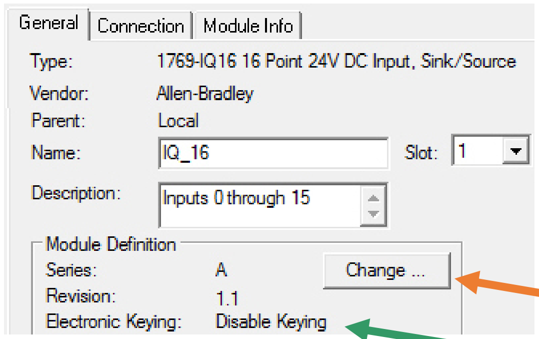
Select Module Type

Catalog | Module Discovery | Favorites

Enter Search Text for Module Type... Clear Filters

Module Type Category Filters	Module Type
☐ Analog	☑ Allen-Bradley
☐ Communication	☐ Hardy Process Solutions
☑ Digital	☐ Prosoft Technology
☐ Other	☐ Spectrum Controls, Inc.

Catalog Number	Description	Vendor	Category
1769-IM12	12 Point 240V AC Input	Allen-Bradley	Digital
1769-IQ16	16 Point 24V DC Input, Sink/Source	Allen-Bradley	Digital



General | Connection | Module Info

Type: 1769-IQ16 16 Point 24V DC Input, Sink/Source
Vendor: Allen-Bradley
Parent: Local
Name: IQ_16 Slot: 1
Description: Inputs 0 through 15

Module Definition
Series: A Change ...
Revision: 1.1
Electronic Keying: Disable Keying

9. Configure the module as shown to the left but ***do <u>NOT</u> click* OK** yet.

Note – ***Slot 1*** is selected because the ***IQ16*** is the first module that is attached to the controller's local communication bus.

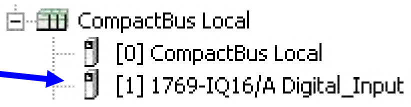
10. In the ***Module Definition*** section of the ***New Module*** window, ***click*** "**Change…**"

11. Change the value of "***Electronic Keying***" to "**Disable Keying**" in the ***Module Definition*** window and ***click*** **OK** to accept that change.

Electronic keying is utilized to ensure that the device defined in your project is the same as the actual installed device including the device's firmware revision. Although Allen-Bradley recommends that the "Disable Keying" option not be used due to safety risks, we will utilize this option in the Q-215 lab due to the variance of the modules that may be present in the lab.

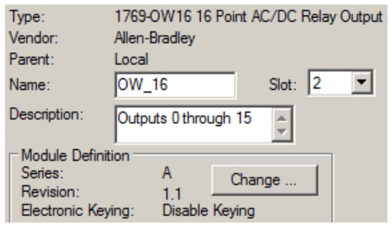
12. The ***Electronic Keying*** option in the ***New Module*** window should now read "***Disable Keying***". If this is correct, ***click*** "**OK**".

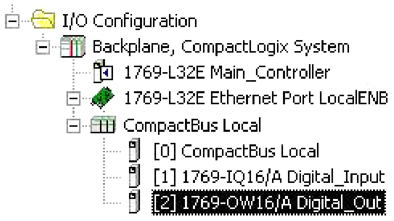
The ***1769-IQ16*** should now appear under ***CompactBus Local*** in the ***I/O Configuration*** folder.


CompactBus Local
[0] CompactBus Local
[1] 1769-IQ16/A Digital_Input

13. Once again, ***right-click*** on ***CompactBus Local*** and select **New Module…** to add the **output module**.

14. ***Select*** and ***configure*** a **1769-OW16** in the same <u>manner</u> that the *1769-IQ16* was configured. Note that the ***OW16*** is the second module (***Slot 2***) on the local communication bus.

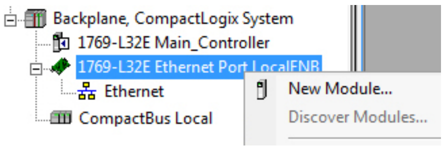

Type: 1769-OW16 16 Point AC/DC Relay Output
Vendor: Allen-Bradley
Parent: Local
Name: OW_16
Slot: 2
Description: Outputs 0 through 15
Module Definition
Series: A
Revision: 1.1
Electronic Keying: Disable Keying


I/O Configuration
Backplane, CompactLogix System
1769-L32E Main_Controller
1769-L32E Ethernet Port LocalENB
CompactBus Local
[0] CompactBus Local
[1] 1769-IQ16/A Digital_Input
[2] 1769-OW16/A Digital_Out

15. Once complete, the ***1769-IQ16*** and the ***1769-OW16*** should both be present under ***CompactBus Local*** in the ***I/O Configuration*** folder.

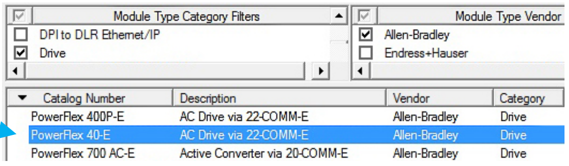
ADD & CONFIGURE THE POWERFLEX 40 VFD CONNECTED TO THE ETHERNET NETWORK:

16. Although it is not directly connected to the PLC's local CompactBus, you must also add the **PowerFlex 40** Variable Frequency Drive to the PLC's **I/O Configuration** since the PLC will communicate with and control the operation of the VFD via the system's Ethernet network.


Backplane, CompactLogix System
1769-L32E Main_Controller
1769-L32E Ethernet Port LocalENB
Ethernet
CompactBus Local

Right-click on ***1769-L32E Ethernet Port LocalENB*** within the ***I/O Configuration*** folder and then select ***New Module***…

17. In the ***Select Module*** window, locate and select a **PowerFlex 40-E** drive.

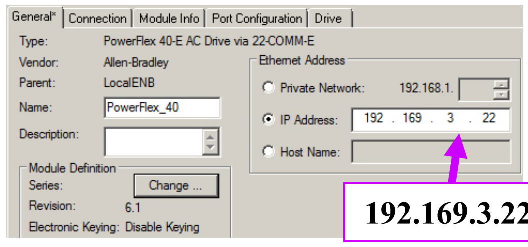


Module Type Category Filters	Module Type Vendor
☐ DPI to DLR Ethernet/IP	☑ Allen-Bradley
☑ Drive	☐ Endress+Hauser

Catalog Number	Description	Vendor	Category
PowerFlex 400P-E	AC Drive via 22-COMM-E	Allen-Bradley	Drive
PowerFlex 40-E	AC Drive via 22-COMM-E	Allen-Bradley	Drive
PowerFlex 700 AC-E	Active Converter via 20-COMM-E	Allen-Bradley	Drive

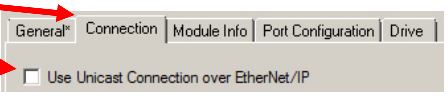
Note – the drive will be easier to locate if you uncheck all of the filters except "***Drive***" and "**Allen-Bradley**".

18. Configure the module as shown to the right but do <u>**not**</u> ***click*** "**OK**" yet. (Be sure to change ***Electronic Keying*** to "**Disable Keying**".)


General* | Connection | Module Info | Port Configuration | Drive
Type: PowerFlex 40-E AC Drive via 22-COMM-E
Vendor: Allen-Bradley
Parent: LocalENB
Name: PowerFlex_40
Description:
Ethernet Address
Private Network: 192.168.1.
IP Address: 192 . 169 . 3 . 22
Host Name:
Module Definition
Series:
Revision: 6.1
Electronic Keying: Disable Keying

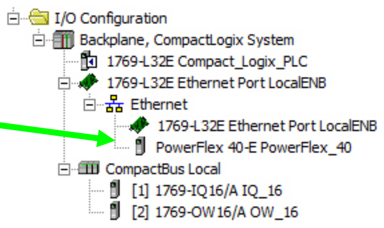
192.169.3.22

19. Select the **Connection** tab in the ***New Module*** window and <u>**uncheck**</u> the box next to "***Use Unicast Connection over EtherNet/IP***".


General* | Connection | Module Info | Port Configuration | Drive
☐ Use Unicast Connection over EtherNet/IP

20. Confirm that the module is configured correctly, ***click*** "**OK**".

The ***PowerFlex 40*** should new appear in the ***I/O Configuration*** folder as an ***Ethernet*** device.


I/O Configuration
Backplane, CompactLogix System
1769-L32E Compact_Logix_PLC
1769-L32E Ethernet Port LocalENB
Ethernet
1769-L32E Ethernet Port LocalENB
PowerFlex 40-E PowerFlex_40
CompactBus Local
[1] 1769-IQ16/A IQ_16
[2] 1769-OW16/A OW_16

All of the control system devices are now configured.

You have completed the initial setup of the project during which you defined and configured the PLC that you will be using, including both the controller (CPU) and the attached input and output modules, along with the variable frequency drive (VFD) that will be controlled by the PLC as a network-connected device.

Note – if additional modules or network-connected devices need to be added into the control system in the future, such as a second Input Module, another VFD, or a Remote I/O Module, they would be added and configured in a similar manner to those previously configured.

At this point, it is recommended that you **save your project** on a <u>USB memory-stick</u> and that you also make a **backup copy** (named "**Lastname_Controller_new.ACD**") of the project file.

Once you complete the tutorial, you can use the "new" version of the project file to begin creating new programs without having to repeat the steps required to select and configure all of the control system devices.

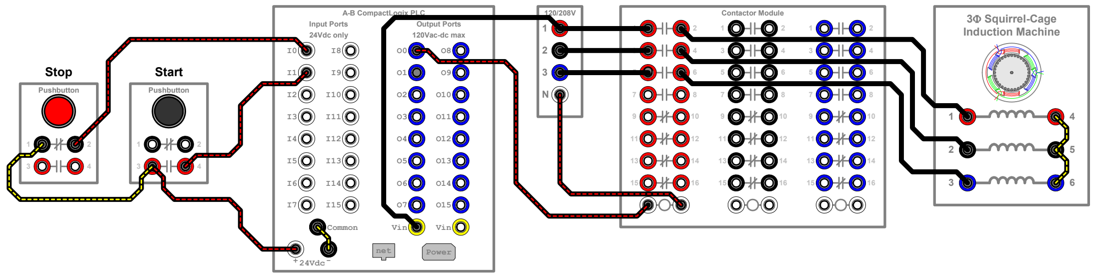
You are now ready to begin creating your first Ladder Logic program.

<u>Notes Regarding the Operation of the Motor Control System for Program #1</u>:

The PLC-based motor control system constructed for use during this experiment contains two induction motors, the first of which is energized by means of a contactor whose field-coil is wired directly to the PLC's output module, and the second of which is supplied by a PowerFlex 40 VFD that will be controlled remotely by the PLC via the control system's Ethernet network.

Although the physical system contains two motors, we will begin by creating a simple program that provides "Start-Stop" control only for **motor #1** such that the PLC will start the motor by energizing the main contactor's field-coil when an operator presses the "Start" button and the PLC will stop the motor by de-energizing the field-coil when an operator presses the "Stop" button.

As a reminder, the normally-closed (NC) "Stop" and normally-open (NO) "Start" pushbuttons are wired to Inputs **0** and **1** respectively of the PLC's input module and the contactor's field-coil is wired to Output **0** of the PLC's output module, as shown below in a simplified drawing of the control system.



Motor Control System Components for Program #1

The ladder logic program for this motor controller will only utilize two of the three primary ladder instructions (XIC and OTE) and a ladder topology that is similar to the control circuit from the basic stop-start motor controller that was presented at the beginning of the semester. This will allow us to focus on the fundamental concepts and procedures associated with creating the ladder logic program and the use of those primary instructions, without having to deal with the more complex logic and the additional instructions that will be required to control the operation of the VFD-supplied motor.

Development of the Initial Ladder-Logic Program

The initial program will be developed in two stages:

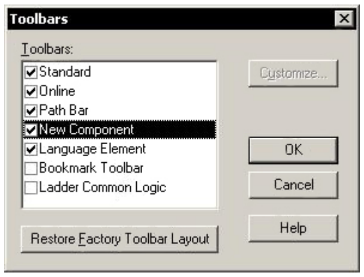
i) You will create a generic ladder diagram that provides the logic required for the proposed motor-control system, after which

ii) You will modify the instructions within the ladder diagram to link their operation to that of the appropriate inputs and outputs to which the pushbuttons and field-coil are wired.

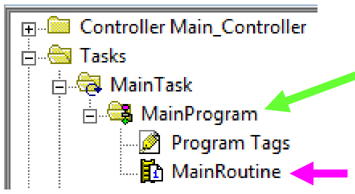
Note that, although these two stages could be combined into one, it will be easier to understand the concepts presented if you perform them one at a time.

Creation of the General Ladder Diagram:

In order to begin creating the ladder diagram, you must open the Routine Editor window within the RSLogix program.



21. In the main RSLogix 5000 window, Choose **View→Toolbars**. When the ***Toolbars*** window opens, be sure the first five options are selected and then ***click*** "**OK**".

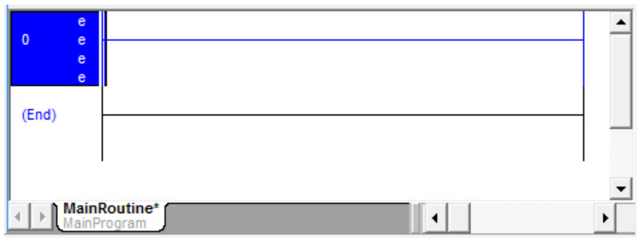


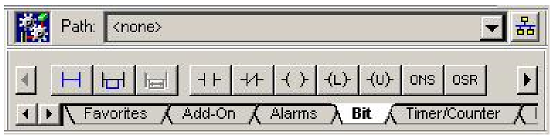
22. In the ***Controller Organizer*** window, expand the ***MainProgram*** folder by ***clicking*** ⊞ next to the folder.

23. ***Double-click*** the ***MainRoutine*** icon to open the ***Routine Editor*** window.

24. The ***Routine Editor*** window should now display a blank ladder diagram as shown below:

Note the column of **e**'s that appear to the left of the rung. The **e**'s signify that the rung is in "**edit mode**" to signify that one or more of the instructions on the rung have not yet been properly defined. Once all instructions have been properly defined, the **e**'s will disappear.

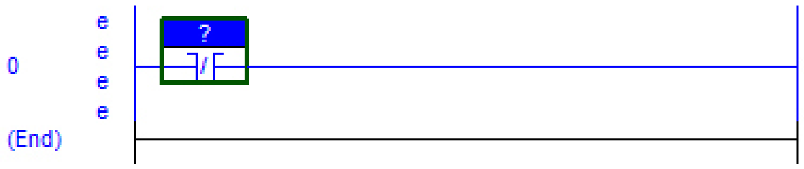




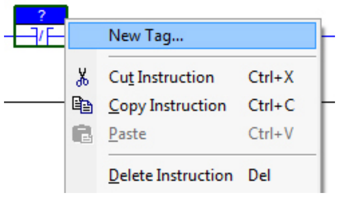
25. Choose the **Bit** tab in the ***New Component*** toolbar. The **Bit** tab displays instructions that are associated with a single-bit in memory.

26. ***Left-click*** and ***hold*** the ***XIO instruction***.

27. Using the mouse, ***drag and place*** this instruction in the left-most position on ***Rung 0***. Note that as you drag the instruction into the ***Routine Editor*** window, small green circles will appear on the ladder diagram that show the available locations in which the instruction may be placed.



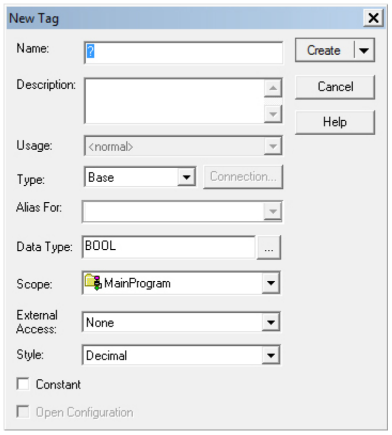
28. ***Right-click*** on the "**?**" above the *XIO instruction* and choose "**New Tag**".



Tags: Tags contain information that identifies <u>data stored in memory</u>, <u>allowing that data to be associated with a specific instruction</u> in order to define its operation.

For example – a bit in memory that is used to define the state of the XIO (0 ≡ TRUE, 1 ≡ FALSE)

New Tag Data Fields:



New Tag

Name: ?

Description:

Usage: <normal>

Type: Base

Alias For:

Data Type: BOOL

Scope: MainProgram

External Access: None

Style: Decimal

☐ Constant

☐ Open Configuration

Name – an alpha/numeric identifier (including underscores) assigned to data stored in memory.

Names can be used to help clarify the ladder logic structure by relating data to a specific task, function, or the physical device linked to that data.

Names must begin with an alphabetic character, cannot be greater than 40 characters in length, and are not case sensitive.

Description – text-based field that may be used to further identify the function of the data.

Type – defines how the tag operates.

Base – a tag that results in the allocation of memory (when initially created) and that provides a reference to that data stored in that memory location, allowing the data to be directly linked to the operation of an instruction.

Alias – a tag that allows the programmer to assign a name to an existing base tag, such as the base tags that are automatically created when a specific module or device is added to the controller's configuration.

For Example – RSLogix 5000 created a new base tag for each input port when the IQ-16 module was added to the controller's configuration. The programmer can then create an Alias tag named "Stop" and assign it to one of those base tags.

Alias For ***(only active for Type: Alias)*** – the pre-defined base tag re-named by the Alias tag.

Data Type – defines the type of data stored at the specific memory location, such as:

BOOL (**Boolean**): 0 or 1 **INT** (**Integer**): –32768 to +32767

Scope – defines whether a tag is ***global*** or ***local***. The controller allows an application to be split into multiple programs or subroutines. A global tag is available to all of the programs, while a local tag is available only to a specific program, such as "***Main Program***".

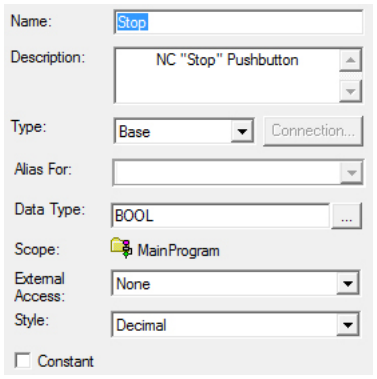
External Access – defines whether the tag will have Read/Write, Read Only, or no (None) access from external applications such as HMIs.

Style – defines how the tag is <u>displayed</u>. (Style does <u>not</u> affect the "***Data Type***")

Binary: Base 2 **Decimal**: Base 10

Constant – prevents executing logic (instructions) from writing values to the tag if checked.

29. Define the ***New Tag*** fields as shown to the right and ***click* OK.**

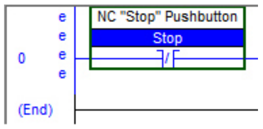


Name:	Stop
Description:	NC "Stop" Pushbutton
Type:	Base
Alias For:	
Data Type:	BOOL
Scope:	MainProgram
External Access:	None
Style:	Decimal
Constant	☐

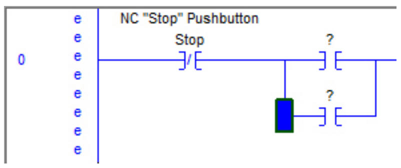
This ***XIO*** instruction will provide the logic for the controller's "***Stop***" button within the ladder diagram. The instruction will return **TRUE** when the bit referenced by the tag "***Stop***" is "**0**", and it will return **FALSE** when the bit is "**1**".

Note – an ***XIO*** was chosen because it resembles a NC contact, and a Stop-Start controller utilizes a NC "*Stop*" button. It turns out that you will need to change the instruction type to *XIC* when its operation is linked to that of Input 0, but for now it should remain as an *XIO*.

30. The ***Stop*** button should now appear in your ladder diagram. Note that a green bar may appear behind the *XIO*.

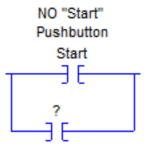


31. Add a parallel ***Branch*** to the right of the "***Stop"*** button and place an ***XIC*** on each of the parallel paths.



The ***XIC*** in the upper branch will provide the logic for the controller's "***Start***" button, while the ***XIC*** in the lower branch will eventually act as a "***hold-in***" contact after "***Start***" is released.

32. Define a ***New Tag*** for the ***XIC*** in the upper branch (similar to the *Stop* tag) but name it "**Start**".



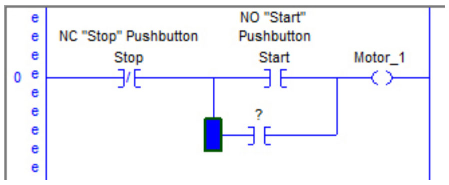
Note – the lower ***XIC's*** tag will be defined later

33. Place an "***OTE***" instruction to the right of the parallel branches.

The ***OTE*** instruction will provide the function of the field coil in the stop-start controller which, when energized, actuates the contactor's main contacts, in-turn energizing motor #1.

In terms of the logic of the ladder rung, when the ***rung condition*** to the left of the *OTE* is **TRUE**, the *OTE* sets its bit to "**1**" (i.e. – energize the motor), and when the rung condition is **FALSE**, the *OTE* resets its bit to a "**0**" (i.e. – de-energize the motor).

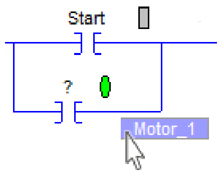
34. Define a ***New Tag*** for the ***OTE*** and name it "**Motor_1**".



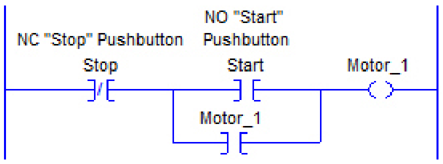
35. In a physical system, an auxiliary-contact from the main contactor is placed in parallel with the NO (Start) button to act as a "hold-in" contact once the field-coil is energized.

Since the ***Motor_1*** *OTE* represents the main contactor's field coil, the "hold-in" *XIC* can be assigned the same tag as the *OTE* such that the *XIC's* state will be determined by the state of the bit **Motor_1**. (I.e. – the *XIC* will be **TRUE** when the *OTE* sets bit **Motor_1** to a "**1**")

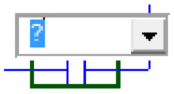
To copy the ***OTE's*** tag onto the ***XIC***, ***left-click*** and ***hold*** the mouse pointer over the *OTE's* **tag name** *"Motor_1"* and ***drag*** the tag name over to the "hold-in" ***XIC*** instruction until a green circle appears next to the *XIC's* "***?***", at which point you can ***release*** the mouse button.



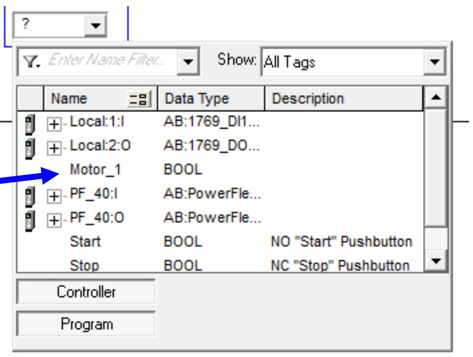
The "hold-in" ***XIC*** should now have the same ***tag name*** as the ***Motor_1 OTE***.



Note – you could also have assigned the ***tag name*** **Motor_1** to the ***XIC*** by ***double-clicking*** on the "*?*" (tag name) above the ***XIC*** to enable a drop-down menu.

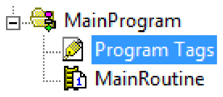


Opening the drop-down menu displays a list of all of the previously-defined ***tags*** from which you could have select the desired tag (**Motor_1**) by locating and ***double-clicking*** on **Motor_1** in the list.



Name	Data Type	Description
Local:1:I	AB:1769_DI1...	
Local:2:O	AB:1769_DO...	
Motor_1	BOOL	
PF_40:I	AB:PowerFle...	
PF_40:O	AB:PowerFle...	
Start	BOOL	NO "Start" Pushbutton
Stop	BOOL	NC "Stop" Pushbutton

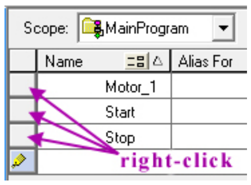
Also note that there are tags displayed that you did not create. These are the base tags that the RSLogix software created when you added the controller/modules/VFD to the project.



Additionally, if you need **to delete** (or edit) a **tag** that you defined, you can ***double-click*** the **Program Tags** entry within the ***MainProgram*** folder in the ***Controller Organizer*** window.

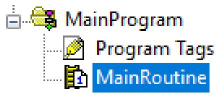
The ***ladder editor*** window will be replaced by a list of the tags that have the Scope: MainProgram. The **Monitor Tags** tab is selected by default, allowing you to <u>view</u> the **tags** and their states.

To delete (or edit) a tag, click the **Edit Tags** tab at the bottom of the tag display.

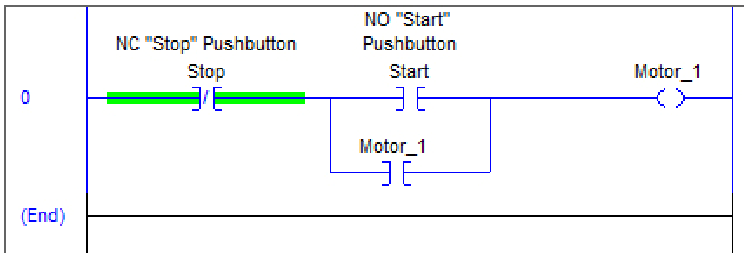


Then, ***right-click*** the grey box next to the tag that you want to delete (or edit) and select **Delete** from the list of options if you want to delete that **tag**.

When you are done viewing or editing the **tags**, ***double-click*** the **MainRoutine** entry within the ***MainProgram*** folder in the ***Controller Organizer*** window to return to the **MainRoutine** editor.



Even though your ladder diagram contains only a single rung, you have already finished creating the **general ladder diagram** that will provide the operational logic for Motor Control System #1, a PLC-based, Stop-Start controller for motor #1. Your ladder diagram should appear as follows:



Assume that the *Stop*, *Start*, and *Motor_1* bits are all initially (normally) "**0**". The ladder diagram shown to the left is currently highlighted for this condition.

Since the ***Stop*** *XIO* is normally **TRUE**, when the ***Start*** *XIC* becomes **TRUE**, the ***rung condition*** will become **TRUE**, causing the ***Motor_1*** *OTE* to set the bit **Motor_1** to "**1**", in-turn causing the ***Motor_1*** *XIC* to also become **TRUE**. When this occurs, even if the ***Start*** *XIC* returns to **FALSE**, the ***rung condition*** will remain **TRUE**, causing the ***Motor_1*** *OTE* to hold bit **Motor_1** at a "**1**".

If the ***Stop*** *XIO* becomes **FALSE**, the ***rung condition*** will become **FALSE**, causing the ***Motor_1*** *OTE* to reset the bit **Motor_1** to "**0**", in-turn causing the ***Motor_1*** *XIC* to become **FALSE**., at which point the ***rung condition*** will remain **FALSE** even if the ***Stop*** *XIO* returns to its normally **TRUE** state.

Although you have created a **general ladder diagram** that provides the operational logic for a basic Stop-Start motor controller, in its present form it will not actually cause the PLC to do anything because the states of the ***Stop***, ***Start***, and ***Motor_1*** instructions have not yet been linked to operation of the physical devices that are wired to the input ports and output ports of the PLC.

For example, the **OW16** Output Module determines the states (ON or OFF) of its output ports based on the values of a set of bits that are contained within its memory, such that:

- When bit ***O.Data.X*** is a "**1**", ***Output Port X*** is switched **ON**, and
- When bit ***O.Data.X*** is a "**0**", ***Output Port X*** is switched **OFF**.

As currently configured, the ***Motor_1*** *OTE* in your ladder diagram only has the ability to set or reset a bit named **Motor_1** within the PLC's memory. But, if the ***Motor_1*** *OTE* could instead be reconfigured to set or reset the bit ***O.Data.0*** in the OW16's memory, then the *OTE* would control the state (**ON** or **OFF**) of ***Output Port 0*** to which the field-coil of the main contactor is wired.

Similarly, the values of a set of bits that are contained within the memory of the **IQ16** Input Module are determined by the states (TRUE or OFF) of its input ports, such that:

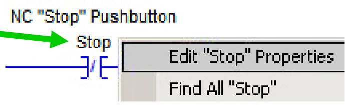
- When ***Input Port X*** detects +24V$_{DC}$, bit ***I.Data.X*** is set to a "**1**", and
- When ***Input Port X*** does not detect +24V$_{DC}$, bit ***I.Data.X*** is reset to a "**0**".

Since pushbuttons are wired between a +24V$_{DC}$ source and ***Input Ports 0*** and ***1***, the states of ***Inputs 0*** and ***1*** are directly affected by whether each of these buttons are being pressed or released.

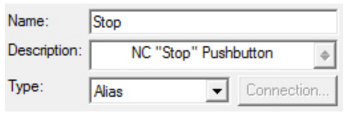
But, as currently configured, the states (**TRUE** or **FALSE**) of the ***Stop*** *XIO* and the ***Start*** *XIC* are only determined by the values of the **Stop** and **Start** bits within the PLC's memory. If these instructions could be reconfigured to instead have their states based on the values of the bits ***I.Data.0*** and ***I.Data.1*** that are located in the IQ16's memory, then the states of the ***Stop*** *XIO* and the ***Start*** *XIC* would in-turn depend on whether or not the **Stop** and **Start** buttons are being pressed.

<u>Modifying/Linking the Ladder Instructions to the States of the Input & Output Ports</u>

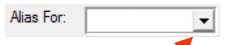
36. ***Right-click*** on the **tag <u>name</u>** of the ***Stop XIO*** in rung 0 and choose **Edit "Stop" Properties**.



37. Change the "***Type***" to "**Alias**".


Name: Stop
Description: NC "Stop" Pushbutton
Type: Alias

38. In the "***Alias For***" field, ***click*** on the down-arrow and locate the "**Local:1:I**" option in the menu that appears.



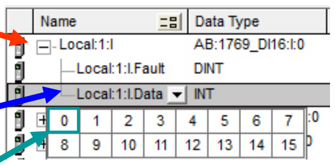
"**Local:1:I**" contains all of the accessible tags that the RSLogix software automatically created when you initially added the Input Module to the project.

"**Local:1**" refers to the device that resides in slot 1 of the controller's CompactBus.

39. Expand "**Local:1:I**" by clicking the preceding ⊞.

40. "**Local:1:I.Data**" is an INT (16-bit integer) tag contains all 16 of the bits associated with the states of *Input Ports 0* through *15*.

Locate and ***click*** the down-arrow next to "**Local:1:I.Data**" to access the individual bits.

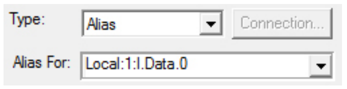


Name	Data Type
Local:1:I	AB:1769_DI16:I:0
Local:1:I.Fault	DINT
Local:1:I.Data	INT

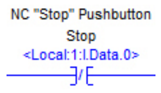
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15

41. To assign bit "**I.Data.0**" to the *XIO*, ***double-click*** on the box containing "**0**".

42. The "*Alias For*" field of the "***Tag Properties – Stop***" window should appear as shown to the right. If so, ***click*** "**OK**".

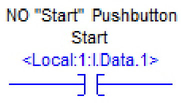

Type: Alias
Alias For: Local:1:I.Data.0

You have now assigned the bit "**I.Data.0**" to the ***Stop XIO***, in-turn linking the state of the *XIO* to the position of the NC ***Stop*** pushbutton that is wired to ***Input Port 0*** (zero) of the PLC.

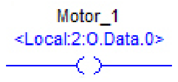


The ***base tag <Local:1:I.Data.0>*** displays (in blue) directly above the *XIO*, the ***alias tag*** **Stop** displays immediately above the base tag, and the ***description*** (NC "Stop" Pushbutton) displays at the very top.

43. Modify the tag of the ***Start XIC*** to become an ***alias tag*** assigned to the base tag ***<Local:1:I.Data.1>*** in order to link the state of the ***Start XIC*** to the position of the NO ***Start*** pushbutton that is wired to ***Input Port 1***.



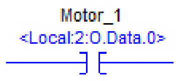
44. Similarly, modify the tag of the ***Motor_1 OTE*** to become an ***alias tag*** assigned to the base tag ***<Local:2:O.Data.0>*** in order to link the value of bit "**O.Data.0**" to the state of the ***Motor_1 OTE***, in-turn allowing the state of the field-coil that is wired to ***Output Port 0*** (zero) to be determined by the state of the *OTE*.



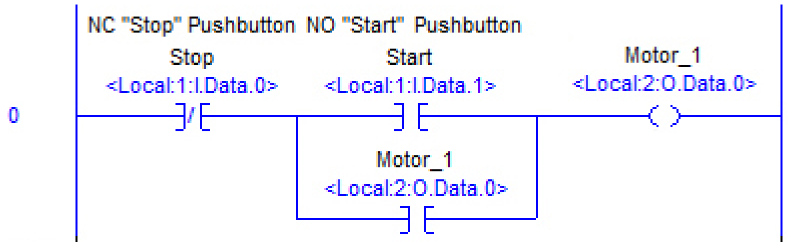
Note that, when ***Edit Tag*** is selected, it is the actual tag itself that is edited, not the specific instruction.

Reminder – the Output Module resides in slot 2 of the CompactBus. (**Local:2:O**).

Thus, the change of tag **Motor_1** to an alias tag will be reflected on all instructions that have been assigned the **Motor_1** tag, including the hold-in *XIC*.



45. Rung 0 of your ladder diagram should now display as shown below:



The program is almost complete! The required instructions have been placed on the rung and alias tags have been assigned to all of the instructions, allowing for their operation to be linked to that of the input and output modules, but there is one last item that needs to be addressed.

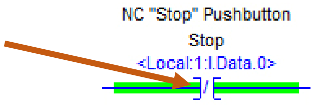
As mentioned in step 29, an *XIO* was initially chosen as the logic instruction for "Stop" because a Stop-Start controller utilizes a NC pushbutton in that position and the *XIO* resembles a NC contact. Although it makes the logic of the ladder rung "appear" similar to that of a physical Stop-Start controller, this actually presents a problem because our system contains a physical NC pushbutton that is wired to **Input 0** of the Input Module as its "Stop" button.

Within the structure of the ladder rung, the ***Stop*** instruction needs to remain (normally) **TRUE** when "Stop" is **not** being pressed and then transition to a **FALSE** state when the "Stop" **is** pressed.

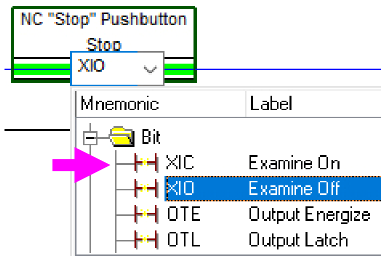
But the ***NC Stop button*** will constantly apply +24V$_{DC}$ to ***Input Port 0***, causing the Input Module to keep bit ***I.Data.0*** set to a "**1**" and, in-turn, causing the ***Stop XIO*** to remain **FALSE** whenever the ***Stop button*** is **not** being pressed. This is contrary to our needs.

On the other hand, the state of an ***XIC*** will remain **True** whenever its bit is held at a value of "1". Thus, in order to provide the correct logic for the Stop-Start controller as wired by compensating for the choice of a physical ***NC Stop button*** that holds bit ***I.Data.0*** at a "**1**" when ***not*** being pressed, a ***Stop XIC*** is required on the rung instead of a ***Stop XIO***.

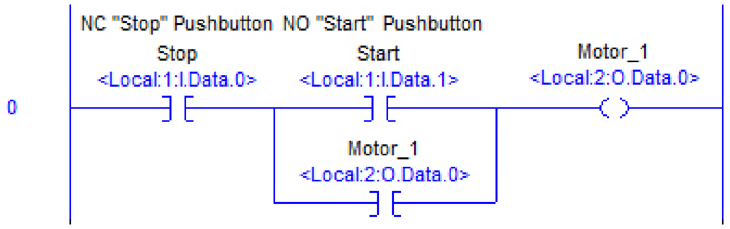
The ***Stop XIO*** can be changed to a ***Stop XIC*** by performing the following step:



46. ***Double-click*** on the actual instruction icon of the Stop *XIO*, expand the drop-down menu that will appear on the *XIO*, and ***double-click*** on the "**XIC Examine On**" option within the ***Bit*** folder to change the *XIO* to an *XIC*.



Once complete, the rung should now appear as:



Now the ***Stop XIC*** will return a **True** state when the ***NC Stop button*** is ***not*** being pressed since the unpressed ***Stop*** button will provide +24VDC to ***Input Port 0*** and bit ***I.Data.0*** will be a "**1**".

<u>Checking Your Program for Errors</u>

At this point, you have completed the initial ladder logic program that is required for the exercise. Your ladder diagram should be identical to the one shown above.

Remember that the presence of a column of **e**'s to the left of any rung indicates that the rung is still in "edit mode" and has not yet been fully configured. But, even if all rungs have been fully configured, there may still be some errors that remain unseen.

To have to ***RSLogix*** software check your ladder diagram for errors:

47. Locate and click the icon at the top of the ***RSLogix 5000*** main window.

If any errors are found, they will be reported as either "fatal errors" or "warning" in the field immediately below your ladder diagram at the bottom of the ***RSLogix 5000*** main window. Make the appropriate corrections until no errors are reported.

Before you proceed any further, you need to **save** an updated version of your project. And since you have completed the first program within this tutorial lab, choose the **Save As** option under the **File** menu in the ***RSLogix*** window and <u>rename</u> your project:

"Lastname_Controller_P1.ACD"

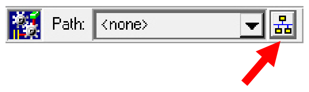
You are now ready to download your program into the PLC and verify its proper operation.

In order to download your program into the PLC, you must first set the communications "***path***" that the ***RSLogix*** software can use to locate your controller.

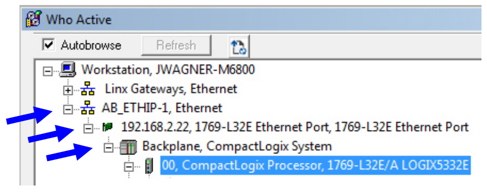
__Setting the Communications Path to the PLC__ **(Requires a physical system to continue)**

You must be at a lab bench with a physical motor control system to complete the remaining steps.

48. ***Click*** the to the right of the "**Path**" field in the main window.



49. In the "**Who Active**" window that appears, expand the "**AB_ETHIP-1, Ethernet**" item by clicking its ⊞ and keep expanding the revealed items until the "**00, CompactLogix Processor**" is located.

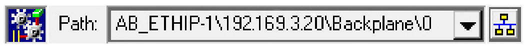


Workstation, JWAGNER-M6800
Linx Gateways, Ethernet
AB_ETHIP-1, Ethernet
192.168.2.22, 1769-L32E Ethernet Port, 1769-L32E Ethernet Port
Backplane, CompactLogix System
00, CompactLogix Processor, 1769-L32E/A LOGIX5332E

Note that the items in your list, such as the IP Address assigned to the controller, may differ slightly from those shown in the list to the right.

50. Select the "**00, CompactLogix Processor**" and ***click*** the **Set Project Path** button, and then close the "***Who Active***" window.

The path should now appear in the "**Path**" field as:



Path: AB_ETHIP-1\192.169.3.20\Backplane\0

Note that, once the correct path has initially been defined, future attempts to set the path can be completed by ***clicking*** the down-arrow on the right side of the "**Path**" field and selecting the correct path from the displayed list of recently set paths.

__Connection Status of the RS Logix Software to the PLC__

The ***RSLogix*** software on the desktop computer will either be in an "**Offline**" or an "**Online**" state with respect to the operation of the PLC.

Offline – the RS Logix software is not in direct (real-time) contact with the PLC.

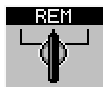
While offline, the ladder diagram displayed within the RS Logix window is completely independent of the program that is stored in the PLC's memory.

The ***RSLogix*** software must be "Offline" while creating or editing the ladder diagram because structural changes can only be made to the ladder diagram while in the offline state.

Online – the RS Logix software is communicating in real-time with the PLC. While online:

- the ladder diagram displayed within the ***RSLogix*** window is directly linked to the program that is stored within the PLC's memory such that the current state of the instructions and the values stored within the tag locations will be shown on the ladder diagram and updated in real time.
- the operator of the desktop computer has the ability to both <u>monitor</u> and <u>affect</u> the operation of the PLC.
- the ***RSLogix*** software will display the PLC's current operational state, which for this exercise will either be **Program Mode** or **Run Mode**.

Note that, when the key-switch is set to the **REM** (remote) position, the ***RSLogix*** software has the ability to remotely toggle the PLC between the Program and Run modes.



Operational State of the PLC

SAFETY NOTE – remotely switching the PLC between Program and Run modes can be **dangerous**, especially in a live industrial setting where unexpected motion or energization could pose a safety hazard to human life.

Program Mode (online) – the ***RSLogix*** software is communicating in real-time with the PLC but the PLC is not executing the program stored in its memory.

While in **Program Mode**, minor (non-structural) changes can be made to the program, such as the values stored in a tag location or specified in a MOV instruction.

Although the logic states of the ladder instructions will be displayed on the ladder diagram while in this mode, the PLC is configured such that all output ports will be false (off or de-energized) while in Program Mode.

Run Mode (online) – the ***RSLogix*** software is communicating in real-time with the PLC while the PLC is executing the program stored in its memory.

While in **Run Mode**, logic states of the ladder instructions will be displayed on the ladder diagram.

Although changes cannot be made to the ladder logic program while the PLC is in this mode, the PLC's operation, including the state of its outputs, can be directly affected by the operator of the desktop computer. (This will be discussed at a later time)

SAFETY NOTE – while using the PLC system in the lab, if the PLC system starts operating in an unexpected or unsafe manner when the PLC is switched from Program to Run mode, **DO NOT PANIC** and **DO NOT CLOSE THE RS LOGIX PROGRAM WINDOW**!!!

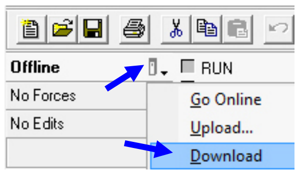
Simple **toggle the PLC back into Program mode** using the ***RSLogix*** software.

If this does not solve the problem, then **shut-off the main breaker** located on the Lab Volt power supply to de-energize the entire system.

Downloading Your Program into the PLC and Going Online with the Controller

Now that the "path" to the controller has been set within the ***RSLogix*** software, you can **Download** your program into the PLC's memory and "Go Online" with the PLC as follows:

51. *Click* the down-arrow just to the right of the "**Offline**" field in the upper-left portion of the main ***RSLogix*** window and select "**Download**" to begin the download process.

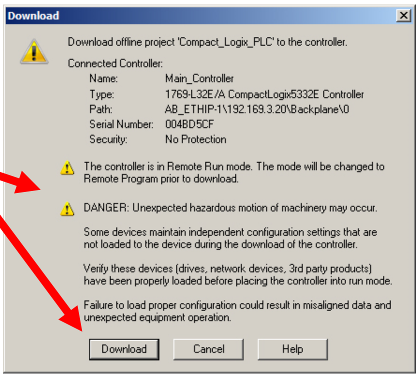


52. If a ***Connecting to Controller*** window appears (it probably won't) stating that you must first update the firmware to be able to connect to the PLC, click on "**Update Firmware**" and answer positively to any other Update windows that may appear until the update is complete.

53. If a "***Connected to Go Online***" window appears stating that "*The open project has changes that aren't in the controller*" appears, choose "**Download**".

54. A "***Download***" window will appear that will allow you to transfer your program into the PLC's memory.

Read the warning and then click "**Download**".


Download

Download offline project 'Compact_Logix_PLC' to the controller.

Connected Controller:

Name:	Main_Controller
Type:	1769-L32E/A CompactLogix5332E Controller
Path:	AB_ETHIP-1\192.169.3.20\Backplane\0
Serial Number:	004BD5CF
Security:	No Protection

The controller is in Remote Run mode. The mode will be changed to Remote Program prior to download.

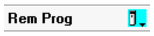
DANGER: Unexpected hazardous motion of machinery may occur.

Some devices maintain independent configuration settings that are not loaded to the device during the download of the controller.

Verify these devices (drives, network devices, 3rd party products) have been properly loaded before placing the controller into run mode.

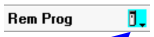
Failure to load proper configuration could result in misaligned data and unexpected equipment operation.

Download Cancel Help

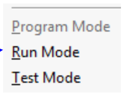
55. Once the ***download*** is complete, the status of the PLC will either switch to ***Remote Program*** mode 

or a warning window will appear asking if you want to "***Change Controller mode to Remote Run?***"

If that window appears, click "**Yes**".



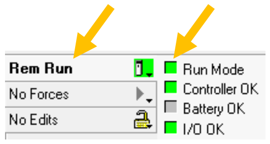
Otherwise, click the down-arrow next to "**Rem Prog**" and select "**Run Mode**" from the list of menu items.

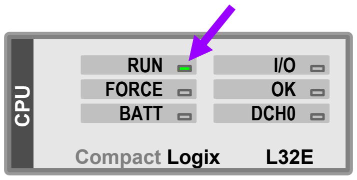


This will cause the PLC to begin executing (running) the downloaded program.

Operational State of the Motor Control System

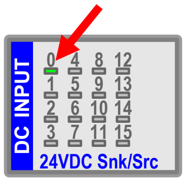
The PLC should now be in "**Run**" mode and the ***RSLogix*** software should be "Online" with the PLC and configured for "**Rem Run**" operation.



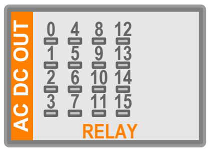


The "**Run**" state of the PLC is also indicated by means of a small green LED that is located on the CPU of the physical PLC, as can be seen by looking through the plexiglass cover of the PLC module that is mounted within the LabVolt bench.

Note that the states of the various input and output ports of the PLC are also indicated by means of a series of (green or orange) LEDs that are located on the input and output modules of the physical PLC.



Whenever +24VDC is detected at one of the input ports, the LED associated with that port will illuminate green, and whenever one of the output ports is switched on, the LED associated with that port will illuminate orange.



Initially, only the LED associated with **input port 0** will be illuminated because a NC (Stop) pushbutton is wired to that input, such that the button will provide +24VDC to that input when it is not being pressed (i.e. – in its normal position).

The LEDs on the physical input and output modules can be invaluable when troubleshooting the operation of a PLC-based motor controller. For example, if this system does not start-up when **Start** is pressed, this could be due to either an electrical problem or a logic (programming) problem.

Since the **Start** button is normally-open, if the LED associated with the input port to which the button is wired does <u>not</u> illuminate when **Start** is pressed, then there is most likely an electrical/wiring problem with that part of the control circuit.

But if the input LED <u>does</u> illuminate, indicating that the input detects +24VDC when **Start** is pressed, and the LED associated with the output to which the field coil is wired does <u>not</u> illuminate, then the cause is most likely a logic (programming) problem.

And if the LEDs associated with the Start button's input and the field coil's output <u>both illuminate</u>, then there is most likely an electrical/wiring problem with either the output module and field coil or the main power connection to the motor via the contactor's main contacts.

Additionally, the ***RSLogix*** software also provides some features that can be invaluable when troubleshooting the operation of a PLC-based motor controller. While the ***RSLogix*** software is "Online" with a PLC, the current operational state of the various instructions contained in that PLC's ladder logic program is indicated on the ladder diagram that is displayed within the ***Routine Editor*** window, such that:

> TRUE **logic instructions** (XICs) will be highlighted in **green** and any Boolean **output instructions** (OTEs) whose assigned **bit** is a "**1**" will also be highlighted in **green**.

Initially, when the PLC is switched to **Run** mode, only the ***Stop XIC*** should be highlighted **green**.

But, if the **Start** button is pressed, the ***Start XIC*** should be highlighted **green** while the button is being held-in. At this point, the rung-condition will be TRUE, causing the ***Motor_1 OTE*** to set its assigned bit to a "**1**", in-turn causing both ***Motor_1 OTE*** and the ***Motor_1 XIC*** to also be highlighted **green**.

When the **Start** button is released, the green highlight will be removed from the ***Start XIC***, but the other instructions will remain **green** because the "hold-in" XIC keeps the rung condition TRUE.

After **Start** is pressed and released, if **Stop** is pressed, the ***Stop XIC*** should no longer be highlighted **green** while the button is being held-in, and since pressing **Stop** will cause the rung condition to become FALSE, the OTE will reset its assigned bit to a "**0**", in-turn causing the green highlight to be removed from both ***Motor_1 OTE*** and the ***Motor_1 XIC***.

And finally, when the **Stop** button is released, ***Stop XIC*** should be highlighted **green** again, and the system is back to its initial condition.

Verify the Proper Operation of the Motor Control System

Verify the operation of the motor control system by utilizing the **Stop** and **Start** buttons to determine if the system performs properly.

While you are testing the system, pay attention to both the operation of the physical setup and the instructions displayed on the ladder diagram within the ***RSLogix*** window.

For example, **verify** that:

> When you press the **Start** button, the contactor's main contacts should immediately actuate, causing Motor #1 to accelerate quickly up to its synchronous speed.
>
> Pressing **Stop** should cause the contactor to dropout and Motor #1 to be de-energized.
>
> The motor should remain de-energized while **Start** and **Stop** are pressed simultaneously.

Be sure to test to all of the system's specified operational characteristics. If any errors are detected or if anything is not working properly:

- Toggle the PLC from **Run Mode** ⇨ **Program Mode**
- Toggle **RSLogix** from **Program Mode** ⇨ **Offline**
- Troubleshoot your program
- Download the updated program and try again!

This ends part D of the experiment.