



PDHengineer.com

Course Nº IC-3002

Programmable Logic Controllers

This document is the course text. You may review this material at your leisure before or after you purchase the course. If you have not already purchased the course, you may do so now by returning to the course overview page located at:

<http://www.pdengineer.com/pages/IC-3002.htm>
(Please be sure to capitalize and use dash as shown above.)

Once the course has been purchased, you can easily return to the course overview, course document and quiz from PDHengineer's My Account menu.

If you have any questions or concerns, remember you can contact us by using the Live Support Chat link located on any of our web pages, by email at administrator@PDHengineer.com or by telephone toll-free at 1-877-PDHengineer.

Thank you for choosing PDHengineer.com.



17350 State Highway 249 | Suite 140 | Houston, TX 77064

Programmable Logic Controllers

Robert J. Scoff, PE

Copyright © 2007, Robert J. Scoff

Table of Contents		Page
1.	Introduction	4
2.	The PLC System	5
	2.1 Input Devices	5
	2.2 Input Modules	5
	2.3 The Processor	6
	2.4 Output Modules	7
	2.5 Output Devices	8
	2.6 Block Diagram of a PLC System	8
3.	Ladder Logic Programming	9
	3.1 Examine If Closed	9
	3.2 Examine if Open	9
	3.3 Output Energize	10
	3.4 Logix Pro	10
	3.5 Download the Program	10
	3.6 An One Rung Program	11
	3.7 An One Rung 'AND' Ladder Logic Program	11
	3.8 A Logic 'OR' Ladder Logic Program	12
	3.9 Start Stop Circuit	12
	3.10 Documentation	13
	3.11 Labeling the Ladder Logic Program	13
	3.12 The Garage Door Opener	14
	3.13 The Silo	14
	3.14 Timers	15
	3.15 Counters	16
4.	Some (But Not All) General Considerations	18
5.	The Garage Door Program	19
6.	Conclusions	19

List of Illustrations

Page

Figure 1.1	Physical Layout of a Typical PLC System	5
Figure 2.1	Symbols for Some Typical Digital Input Devices	5
Figure 2.2	Layout of Typical Input Module Showing Input Devices	6
Figure 2.3	Simplified Block Diagram of the Memory Map of a Typical PLC	6
Figure 2.4	Layout of a Word in a Typical PLC	7
Figure 2.5	Push Button Switch Connected to Input Module 1, Terminal 0	7
Figure 2.6	Layout of a Typical Output Module Showing Output Devices	8
Figure 2.7	Information and Control Signal Flow in a PLC System	8
Figure 3.1	The Examine If Closed Instruction	9
Figure 3.2	The Examine If Open Instruction	9
Figure 3.3	Output Energize Instruction	10
Figure 3.4	Getting Started With a One Line Program	11
Figure 3.5	'AND' Circuit Done With Ladder Logic	11
Figure 3.6	'OR' Circuit Done With Ladder Logic	12
Figure 3.7	Schematic of Hard Wired Start Stop Circuit	12
Figure 3.8	Ladder Logic Rung for a Start Stop Circuit	13
Figure 3.9	A Timer Example	15
Figure 3.10	An Example of an Up Counter	17
Figure 3.11	An Example of a Down Counter	18
Figure 5.1	A Workable Garage Door Opener	20

1. Introduction

It would be hard to imagine today's industrial world without **Programmable Logic Controllers (PLC,s)**. Before PLC's, most of the industrial processes in the world were controlled by relays and ladder logic. This worked quite well, because everyone who was concerned with industrial control systems knew all about them. They did have drawbacks. Although relays generally worked well, they would sometimes fail. But the technicians and electricians of the time understood relays and ladder logic well enough that they could get the systems to function again relatively quickly. They were also slow, but the control system designers of the time would take that into account. In more complicated systems, a condition called '**Relay Race**' could occur. One big disadvantage was that when any changes were needed by the process, the ladder logic would have to be rewired to implement the changes. This was usually a big problem. Then, the changes had to be documented. This was usually a bigger problem. Remember, this was before Auto Cad. The master drawings were done on velum. Someone had to find them, change them, copy them, and throw out the old, now wrong, copies, and make new copies to replace the outdated versions.

In 1968, someone at Hydromatic Division of General Motors said that there had to be a better way. So the thought of using computers entered the picture. The initial conditions that the new computer controlled system had to follow were:

- A. The new system would adapt to change.
- B. Be able to be placed on the factory floor and keep functioning.
- C. Take up no more space than the relay ladder logic control systems.
- D. Be maintained by plant electricians.

There was resistance to changing a system that everyone knew how to operate. But Richard Morley of Bedford Associates was contracted by General Motors and invented the PLC and called it a **MODULAR DIGITAL CONTROLLER**. The **MODICON** was the first PLC. At first, the PLC's were relatively simple, having only digital inputs. Since that time, many other functions have been added to the PLC, including the ability to handle analog signals.. A partial list of what they do consists of relay operations, timing, counting, data moves, comparisons, math functions, and other advanced operations.

We can't really talk about the PLC itself without working on the **PLC system**. The PLC system consists of 5 basic parts. They are **Input Devices, Input Interface, Processor, Output Interface, and Output Devices**. These 5 basic parts will be described in detail.

Another thing that needs to be looked at is **Digital Input/Outputs** and **Analog Input/Outputs**. Digital Inputs and outputs have nothing to do with digital computers. They are merely inputs or outputs that have 2 states or conditions, like on and off. Since digital computers work on 2 states, 0 and 1, these 2 state input/outputs are referred to as digital input/outputs. Analog input/outputs are continuously variable, over a range. Examples are process variables, such as pressure, water depth, flow, temperature, ect. They are usually translated to an electrical variable such as voltage or current. Common analog input/outputs are 0 to 10 Volts DC and 4 to 20 milliamps DC.

Before we do more on PLC's. it would help to look at the physical layout of a typical PLC. Let's look at Figure 1.1. This shows the physical layout of a typical PLC as it would be purchased. This happens to be modeled after an Allen Bradley Rockwell Automation SLC 500. The power supply is necessary to operate the processor and the input and output modules. Note that for this particular PLC, the

processor is in a special place called slot 0. The other 7 slots can be either input or output modules. The input and output modules can be either digital or analog modules. It is possible to have more than 8 slots, but for the purpose of this course, we will limit the system to 8 slots.

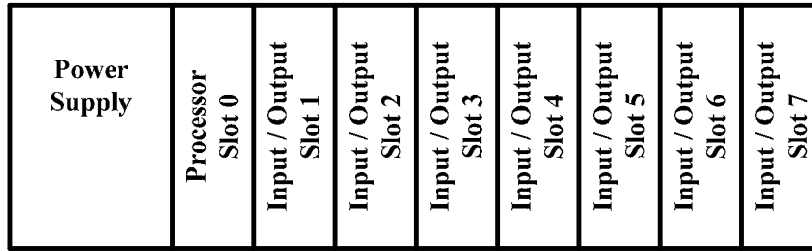


Figure 1.1 Physical Layout of a Typical PLC System

Finally, actual programming will be done on a on-line simulator. This simulator is found on line at thelearningpit.com. Pick the option with the 15 day free trial. The free 15 day option has certain limits, but it will work for this course. If you find it a lot of fun to do PLC programming, and want to do more, the cost of the full scale simulator is \$30.

2. The PLC System

2.1 Input Devices

The first thing that any control system needs is a way to get information into the system. For most industrial control systems, digital inputs are push button switches, selector switches, limit switches, proximity switches, and contacts from relays. The important thing about all of these devices is that they are either off or on.

Analog inputs are the other type of input device. This course will not cover analog inputs in any great depth. We will say that analog inputs are usually 0 to 10 volts or 4-20 milliamps (ma). Even the 4-20 ma inputs are turned into a 1 to 5 volt or 0.4 to 2 volt by the input module. So we can say that an analog input is looked on as a voltage range by the processor. Figure 2.1 shows the symbols for some digital input devices.

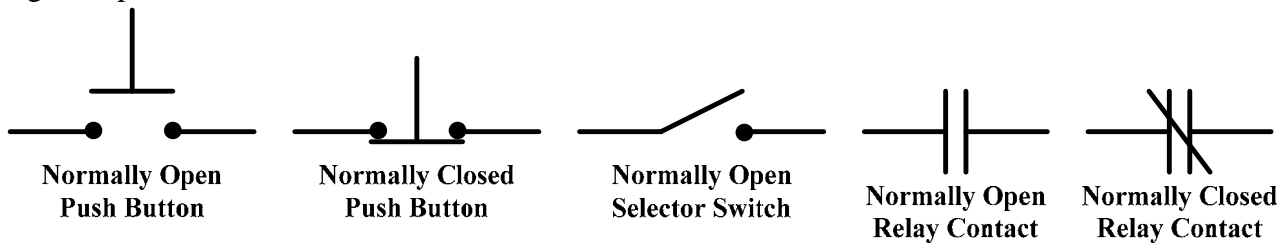


Figure 2.1 Symbols for Some Typical Digital Input Devices

2.2 Input Modules

Input modules separate the input devices, voltages, and conditions from the processor. Input devices are subjected to a lot of abuse. The input module takes the signal from the input device and sends that information to the processor. It also protects the processor. There are isolation devices in the input modules that do the separation. A typical input module would have 16 inputs, corresponding to the 16

bits of the words of the special purpose computer which is the processor. There could be a number of input modules in a typical PLC. These modules fit in slots that are designed to accept them. The slots will accept input or output modules and are part of the PLC. Figure 2.2 shows the layout of a typical input module with input devices connected. Only 4 inputs are shown for simplicity, but 8 or 16 inputs per module is more standard.

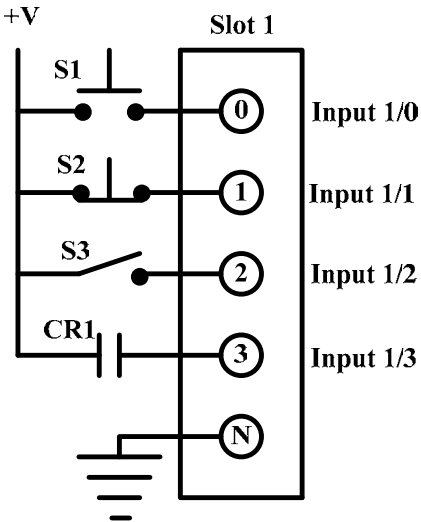


Figure 2.2 Layout of Typical Input Module Showing Input Devices

2.3 The Processor

The processor is a 16 bit special purpose computer. It consists of a number of words separated into sections. When using PLC's it helps to have a little understanding of how the words of this special purpose computer are arranged.

Output Table
Input Table
Status Table
Timer Table
Counter Table
Number Table

Figure 2.3 Simplified Block Diagram of the Memory Map of a Typical PLC

To help in this area, let's look at Figure 2.3. It shows a simplified memory map of a typical PLC memory. Each section is a number of 16 bit words. Each word has an address, and each bit of each word also has an address. As an example, let's look at the first word of the input table, as shown in figure 2.4.

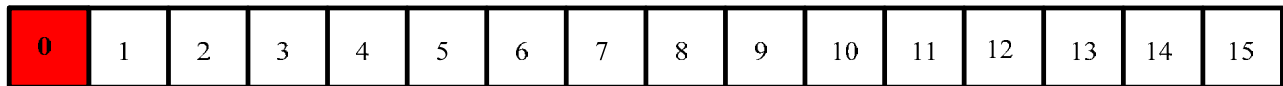


Figure 2.4 Layout of a word in a typical PLC

There needs to be a way to identify each bit of each word of this little computer. If this is an input word, the first bit could be labeled [I:1/0]. That address is very important. The letter I tells us that we are working with an input. The 1 tells us that this is the number 1 word in the input file and also the input module located in slot number 1 in the PLC. The 0 tells us that we have the number 0 bit. That particular bit is shaded in red in Figure 2.4. Notice that we are counting from 0 to 15 instead of from 1 to 16. This is in deference to the way that a computer works. An important thing to notice now, is that each bit of the input word I:1/0 is connected to a terminal on an input module. Therefore, the input module can have up to 16 terminals, each one of which can be connected to an input device such as a switch or relay contact. Figure 2.5 shows how this is done. Figure 2.2 is also an example of this.

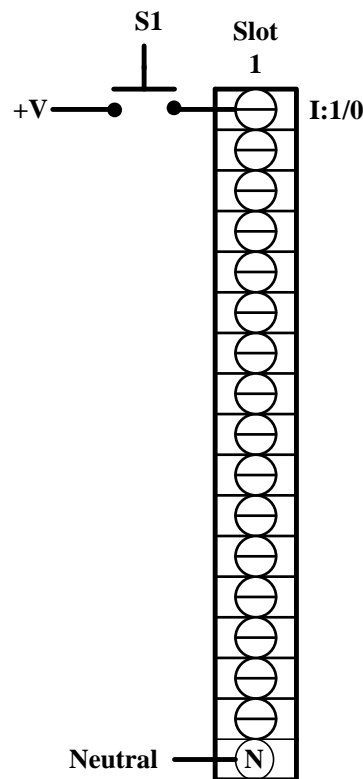


Figure 2.5 Push Button Switch Connected to Input Module 1, Terminal 0

2.4 Output Modules

Output modules look a lot like input modules. A big difference is that output devices are connected to output modules. Typical output devices include lights, relays, starters, and solenoids. Figure 2.6 shows an output module connected to various loads. Notice that the output devices all require some power to operate them. Output devices are usually fused to protect the output modules in case of a short circuit. The fuses are shown in this typical layout to emphasize their importance. Without these individual fuses, a failure of one output device could cause an entire system to shutdown, or burn out

an output module. Notice that slot number 2 was chosen for the output devices. Since, in our examples, we have used slots 0 and 1, slot number 2 is the first available slot in our imaginary system.

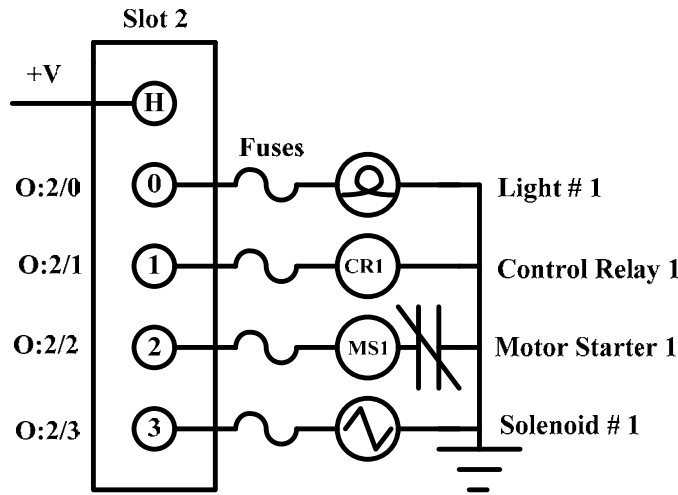


Figure 2.6 Layout of a Typical Output Module Showing Output Devices

2.5 Output Devices

We have already shown typical digital output devices. Again, digital means devices that have only two states. In this case the two states are off and on. Figure 2.6 shows lights, control relays, motor starters, and solenoids. These four devices cover most of the digital devices used in the industrial control world.

2.6 Block Diagram of a PLC System.

Now that we've seen the 5 parts of a PLC system, let's look at how they are connected. Figure 2.7 shows this.

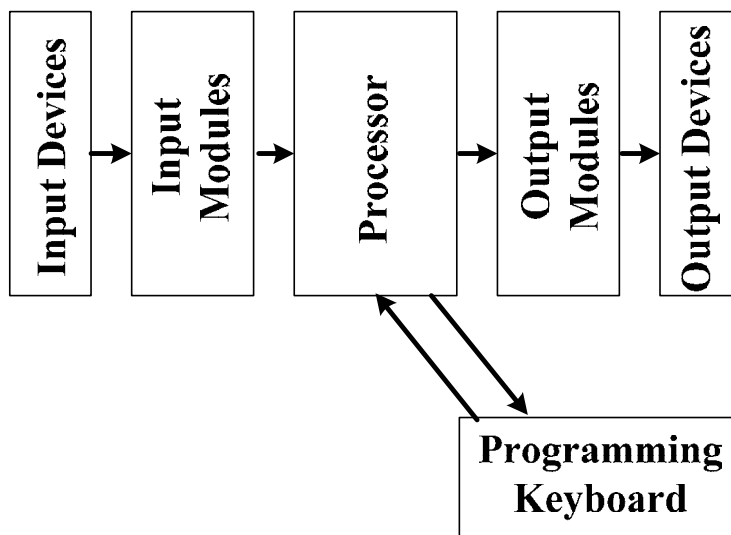


Figure 2.7 Information and Control Signal Flow in a PLC System

What really happens in our PLC system is that the input devices send information to the processor, the processor takes that information, does something with it depending upon a program in the processor, and sets the output devices. Looked at in this manner, it's a pretty simple device. With that in mind, let's see how to program this device.

3. Ladder Logic Programming

3.1 Examine if Closed

There is a new language that we have to learn. Fortunately, to do PLC programming we only need to learn three words right now. The first word is **EXamine If Closed**, or **XIC**. The symbol for this new word is shown in Figure 3.1.

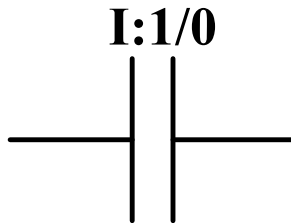


Figure 3.1 The Examine If Closed Instruction

In the programming that is used in PLC's, the XIC instruction is used to either pass logic or not pass logic. Instead of using the words '**EXamine If Closed**', let's use the words '**Pass Logic If Closed**'. Now notice that the instruction has an address associated with it. It is an input address. It refers to an input device connected to an input module. If we look at Figure 2.2, it has a normally open pushbutton switch connected to input number I:1/0. So we can say that the XIC instruction shown in Figure 3.1 is associated with that **Normally Open** pushbutton switch at Input Number I:1/0. The instruction actually represents the push button switch. So why don't we just call the instruction **Normally Open**, instead of **EXamine If Closed**. The reason is that the instruction is now passing logic. So if the pushbutton is closed and we **EXamine If Closed**, the instruction will pass logic. Another way to think of this is instead of saying **EXamine If Closed** every time you see the symbol of Figure 3.1, say instead **Pass Logic If Closed**.

3.2 Examine If Open

The second word in our new language is **EXamine If Open** or **XIO**. The symbol for the second word in our new language is shown in Figure 3.2.

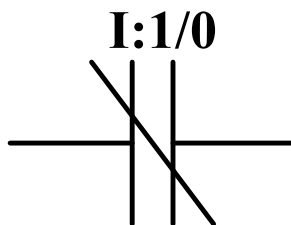


Figure 3.2 The Examine If Open Instruction

The **Examine if Open** instruction is used just like the **Examine if Closed** instruction. It will pass logic if the device connected to the input associated with it is open. If we use I:1/0 from figure 2.2, the symbol in Figure 3.2 will pass logic if the switch is open. Hence, **Examine if Open** is used if we want to pass logic when the switch connected to Input 1/0 is open. We can say '**Pass Logic If Open**' in place of '**Examine If Open**'.

3.3 Output Energize

The third word to learn right now is **Output Energize**. Output energize is used anytime we want to turn an output on. Its symbol is shown in Figure 3.3.

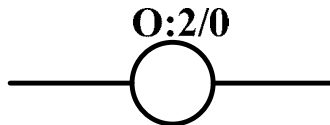


Figure 3.3 Output Energize Instruction

Something to notice now is that the **Output Energize** instruction is associated with an output device. For example, Output 2/0 from Figure 2.6 is controlling the light (Light number 1) connected to Output 2/0. So, if we energize output 2/0, Light number 1 turns on. With these three instructions, we are now ready to do some simple programming.

3.4 Logix Pro

To do some simple simulations, we will now download a free 15 day free trial simulator. This is one of the best PLC simulators available today. Just go to the website <thelearningpit.com> , find Logix Pro PLC Simulator, Download Now, and Download Now again on the next page. You will download a 5.1 Mega Byte file. It is free for 15 days. If you choose to buy it, the cost is \$30. The free version has a few limitations, but will not affect anything we do in this course. One of the biggest disadvantages is that without paying for the software, you cannot save any of the programs that you write.

3.5 Download the Program

After downloading the program, open it. You should see one of the simulations on the left, and a place to write a ladder logic program on the right. Next, find **Simulations** in the upper left hand corner. Under **Simulations**, find **I/O Simulator**. This should open 2 columns of input switches, and 2 columns of output lights on the left hand side of the screen. The first and third columns will be switches or inputs, and the second and fourth columns will be lights or outputs. For right now we will work with columns 1 and 2.

Just to see what happens, put the cursor on the first switch, I:1/0. Left clicking on it will change its state. When it changes state the address will light up telling you that that particular input is now on. Real PLC's do something very similar. When an input switch is closed, a light on the input module turns on to let you know that the switch is closed. Now put the cursor on the switch again. Right click on the switch several times to see the different kinds of switches available. You should be able to find selector switches, normally open pushbuttons, normally closed pushbuttons, and limit switches.

Now put the cursor on the output O:2/0. That would be the first light in the second column. When you left click on the light nothing happens. That is because the only way to change the state of the outputs is to use the program to do it. When you right click on the outputs, you are given the option to change the color of the lights. Try these things and then we will make several one line programs to turn lights off and on.

3.6 An One Rung Program

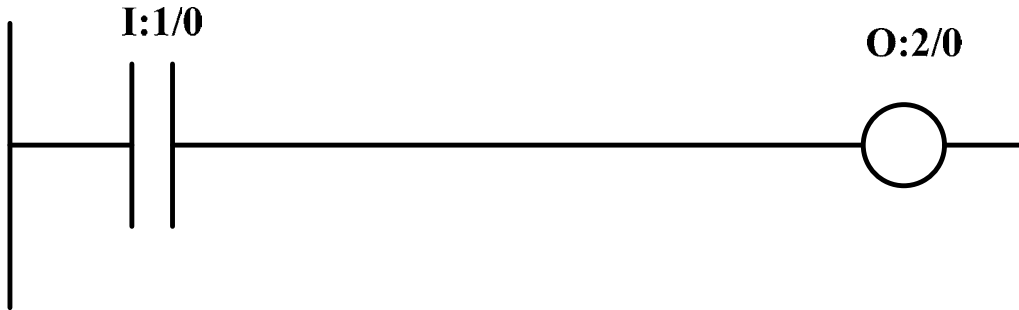


Figure 3.4 Getting Started With a One Line Program

Figure 3.4 shows a one rung Ladder Logic program whose only function is to turn on the light connected to Output 2/0 when the switch connected to Input 1/0 is closed. Because this is a computer, things have to be done in a very specific manner. In the center top of the screen, just below the crossed wrench and hammer icon, there is a rectangular box. Click on 'User'. Find the XIC icon and grab and drag it down to the screen directly below the rectangular box. When a green square shows up on the line, drop the XIC. Then grab an Output Energize icon and drop it the same way on the green square. You should have something similar to Figure 3.4. To enter the addresses, double click on the XIC icon, and a '?' should appear. When it does, simply type in I:1/0 and then Enter. Do the same thing with the Output Energize and type in O:2/0, and then Enter. You should get a line just like Figure 3.4.

Now, we have to follow a certain procedure to download and test our program. This same procedure is followed for a one line program, or a 100 page program, or anything in between. Find the small looped arrow in the upper right hand side of the rectangular box. Click on it. The box will change. Click on Download. The screen will flash several times. Wait until it stops, and then click on Run. This has to be done in that sequence. Now you should be able to make output O:2/0 turn on and off by turning input I:1/0 on and off. If you did, you have just made a PLC ladder logic program. This ladder logic program had only one line, but that was on purpose to give you an idea of how it all works. Let's now do some more complicated (but not too much more) logic.

3.7 An One Rung 'AND' Ladder Logic Program

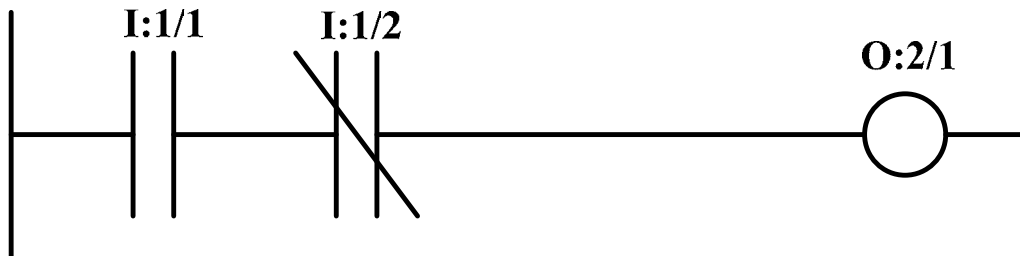


Figure 3.5 'AND' Circuit Done With Ladder Logic

Figure 3.5 shows an XIC instruction in series with an XIO instruction. This is an AND circuit because both elements have to pass logic to turn O:2/1 on. Lets go back to our Logix Pro program. We need to first get back into the program mode. Then, use the small looped arrow to change the rectangular box to element selection mode, and choose 'User'. Then grab an XIC instruction, an XIO instruction, and an Output Energize instruction, and build a rung like Figure 3.5. Then use the small looped arrow to get to the download box, do the download, and go to run mode. Make switches 1 and 2 be selector switches. As you turn switches 1 and 2 off and on, watch what happens to the light number 1 (O:2/1).

3.8 A Logic 'OR' Ladder Logic Program

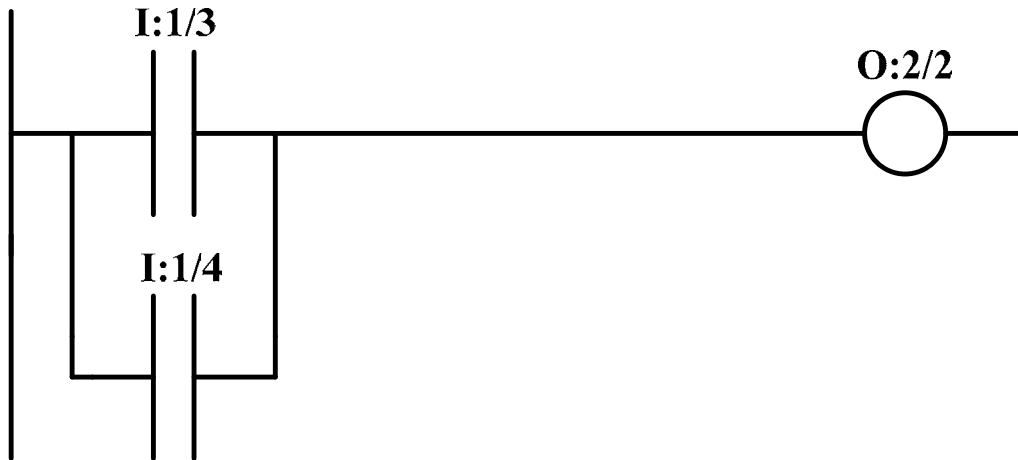


Figure 3.6 'OR' Circuit Done With Ladder Logic

To make an OR logic function, we need to use a new instruction called a **Branch** instruction. It is located just to the left of the XIC instruction. Grab it and drag it down to next line in your ladder logic program. Now grab 2 XIC's and put them in the two branches as shown in Figure 3.6. Grab an Output Energize and put it at the end of the rung as before. Do the Download – Run routine again, and test your OR circuit. Closing either input or both inputs will turn on output O:2/2.

3.9 Start Stop Circuit

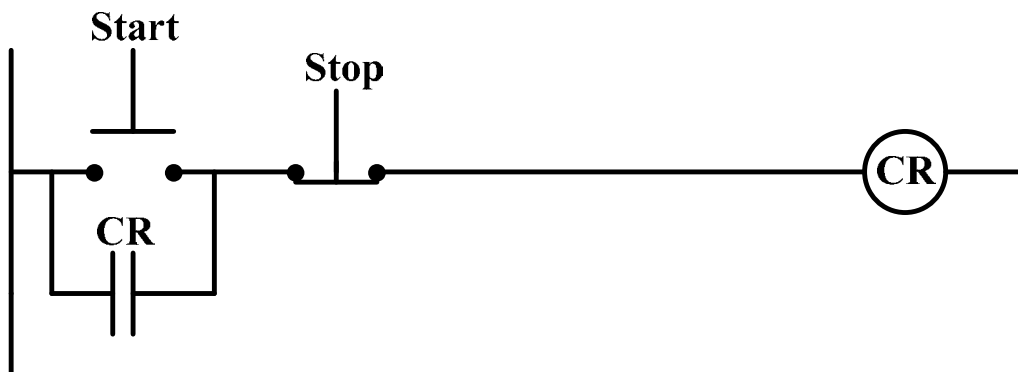


Figure 3.7 Schematic of Hard Wired Start Stop Circuit

Figure 3.7 is a schematic of a standard hard wired Start Stop circuit. When the Start pushbutton is pushed, current flows through the Start and Stop switches. This causes coil CR to be energized. If CR is a relay with a normally open contact, that contact can be placed in parallel with the Start pushbutton. Then when the coil of CR is energized, the contact CR pulls in and allows current to continue flowing to the coil of CR. This is called a seal-in circuit. Let's do the same function with a PLC ladder logic

program. Continuing where we left off in the PLC program that we're writing, let the Start pushbutton be Input Number I:1/5 and the Stop pushbutton be Input Number I:1/6. Control relay CR will be Output Number O:2/3. Right click on Input Number I:1/5 until it is a Normally Open pushbutton. Then make Input Number I:1/6 be a Normally Closed pushbutton. Then program a ladder logic rung as shown in Figure 3.8.

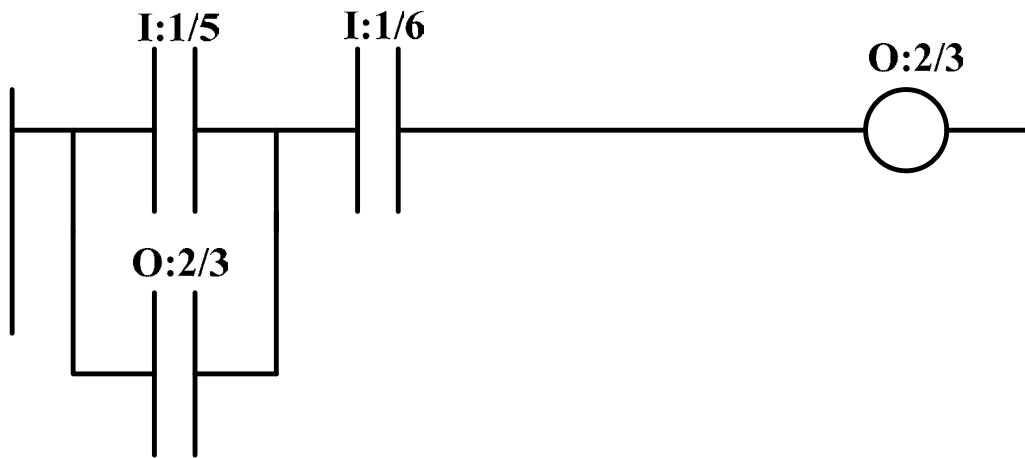


Figure 3.8 Ladder Logic Rung for a Start Stop Circuit

There are several things to notice about this rung. First, the Stop pushbutton is represented by a XIC. That is correct. The Stop pushbutton is normally closed, so we want it to pass logic if it is closed. The XIC only becomes open if the Stop pushbutton is pushed. The next thing to notice is that the Output Energize has a contact associated with it. The XIC contact labeled O:2/3 is similar to a relay contact. We still have to use the new word in our language, EXamine If Closed. Then when Output O:2/3 is energized, the XIC contact, O:2/3, passes logic. The next step is to add the rung shown in Figure 3.8 to the ladder logic program that we are writing, and test it to see if it really is a Start Stop circuit.

From here on you can make more complicated ladder logic programs. There are several things that need to be understood. First, notice that no matter what you did the program generated an 'END' statement. That is because when this kind of ladder logic program works, it does the rungs one at a time in sequence, starting at the top and ending at the END statement. When the program hits the End statement, it updates all of the outputs, and then goes to the first rung and repeats the operation. It continues this loop as long as the controller is in the Run mode. Since it is a digital computer, it has to do things in sequence. The length of time that it takes to do a complete loop is called the cycle time. It is important to notice this because in some operations the order in which rungs are done can influence the results.

3.10 Documentation

As with most things in the industrial control world, documentation is very important. For PLC programming lists of inputs and outputs are first made. Then each input and output is assigned an address. This address is also the address on the input and output modules, and quite often the wire number. Doing this part of the PLC system properly helps everyone, from the person doing the parts layout, the wiring person, and the programmer. When the engineer does his part properly at the beginning, everyone's job is made easier.

3.11 Labeling the Ladder Logic Program

So far we have done very simple things that we could keep track of in our heads. This is not usually the case. Ladder Logic Programs can be many pages long. There is just too much to keep track of easily. So now we want to label the individual symbols in a program, as well as the rungs.

To give each symbol a name, right click on the symbol, and a table with 7 options will drop down. Left click on 'Edit Symbol'. A space will appear directly below the symbol where you can type in whatever you want to identify that symbol. After you type what you want to type there, hit enter. If you use that particular address anywhere in the program, the software will label it for you. Note that you must be in the program mode to edit symbols.

To edit rungs, you must also be in the program mode. The first step is to left click on the rung number. A red field will appear around it. Then right click on the red field. A table with 6 options will appear. Choose 'Edit Rung Comment', and a place to put information about that rung will appear. Simply type in your comment, and hit enter.

Go ahead and do this for your program, and when you are ready we will make a garage door opener work.

3.12 The Garage Door Opener

To get ready for the Garage Door Opener, go to program mode and erase all of the rungs that you have programmed. You wouldn't be able to save your work unless you have paid for the software. One way to erase is to left click on each rung number, then right click on the red area that appears, and cut each rung. When your ladder logic area only has an END statement left, go to simulations, and open 'Garage Door'. Make the left hand side of the screen big enough to show the entire door. Do this by grabbing the vertical line in the center of the screen between the picture of the door and the program. Notice that the door has 5 Inputs. They are Open, Close, Stop, Top Limit Switch, and Bottom Limit Switch. There are also 5 Outputs. They are Ajar Light, Open Light, Shut Light, Motor Up, and Motor Down.

Now is the time to have some fun. Using the ideas that we have discussed so far, make the garage door open and shut. I have included a program that will work at the end of this course. Note that even a simple process like this could have many possible solutions. If you don't use the limit switches to turn the motor off at the top and bottom, the motor burns out. Sometimes you can get the motor to burn out even if you use the limit switches. You can speed up, or slow down the operation by finding the speed control on the right hand side of the Download screen. Moving the pointer will cause the door to move faster or slower. After you have gotten the Garage Door to work in a satisfactory manner, erase your work (or save it if you have purchased the software) and load the Silo simulator. You can either erase the program one line at a time or do the following sequence. Step 1, Go to the file menu. Step 2, click on 'New'. Step 3, On the screen that appears, click 'OK'. Your program should disappear into digital wonderland.

3.13 The Silo

The Silo is a bit more difficult to make work properly. Go to simulations, and open the Silo simulation. Do not worry about the A-B-C switch. Notice that the inputs consist of Start, Stop, Prox Sensor to detect the box, and Level Sensor to detect box full. The outputs consist of Run, Fill, and Full lights, and a Fill Solenoid, and a Motor Run output.

This program spills pink stuff on the floor if you don't do it just right. The trick is to make the full light work right. The full light turns on when the box fills up, and stays on until the box leaves the prox switch. See what you can do with this box loading program.

After you struggle with this a bit, there is a program that will work in the software package. Look in the file menu, under open. Then look in Logix Pro, and open Silo. A workable program will download into the programming area. Download it and go to Run mode. The program will run. It is only 4 steps long.

3.14 Timers

Timers, before PLC's, were big, expensive, not very accurate, and not very flexible. After the advent of PLC's, they became small (just three words in a memory somewhere inside the PLC), cheap (just three words in memory), accurate (To within the time base of the PLC, as small as 0.01 seconds), and flexible (we'll get into that now).

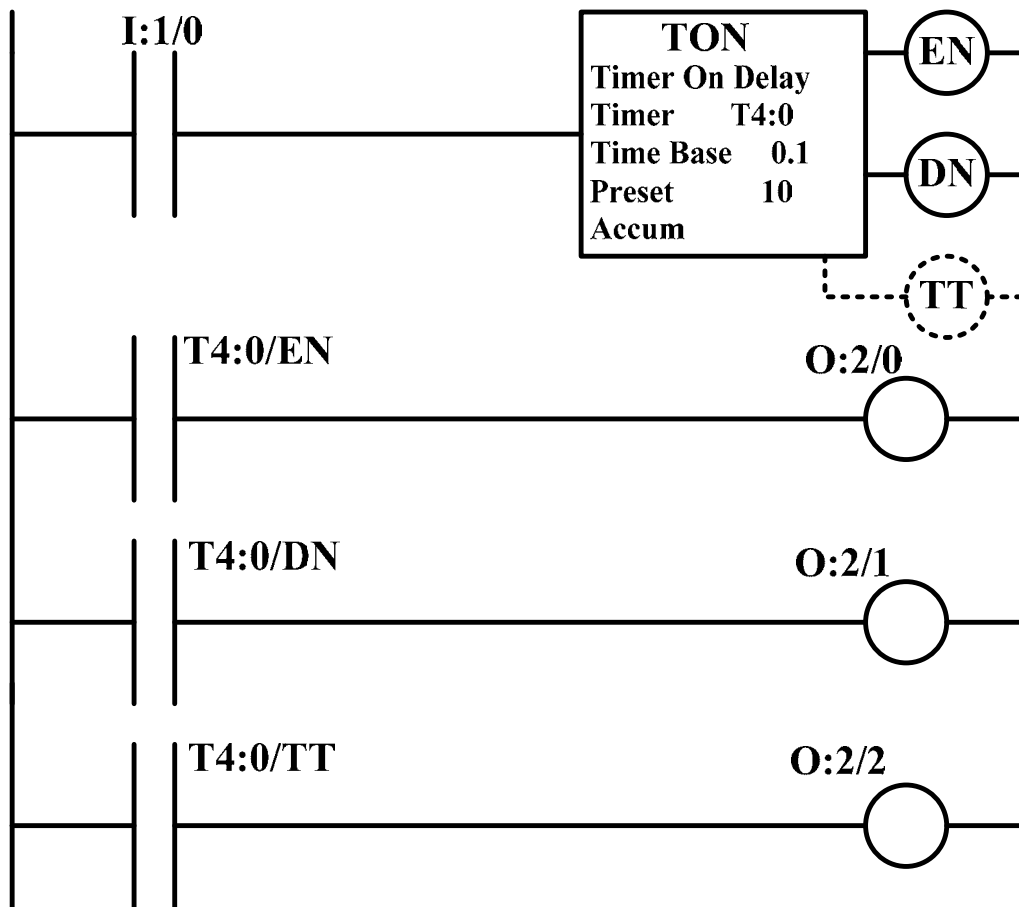


Figure 3.9 A Timer Example

The easiest way to understand a timer is to build a ladder logic program that uses one. Let's start by erasing all that we've done so far and loading the I/O simulator again. Then build the ladder logic program shown in Figure 3.9. To do that, first find the Timer/Counter tab to the right of the user tab that we have been using. Click on it and drag a **On Delay Timer** (TON) to the first line in your new program. Notice that the screen shows EN and DN outputs. This stands for **ENABLE** and **DONE**. There is also a third output that is not shown called **TIMER TIMING** (TT). These outputs can be

used just like output energize outputs. In the PLC they are actually three bits that are part of the three words that make up the timer. To the PLC, they look just like outputs. To program the **On Delay Timer**, first click on the '?' where the Timer Number is located. Hit 'Enter' and the PLC will generate an address for you. If you use more than one timer, and the software gives the same number again, just change it to the next one in sequence. The time base 0.1 seconds should appear in the next line. This is important because the timer actually counts time bases to get the total time. The next number, **PRESET**, is programmed by the programmer to tell the timer how many time bases to count. So, the PRESET times the TIME BASE gives the total timer time. The last line, **ACCUM**, tells how many time bases have been counted when the timer is timing. The timer is actually working as a counter of time bases.

Now let's see what happens to **EN, DN, and TT** when the timer is activated. The timer starts timing when the input to it is made logically true. In our example, that happens when the switch represented by XIC contact I:1/0 is closed. After copying the example from Figure 3.9, download the program, and go to Run mode. Make I:1/0 a normally open pushbutton switch. Then hold it closed and watch the timer. The Accumulator will count from 0 to 10. Enable (EN) will be on as long as you hold the pushbutton switch closed. Done (DN) will turn on when the accumulator equals the preset. This is really your **time delay on operate**. Notice here that the enable output looks just like a relay or Output Energize. The **Timer Timing** output turns on whenever the accumulator is changing value. The TT output is very useful in some applications. This output was not usually available when timers were big, expensive, inaccurate, and inflexible. If you want to make the preset bigger to help you watch the timer operate, try 30 or 50 for the Preset.

If the input to the timer is stopped before it times out, EN turns off, TT turns off, and DN never turns on. The accumulator then resets, and the timer is ready to operate like a time delay again. If the input to the timer is held true, EN stays on, DN stays on, and TT goes off.

There is another Time Delay on Operate called **RTO** or Retentive Timer on Operate. The big difference is that the Accumulator does not automatically reset when the input goes away. It retains its value and keeps accumulating only when the input is logically true. To make the accumulator reset, there is an instruction called RES or reset. When RES goes logically true, it resets the accumulator.

The next timer is called Timer Off Delay or **TOF**. It starts out with the Accumulator equal to the Preset. As soon as the input goes logically true, the accumulator is set to zero, the Enable output goes true, and the Done output goes true. The Timer Timing output stays off or false. When the input then goes false, the Enable output turns off, the Timer Timing output turns on, and the accumulator counts up until it equals the Preset. Then Done and Timer Timing both turn off and the accumulator stays at the Preset value. I recommend that you just change the TON to a TOF. One way to do this is to delete the TON and replace it with a TOF in your program. One way to delete a timer is to right click on it, and cut it out of the rung. Then run the program (after you download it) and watch the outputs change as you change the input.

3.15 Counters

The next useful function that most PLC's can do is to count events of one sort or another. These work very much like timers. Let's build an example right below the timer that we just built to see how it operates. Add the four rungs shown in Figure 3.10, and we will test a counter. **RES** means Reset and is one of the instructions in the Timer/ Counter set.

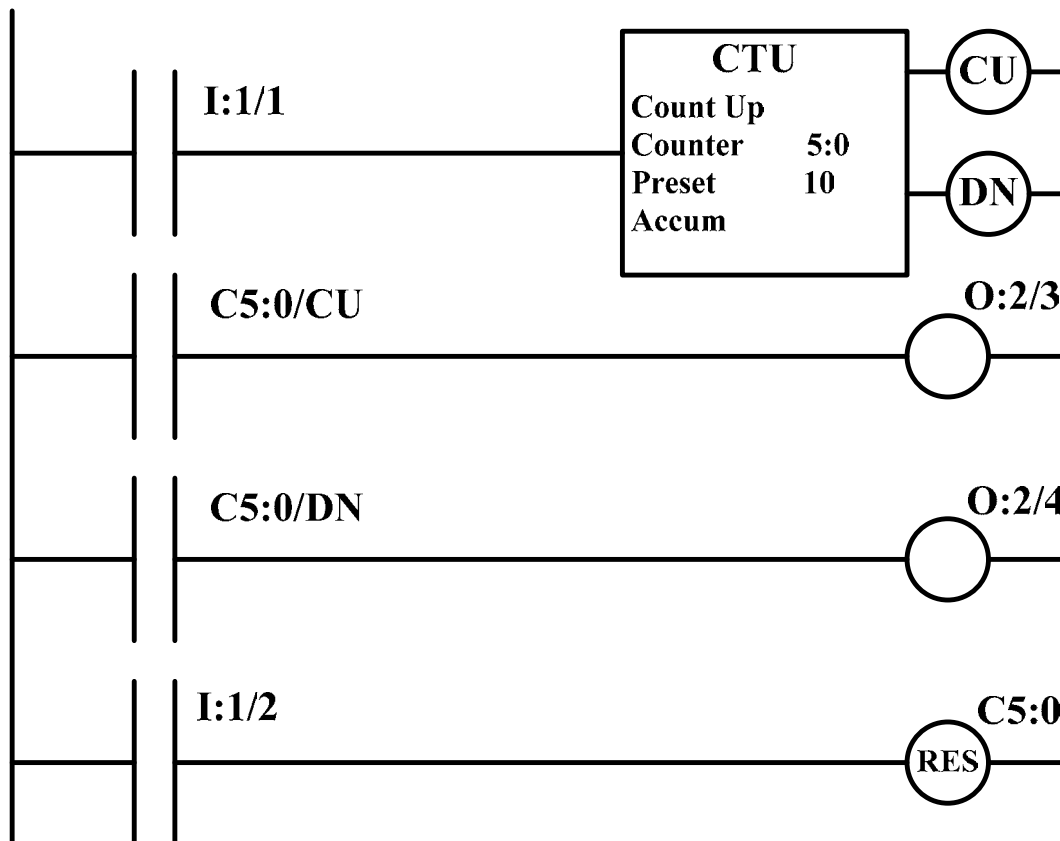


Figure 3.10 An Example of an Up Counter

After programming this counter (Remember, that after programming, download and put the processor in the Run mode), make sure that Inputs 1 / 1 and 1 / 2 are normally open pushbuttons. Then push pushbutton 1 / 1 10 times while watching outputs CU, DN, 2 / 3, and 2 / 4. Notice that Accum now equals 10. This makes Accum equal to the Preset. You set the Preset when you programmed the Up Counter. Now press Input 1 / 1 again and notice that the Accumulator keeps incrementing and DN stays on. It will do that until a condition called overflow occurs. That happens when the Accumulator equals 32767. That is as high as the 16 bit word can count if one bit is reserved for the plus or minus sign. When that happens, an output bit is set. It is called OV, for overflow. It can be used in the program as **C5:0/OV**. We didn't test it here because we would have to count to 32767 to do that. It should be noted that the Accumulator can be set to any value between -32768 and +32767 while in the programming mode. Reset, however, always resets the counter to zero.

Up Counters can be used to count events, such as boxes of product shipped. A counter can be set to cause lubrication after a certain number of operations. They can be used, in conjunction with a timer, to determine the rate of a process. Actually using these capabilities is limited only by the imagination.

Another counter is called the Down Counter. It counts down, instead of up. Let's do an example of a down counter. Put your program in the programming mode and add the Down Counter shown in Figure 3.11.

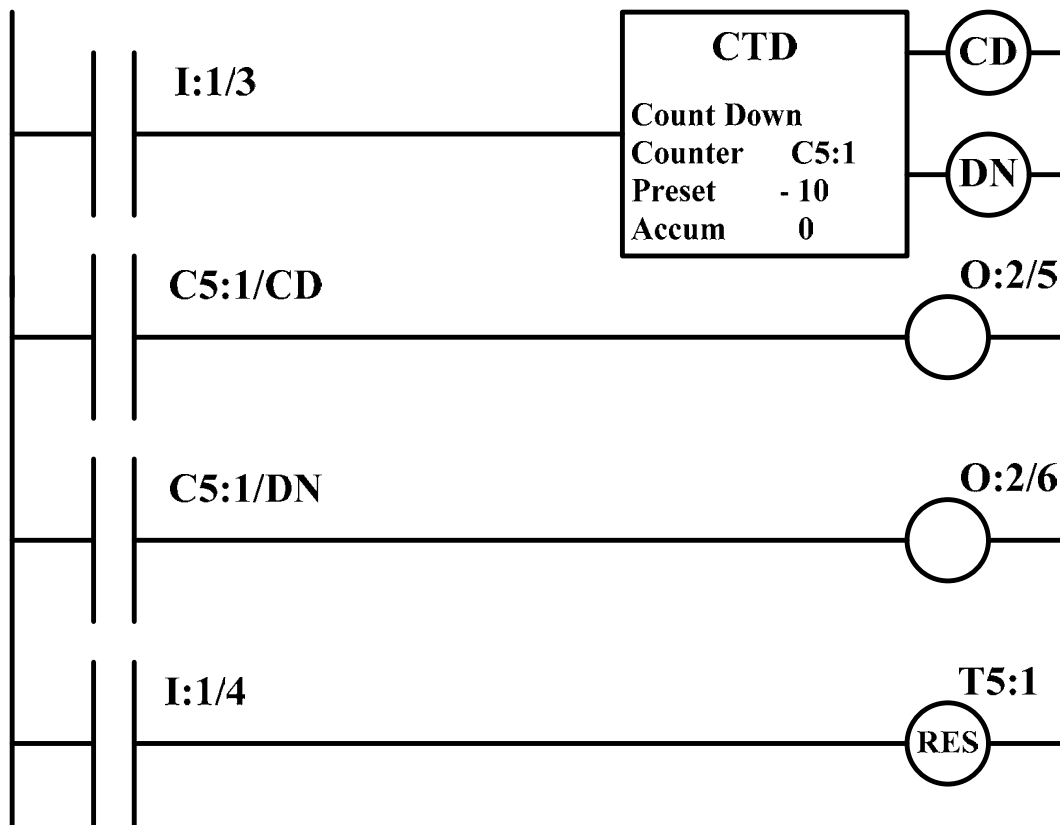


Figure 3.11 An Example of a Down Counter

The Down Counter works a little differently than an Up Counter. One of the big differences is that the Accumulator starts at zero and counts down. Also, DN is **ON** whenever Accumulator is greater than or equal to preset. DN only turns **OFF** when Accumulator becomes less than preset. Activating Reset, turns the Accumulator value to Zero. The Accumulator value can also be set to any integer between -32768 and +32767 while in the programming mode. However, it cannot be reset to anything other than zero without going to programming mode. A condition called underflow (similar to overflow) occurs when the counter goes below -32678. The underflow condition would be indicated by '**UF**'. For the down counter in Figure 3.10, the instruction '**C5:1/UF**' could be used by the program to indicate an underflow condition.

4. Some (But Not All) General Considerations

Something that we haven't emphasized that is very important is the sequence of operations of the **P**rogrammable **L**ogic **C**ontroller. The PLC is a small special purpose digital computer. As such, it works just like a computer, and only does one step at a time. It does these steps pretty quickly, but they are still done one step at a time, and in sequence. So, in the programs that we have written, the computer executes the first rung in your ladder logic diagram. It then executes the second rung, then the third, and so on, until it gets to the **END** statement. When it gets to the END statement, it changes any Outputs that have changed as the program was being run through one cycle. Notice that the Outputs do not change until the program has run through the entire ladder logic program. This is so fast, that most times it does not affect the program. But there are times when something can happen on one line, and then be negated on another line. This could lead to confusion. If we look at the counters, we can put a Reset just below the counter, and reset it with the same input that it is trying to count.

The counter Accumulator would never get above zero. This is just one of the things that need to be addressed when programming a PLC.

Another thing that is very helpful in programming PLC's is to be able to look at the words in the memory that represent the Inputs and Outputs. To do that click on the crossed wrench and hammer icon near the top center of the screen. A screen will appear that shows the output table in binary form. With 0 meaning off and 1 meaning on, you can determine what all of the outputs are at a glance. The same thing can be done for the inputs. Just change the table from Output to Input. There are other tables that can be examined from this menu. For example, we can look at the condition of the timers and counters by changing the table to Timer or Counter. Notice that the Radix of the data can be changed. Go ahead and see what options are available.

There are many math functions included in the PLC. The use of them is beyond the scope of this course. The same is true of the compare functions. Look at the compare functions available. To determine what any instruction will do, go to the help menu, click on Rockwell RsLogix Instruction Help, and then click on whatever instruction that you need help on. A good description is given of how to use them, and what they do.

5. The Garage Door Program

As promised earlier, a workable garage door opener program will be shown. A special definition of door ajar was made. The door is ajar when it is stopped somewhere in the middle. This program implements that definition. One of the nice things about PLC programming, is that if you change the definition of ajar, you only need to change the programming, not the hard wiring. What would you change to make the Ajar light turn on whenever the door is not completely open or completely closed? Go ahead and try it and see what happens. Notice that this program takes into account the possibility of pushing the Up button when the door is going Down, and vice versa. It prevents the door from trying to go in the opposite direction when it is moving. Figure 5.1 shows this program.

If you haven't made the garage door opener work, just go ahead and copy this one and test it. Then, just for the fun of it add timers so that you have to hold the button for 3 seconds before the door opener operates. This can be done on both the up and down directions. Then put a counter in the program that counts number of operating cycles. Then, lock the door down after a number of open close cycles. This shows the flexibility of PLC programs, and how operations can be changed, without changing hardware.

For the program of Figure 5.1, there were no labels put on the elements or the rungs. To make the program more understandable, add your comments to the elements and rungs. As programs get more and more complicated, these comments are more and more useful.

PLC's can be a lot of fun.

6. Conclusions

This course is intended to be a start in your understanding of how PLC's actually work. If you get a real machine problem in the future, you now have a good start on understanding what you need to do to make your automatic process work. In closing, most of the problems with PLC systems have little to do with the actual PLC. About **60 % of the problems are loose wires**, and most of the rest of the

problems are **Input Devices**. The PLC itself is relatively trouble free. However, having an idea of how to read and understand ladder logic diagrams is a great help in finding the area of your automatic system that is causing problems at any given time.

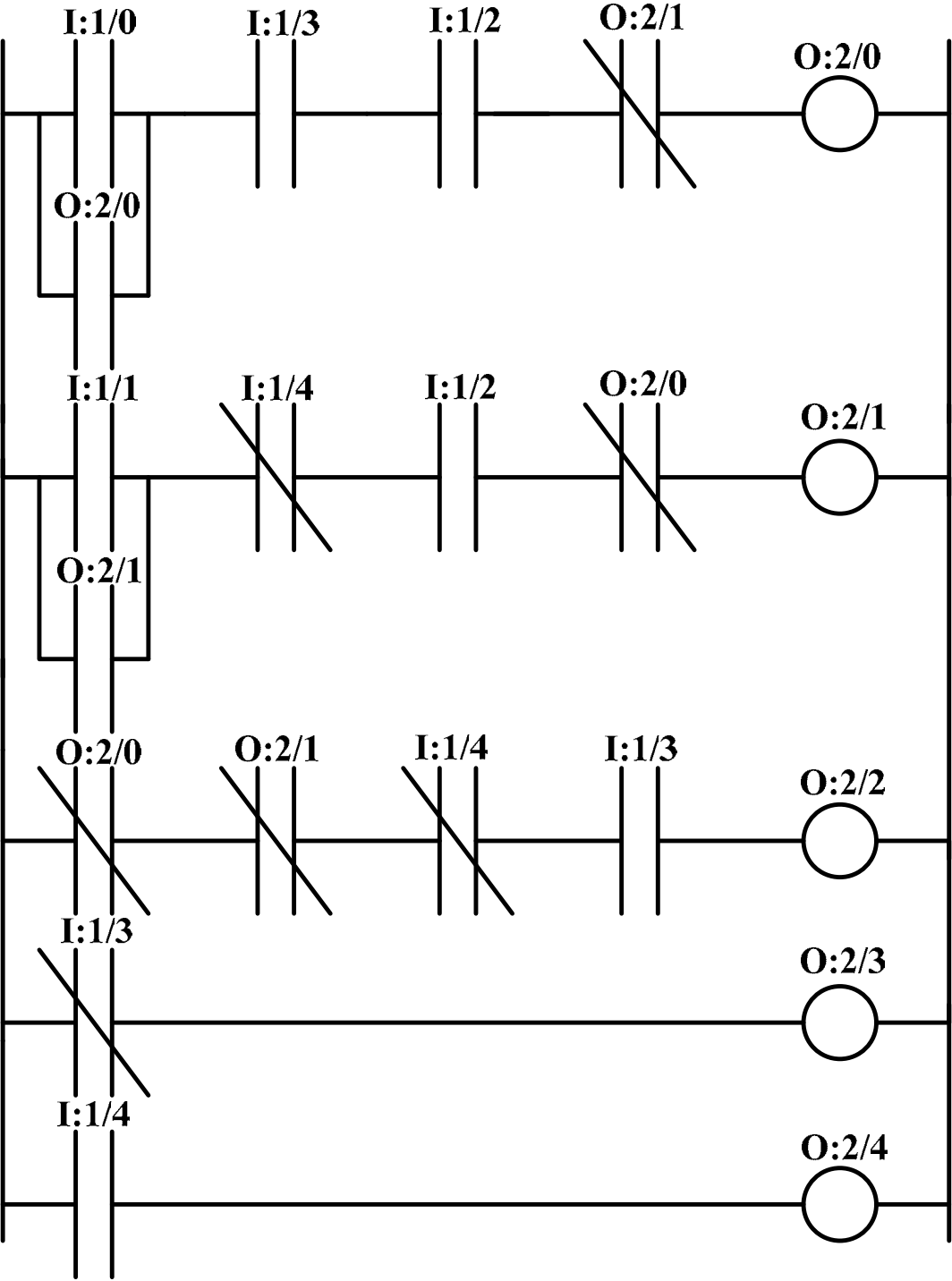


Figure 5.1 A Workable Garage Door Opener