101 BASICS SERIES

LEARNING MODULE 24:
PROGRAMMABLE LOGIC CONTROLLERS (PLCs)

Cutler-Hammer

EATON

WELCOME

Welcome to Module 24, which covers **Programmable Logic Controllers**, or PLCs. The Programmable Logic Controller (PLC) was invented in the 1960s to replace the sequential *relay circuits* traditionally used in machine control. A PLC is a solid-state, electronic device that controls the operation of a machine. It uses *logic* functions, which are programmed into its memory, via programming software.

Almost any "real world" application that needs electrical control needs a PLC. In fact, whether you work in machining, packaging, material handling, automated assembly, or countless other industries, you are probably already using a PLC.

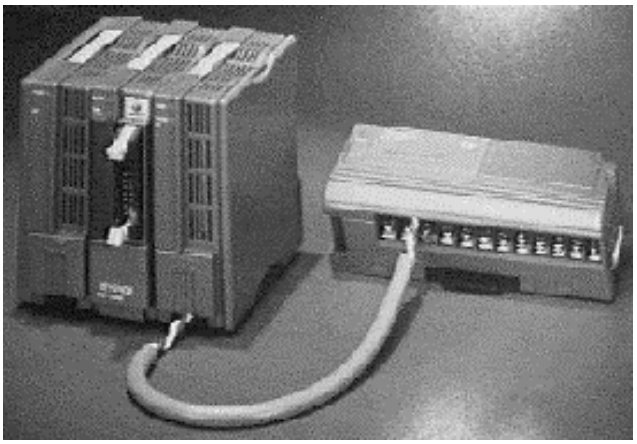


FIGURE 1: A PROGRAMMABLE LOGIC CONTROLLER

Like the other modules in this series, this one presents small, manageable sections of new material followed by a series of questions about that material. Study the material carefully, then answer the questions without referring back to what you've just read. You are the best judge of how well you grasp the material. Review the material as often as you think necessary. The most important thing is establishing a solid foundation to build on as you move from topic to topic and module to module.

A Note on Font Styles

Key points are in bold.

Glossary terms are underlined and italicized the first time they appear.

Viewing the Glossary

You may view definitions of glossary items by clicking on terms and words that are underlined and italicized in the text. You may also browse the Glossary by clicking on the Glossary bookmark in the left-hand margin.

WHAT YOU WILL LEARN

In this module, we'll **step through each of the following topics** in detail:

WHAT YOU WILL LEARN (CONTINUED)

A BRIEF HISTORY

PLCs were introduced in the late 1960s **to take on the role previously played by sequential relays in machine control.** *Relays* are placed onto a single panel to provide a special control circuit referred to as logic or relay logic. The purpose of a logic circuit is to allow an event, such as the starting of a motor, to occur only if predetermined conditions are met.

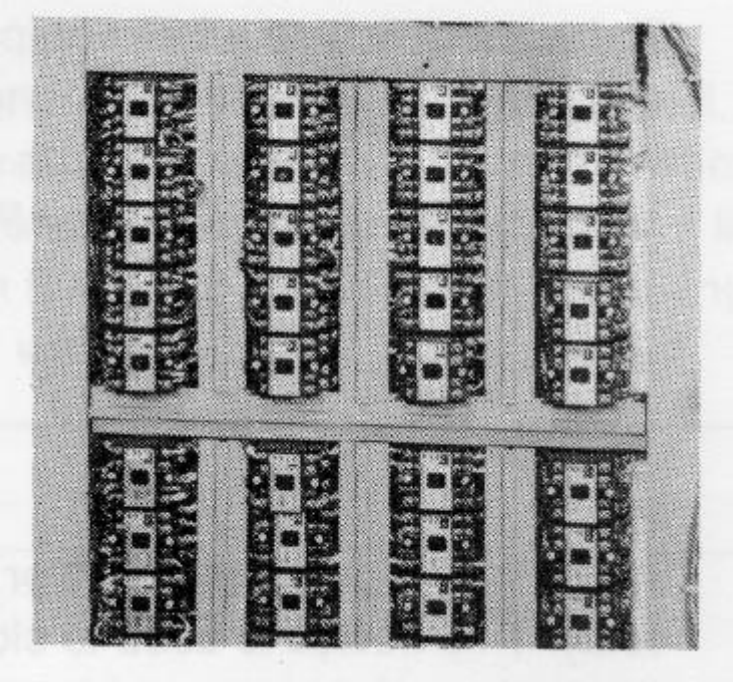


FIGURE 2: A RELAY PANEL

Although relay circuits performed their job well, they could be very expensive to install and maintain. In fact, the primary reason for designing PLCs was **eliminating the large cost of replacing complicated relay-based machine control systems**. Picture a machine control panel that included hundreds or thousands of individual relays. The size could be mind boggling. How about the complicated initial wiring of so many individual devices? These relays would be individually wired together to yield the desired outcome. As I'm sure you can imagine, such a complicated system brought with it many problems.

When production requirements changed, the control system had to be updated. If frequent changes were required, system updates became very expensive. Since relays are mechanical devices they also have a **limited lifetime, requiring strict adhesion to maintenance schedules. Troubleshooting was also time consuming** with so many relays involved.

A BRIEF HISTORY (CONTINUED)

To be a cost- and time-efficient replacement for relays, PLCs needed to be easy for maintenance and plant engineers to program, their lifetime had to be long, and they had to survive the harsh industrial environment. That's a lot to ask! The answers lay in using a programming technique—Relay Ladder Logic—based on the relay technology people were already familiar with, and **replacing mechanical parts with solid-state ones.**

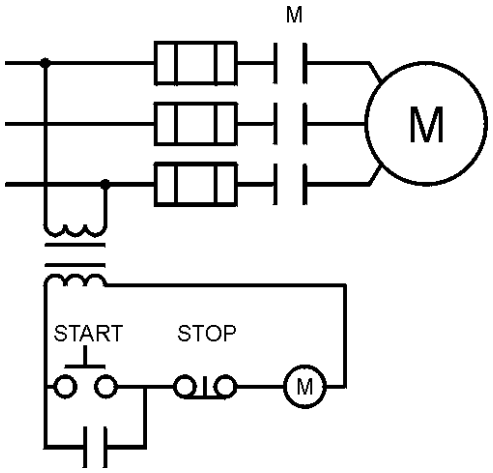


FIGURE 3: TRADITIONAL RELAY LOGIC

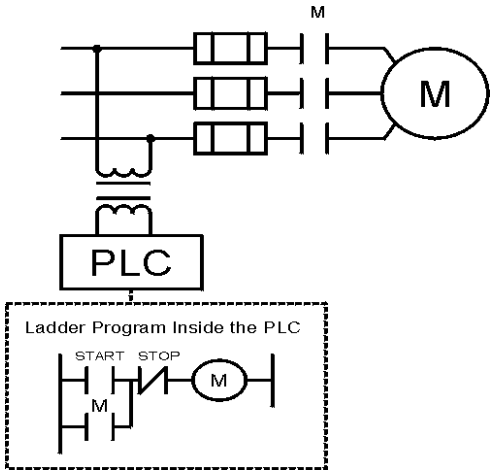


FIGURE 4: PLC LOGIC

In the early 1970s, the dominant PLC technologies were sequencer-state machines and the *bit*-slice based *Central Processing Unit* (CPU). Initially, conventional microprocessors lacked the power to solve PLC logic quickly in all but the smallest PLCs. However, as conventional microprocessors evolved, larger and larger PLCs were based upon them.

Communications abilities began to appear around 1973. The PLC could now talk to other PLCs and could be far away from the machine it was controlling. Because PLCs could also now be used to send and receive varying voltages, they were able to enter the *analog* world. But despite these advances, lack of standardization coupled with continually changing technology still made PLC communications a nightmare of incompatible protocols and physical networks. The 80s, however, saw an attempt to standardize communications. PLCs also got smaller in size and became software programmable through symbolic programming on personal computers (previously, PLCs had required dedicated programming terminals or handheld programmers). Today, the world's smallest PLC is about the size of a single control relay!

The 90s have seen a **gradual reduction in the introduction of new protocols**, and the modernization of the physical layers of some of the more popular protocols that survived the 1980's. **The latest standard (IEC 1131—3) has tried to merge PLC programming languages under one international standard.** We now have PLCs that are programmable in function block diagrams, instruction lists, C, and structured text all at the same time. Personal Computers (PCs) are also being used to replace PLCs in some applications. What will the future bring? Only time will tell.

PLC/RELAY COMPARISON

To see how far we have progressed since the time of the relay, consider the chart below. It summarizes the value of the PLC over the relay.

Relays	PLCs
• Large complicated systems that take up a lot of space	• One PLC can control a large system. Takes up less floor space than a relay-based system.
• Hardwired devices used to configure relay ladder	• Only the input and output devices are hardwired. The inner working of the PLC are solid-state
• Difficult to modify or update program	• With the programming software it is simple to write a new program (or modify an existing one) and then download it into the PLC
• Limited mechanical life	• The PLC, itself, is a solid-state device. It has a very long life and requires little maintenance
• Require separate hardwired timers and counters	• Counters and timers are internal, solid-state, devices

IN THE WORKPLACE

To see the real-world benefits of using a PLC, let's look at a batch plant operation where two ingredients (A and B) are added into a tank in a specified proportion and properly mixed and conveyed to another area of the process.

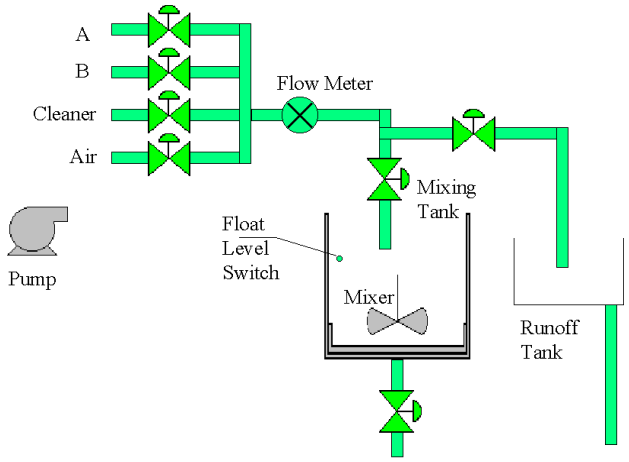


FIGURE 5: BATCH PLANT OPERATION

Two additional input lines are required, one for cleaning solution and one for air. All 4 lines are valve-controlled into one common pipe with a flow meter (1 pulse output per gallon). The other side of the flow meter connects to a Y pipe configuration, where each leg has a valve. One leg goes to the mixing tank, and the other goes to a runoff (or wastewater) area. As an example, let's assume that the goal is to mix 420 gallons of A with 280 gallons of B and send the mixture to the next process area. As in any process of this type, there needs to be a safety level float switch in the tank to shut down the process and sound an alarm if a certain level is exceeded. With a relay-based system, the sequence of events to control might look like something like this:

1. Open valve for cleanser, other 3 closed, tank inlet valve closed, run-off valve open.
2. Start pump, measure 50 gallons flow of cleanser.
3. Turn off cleanser valve, turn on air flow for 5 seconds.
4. Open valve for A, open tank inlet valve, close run-off valve.
5. Start pump, measure 420 gallons flow of A.
6. Turn off valve A, close tank inlet valve, open run-off valve.
7. Open cleanser valve, start pump, and measure 50 gallons flow of cleanser.
8. Turn off cleanser valve, turn on air flow for 5 seconds.
9. Open valve for B, open tank inlet valve, close run-off valve.
10. Start pump, measure 380 gallons flow of B.
11. Turn off valve B, close tank inlet valve, open run-off valve.
12. Start the tank mixer motor and run for 5 minutes.
13. Open cleanser valve, start pump, and measure 50 gallons flow of cleanser.
14. Turn off cleanser valve, turn on air flow for 5 seconds.
15. Once mixing is complete, open valve at tank outlet to allow discharge of mixture.

Pretty complicated and time consuming, wouldn't you say? With a relatively small, inexpensive PLC, on the other hand, all of this process can be controlled with the following I/O configuration:

1. **10 digital outputs (one for each of the 7 valves, 1 for the pump motor, 1 for the mixer motor, and 1 for the alarm)**
2. **2 digital inputs (1as a counter input from the flow meter and 1 as a safety level float switch in the tank).**

15 steps with relays, or two steps with a PLC. Guess why most businesses prefer to use PLCs instead of relay-based systems.

REVIEW 1

Answer the following questions without referring to the material just presented. Begin the next section when you are confident that you understand what you've already read.

1. Put these developments in the history of PLCs in the correct order:

_____ A. Standardized communications

_____ B. Programmable mechanical devices with limited lifetime

_____ C. Ability to communicate

_____ D. International standards

_____ E. Introduction of microprocessors

2. List three benefits of using PLC control instead of relay control.

_____

_____

_____

HOW A PLC WORKS

A PLC works by continually scanning a program. We can think of this scan cycle as consisting of 3 important steps: checking input status, executing the program, and updating output status.

Step 1—**CHECK INPUT STATUS**—**The PLC takes a look at each input to determine if it is on or off.** In other words, is the *sensor* connected to the first input on? How about the second input? How about the third? etc. It records this data into its memory to be used during the next step.

Step 2—**EXECUTE PROGRAM**—**The PLC executes your program one instruction at a time**. Maybe your program said that it should turn on the first output if the first input was on. It already knows which inputs are on/off from the previous step. Therefore, it will be able to use the state of the first input to decide whether the first output should be turned on. It will store the execution results for use later during the next step.

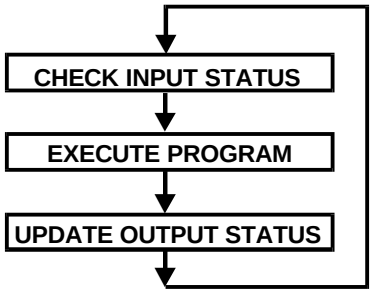


FIGURE 6: A TYPICAL SCAN

Step 3—**UPDATE OUTPUT STATUS**—Finally, **the PLC updates the status of the outputs based on which inputs were on during the first step and the results of executing your program during the second step**. Using the example in step 2, it would turn on the first output because the first input was on and your program said to turn on the first output when this condition is true. After the third step, the PLC goes back to step one and repeats the steps continuously.

One scan time is defined as the time it takes to execute the 3 steps listed above.

Example

Let's say that we have the following program in our PLC, where M is a motor *starter* that controls a conveyor motor.

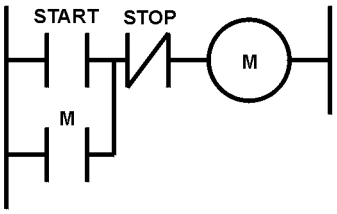


FIGURE 7: EXAMPLE PROGRAM

Action: The operator pushes the start button to start the conveyor.

Step One: The PLC will see that the Start button, an input, has been activated. (The diagram below illustrates the status of the system after this action.)

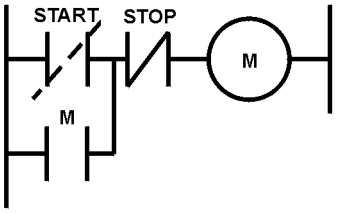


FIGURE 8: STATUS OF THE SYSTEM AFTER STEP ONE

Step Two: The PLC will run the logic and see that if the Start button has been pushed, there is a complete path to the motor starter.

Step Three: Since there is now a complete path or circuit to the motor starter, the PLC turns the motor starter (an output) on.

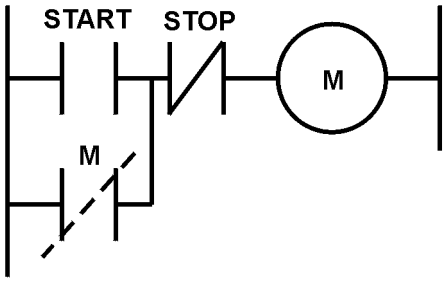


FIGURE 9: STATUS OF THE SYSTEM WHEN START PUSHBUTTON IS RELEASED

(Because the Start pushbutton is traditionally a momentary pushbutton, a latching contactor maintains a closed circuit path.)

When the Stop button is pushed, the PLC will see that the path to it is broken and turn the motor starter off.

PLC COMPONENTS: THE CONTENTS OF "THE BOX"

The PLC mainly consists of a CPU, memory areas, and appropriate circuits to receive input and output data. We can consider the PLC to be a box full of hundreds or thousands of separate relays, counters, timers and data storage locations. These counters, timers, etc. don't "physically" exist but instead are simulated and can be considered software counters, timers, etc. These internal relays are simulated through bit locations in *registers* (more on that later).

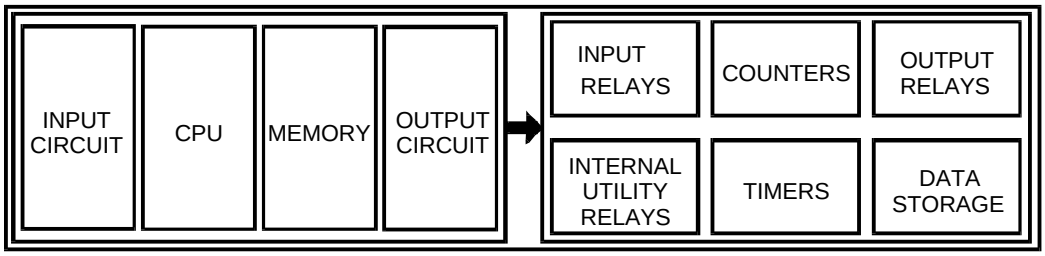


FIGURE 10: THE CONTENTS OF THE BOX

What Each Part Does

The Central Processing Unit **(CPU)** is the most important part of the PLC. It holds the processor that defines what the PLC can and cannot do. **The Processor's** functions are preset so that the PLC has certain fixed limits. These limits are usually the maximum number of inputs and/or outputs (*I/O*) available, but they can also include the maximum number of timers, counters, and registers, as well as type of functions the PLC can perform.

INPUT RELAYS are connected to the outside world. They physically exist and receive signals from switches, sensors, etc. Typically they are *not* relays, but are transistors.

INTERNAL UTILITY RELAYS do not receive signals from the outside world, nor do they physically exist. **They are *simulated* relays and are what enables a PLC to eliminate external relays.**

COUNTERS do not physically exist. **They are simulated counters and they can be programmed to count pulses.** What does the term "pulse" mean in this context? Well, one example of a pulse would be a bottle passing by a sensor. Typically these counters can count up, down, or both up and down. Since they are simulated they are limited in their counting speed. Some manufacturers also include *high—speed* hardware based counters.

TIMERS do not physically exist. They come in many varieties and increments. The most common type is an *on-delay*. Others include *off-delay*, *retentive* and *non-retentive*. Increments vary from 1ms (millisecond) through 1s (second).

OUTPUT RELAYS are connected to the outside world. They physically exist and send on/off signals to *solenoids*, lights, etc. They can be *transistors*, relays, or *triacs*, depending upon the model chosen.

DATA STORAGE. There are typically **registers assigned simply store to data**. They are usually used as temporary storage for math or data manipulation. They are also often used for retentive data storage.

An Outside View

Now that we have discussed the inner workings of the PLC, let's take a look at the outward appearance of the device. There are two basic forms that a PLC comes in: "block" and "rack mounted" I/O.

Block and Block I/O with Expanders

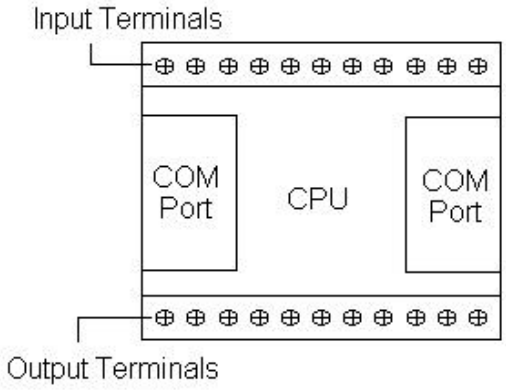


FIGURE 11: BLOCK I/O

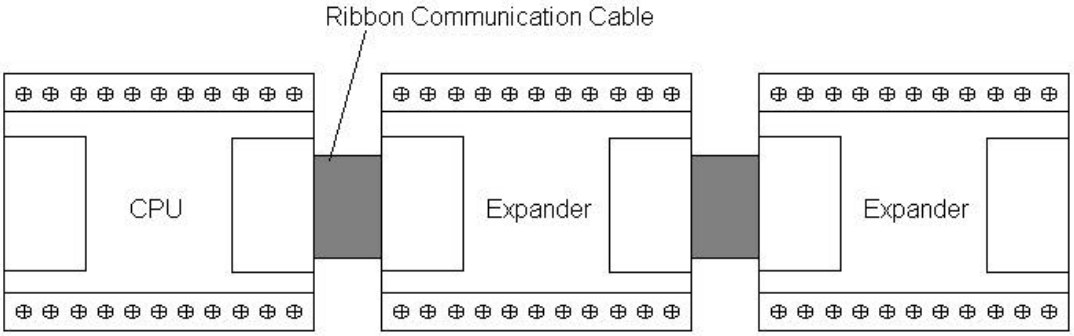


FIGURE 12: BLOCK I/O WITH EXPANDERS

The block I/O is a design more common to PLCs that communicate with small amounts I/O. ("Small amount" refers to a quantity less than 60 I/O.) The input and output terminals are where the user would hardwire the devices to be controlled by the PLC. Each terminal has a unique "address." (We will discuss addressing in greater detail in the next section.) The CPU is located inside of the block. The communication ports allow the PLC to be connected to a computer or hand held programmer. They may also be used to connect special modules or *expanders*. Expander blocks do not contain a CPU. They merely "expand" the number of I/O controlled by the CPU. Based on the manufacturer, each expander could allow a different type of input or output to be used. For example, the base unit could control *digital* I/O and the first expander may control analog outputs only.

Rack Mounted I/O

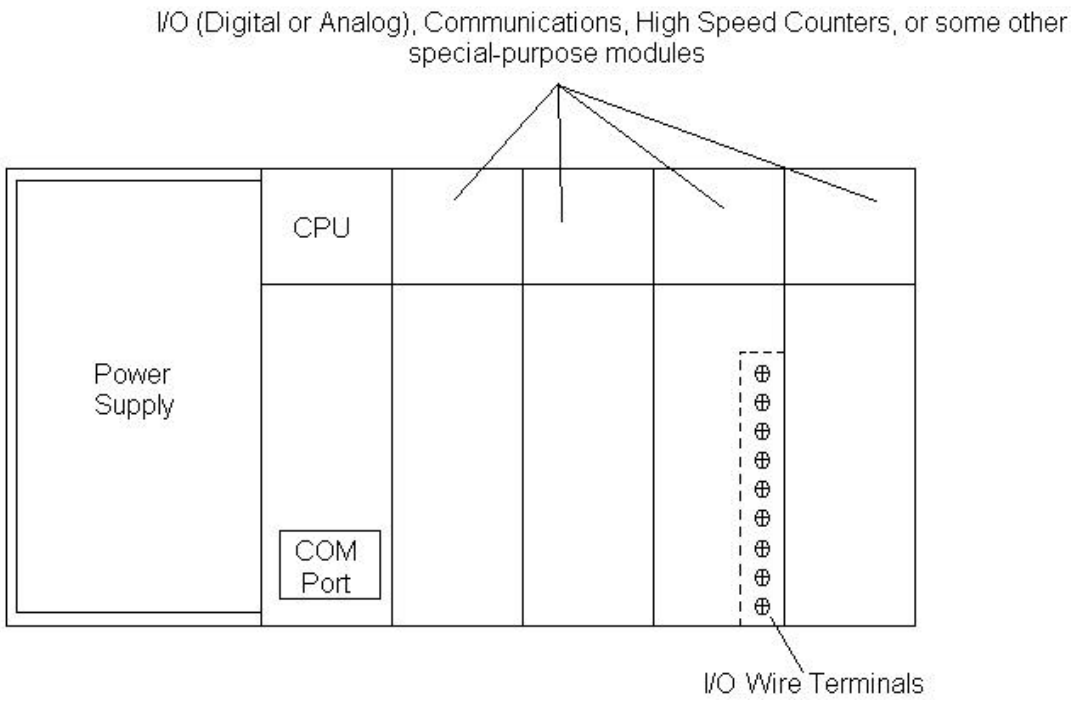


FIGURE 13: RACK MOUNTED I/O

Rack mounted I/O is composed of several printed circuit board I/O cards that are mounted on a “rack” or metal back plate. Generally, the “rack” is designed to hold 4, 6, 8, or more cards. Hundreds of inputs and output devices can be controlled with rack I/O. Like the block I/O, each terminal on each card has a specific PLC address. Unlike the block I/O, based on the end user’s needs, each card can control different types of I/O. For example, a digital, an analog input, and a triac output can all be mounted on the same rack. It is also possible for many rack mounted PLC products to support additional racks of I/O modules located as much as hundreds or thousands of feet from the CPU. In this configuration, there is a “master” CPU attached to “remote” I/O. (A brief discussion of this set up is covered in the last section of the manual, titled “PLC Communications.”)

HOW A PLC THINKS

To make the adjustment to PLCs easier for end users accustomed to wiring relay controlled systems, the programming software for PLCs was modeled after relay wiring schematics. The resulting programming language, **Relay Ladder Logic** (**ladder**), utilizes basic relay wiring symbols to create the logic needed to control a machine or process.

When you consider Relay Ladder Logic, it may be useful to think of a street map**. A street map is like a relay panel**; the city blocks are like relays, and the intersections are similar to the relay's poles. The city streets are the connecting wires. As an example, let's say we know the address of a store where we want to shop. However, because of the number of one-ways, detours and winding streets, you can not go in a straight line. Trace the route on the map from your house to the store. This route is like a hand-wired circuit in a relay panel.

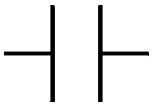
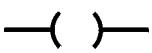
The installed wires in a relay panel are called the circuit path. The intersections represent the poles or contacts on the relays. You will only arrive at your destination without having to stop if all the traffic lights at the intersections are green. **A circuit path will be complete only if all the contacts are in a closed state.** A circuit path will be interrupted if any one of the contacts in the path is open.

The street map design to wire a relay panel is called an installation or wiring diagram. A simple design of the same instruction is called a *ladder diagram*. We must first create a ladder diagram to apply a PLC. **A ladder diagram consists of individual rungs just like on a real ladder. Each rung must contain one or more inputs and one output. The first instruction on a rung must always be an input instruction and the last instruction on a rung should always be an output** (or its equivalent).

We have to create a ladder diagram **because a PLC can't understand a schematic diagram. It only recognizes code.** Fortunately, most PLCs have software that converts ladder diagrams into code and saves us from having to learn the PLC's code.

Basic Instructions

The table below contains the symbols you are likely to see and use most frequently.

SYMBOL	DEFINITION
	Normally Open Contact (Input)
	Normally Closed Contact (Input)
 or 	Coil (Output)

Creating a Ladder Diagram

- **First step—We have to translate all of the items we're using into symbols the PLC understands**. The PLC doesn't understand terms such as switch, relay, and bell. It prefers input, output, coil, contact, etc. **It doesn't care what the input or output device is. It only cares that it's an input or an output.**

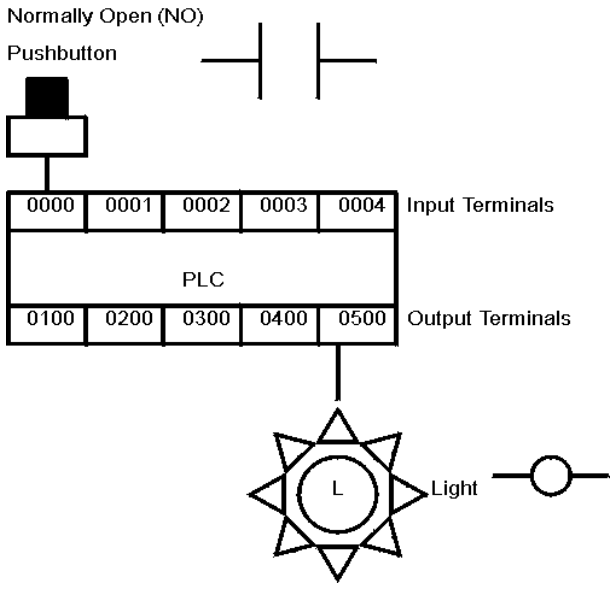


FIGURE 14: CREATING A LADDER DIAGRAM

Creating a Ladder Diagram (continued)

- **Second step—We must tell the PLC where everything is located. In other words, we have to give all the devices an address.** Where is the pushbutton going to be physically connected to the PLC? How about the light? We start with a blank road map in the PLC's town and give each item an address. Could you find your friends if you didn't know their address? You know they live in the same town but which house? The PLC town has a lot of houses (inputs and outputs) but we have to figure out who lives where (what device is connected where). For now, let's say that our input will be called "0000" and our output will be called "500". (Please note that each PLC manufacturer uses different addressing methods.)

- **Final step—We have to convert the schematic into a *logical sequence* of events.** This is much easier than it sounds. The program we're going to write tells the PLC what to do when certain events take place. In our example, we have to tell the PLC to make the light illuminate when the operator presses the button. The picture below is the final converted diagram.

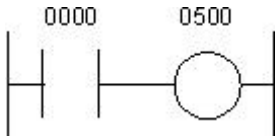


FIGURE 15: COMPLETED LADDER DIAGRAM

INPUTS AND OUTPUTS (I/O)

Inputs	Outputs
Pushbutton	Indicating Light
Selector Switch	Alarm Horn
Analog Signal	Analog Signal
Photoeye Sensor	Motor Starter
Limit Switch	Solenoids
Temperature Sensor	Triacs
Floating Switch	Relays
Operator Interfaces	Transistors

A Sample Program

Now let's compare a simple ladder diagram with its *real world* external physically connected relay circuit and SEE the differences.

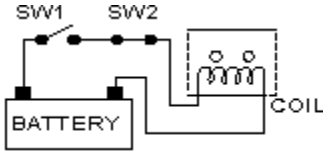


FIGURE 16: A SIMPLE CIRCUIT

In the above circuit, the coil will be energized when there is a closed loop between the + and — terminals of the battery. We can simulate this same circuit with a ladder diagram. Remember, a ladder diagram consists of individual rungs just like on a real ladder. **Each rung must contain one or more inputs and one output. The first instruction on a rung must always be an input instruction and the last instruction on a rung should always be an output** (or its equivalent).

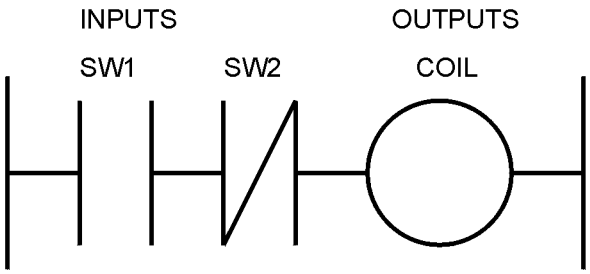


FIGURE 17: CIRCUIT CONVERTED TO LADDER DIAGRAM

Notice in this simple one rung ladder diagram we have recreated the external circuit above with a ladder diagram. Here we used the Normally Closed and Output instructions. Some manufacturers require that every ladder diagram include an END instruction on the last rung. Some PLCs also require an ENDH instruction on the rung below the END rung.

REVIEW 2

Answer the following questions without referring to the material just presented. Begin the next section when you are confident that you understand what you've already read.

1. How does a PLC work?

   _____

2. What is a CPU?

   _____

3. How is one scan time defined?

   _____

4. What is the purpose of each part of a PLC? Which ones physically exist?

   - Counters _____
   - Timers _____
   - Input Relays _____
   - Internal Utility Relays _____
   - Output Relays _____
   - Data Storage _____

4. Why is it necessary to create ladder diagrams when working with PLCs?

   _____

5. What does a ladder diagram consist of?

   _____

6. Outline the three main steps involved in creating a ladder diagram.

   _____

7. Draw the symbol for a Normally Open contact.

8. Draw the symbol for a Normally Closed contact.

COUNTERS

A counter is a simple device intended to do one simple thing: count. Using them, however, can sometimes be a challenge because every manufacturer seems to use them a different way.

What kinds of counters are there? Well, there are **up-counters** (they only count up 1,2,3...). There are **down counters** (they only count down 9,8,7,...). There are also **up-down counters** (they count up and/or down 1,2,3,4,3,2,3,4,5,...)

Typically a high-speed counter is a "hardware" device. The normal counters listed above are typically "software" counters. In other words, they don't physically exist in the PLC but instead are simulated in software. Hardware counters do exist in the PLC and are not dependent on scan time.

To use them we must know 3 things:

1. **Where the pulses that we want to count are coming from.** Typically this is from one of the inputs.(a sensor connected to input 0000 for example)
2. **How many pulses we want to count before we react.** Let's count 5 widgets before we box them, for example.
3. **When/how we will reset the counter so it can count again.** After we count 5 widgets lets reset the counter, for example.

When the program is running on the PLC the program typically displays the current or "accumulated" value for us so we can see the current count value.

Typically, counters can count from 0 to 9999, -32,768 to +32,767 or 0 to 65535. Why the weird numbers? Because most PLCs have 16-bit counters. 0—9999 is 16-bit BCD (binary coded decimal) and —32,768 to 32767 and 0 to 65535 is 16-bit binary.

In this counter we need 2 inputs. One goes before the reset line. When this input turns on the current (Accumulated) count value will return to zero. The second input is the address where the pulses we are counting are coming from.

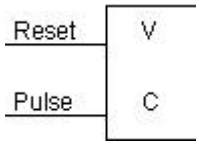


FIGURE 18: COUNTER SYMBOL

For example, if we are counting how many widgets pass in front of the sensor that is physically connected to input 0001 then we would put normally open contacts with the address 0001 in front of the pulse line.

C is the name of the counter. If we want to call it counter 000 then we would put "C000" here. V is the number of pulses we want to count before doing something. If we want to count 5 widgets before turning on a physical output to box them we would put 5 here. If we wanted to count 100 widgets then we would put 100 here, etc. When the counter is finished, it will turn on a separate set of contacts that we also label C.

Note that the counter—Accumulated value ONLY—changes at the off to on transition of the pulse input.

TIMERS

Let's now see how a timer works. What is a timer? It's exactly what the word says: **an instruction that waits a set amount of time before doing something.**

As always, different types of timers are available with different manufacturers. Here are brief descriptions of the most common:

- **On-Delay timer**—This type of timer simply "delays turning on". In other words, after our sensor (input) turns on we wait x-seconds before activating a solenoid valve (output). This is the most common timer.
- **Off-Delay timer**—This type of timer is the opposite of the on-delay timer listed above. This timer simply "delays turning off". We hold the solenoid on for x-seconds before turning it off. It is less common than the on-delay type listed above.
- **Retentive or Accumulating timer**—This type of timer needs 2 inputs. One input starts the timing event (i.e., the clock starts ticking) and the other resets it. The on/off delay timers above would be reset if the input sensor wasn't on/off for the complete timer duration. This timer, however, holds or retains the current elapsed time when the sensor turns off in mid-stream. For example, we want to know how long a sensor is on for during a 1 hour period. If we use one of the above timers they will keep resetting when the sensor turns off/on. This timer, however, will give us a total or accumulated time.

Let's now see how to use them. We typically need to know 2 things:

1. **What will enable the timer**. Typically this is one of the inputs.(a sensor connected to input 0000 for example)
2. **How long we want to delay before we react**. Let's wait 5 seconds before we turn on a solenoid, for example.

When the instructions before the timer symbol are true, the timer starts "ticking". When the time elapses, the timer will automatically close its contacts. When the program is running on the PLC the program typically displays the elapsed or "accumulated" time for us so we can see the current value. **Typically, timers tick from 0 to 9999 in 10 and 100 msec. Increments.**

TIMERS (CONTINUED)

Shown below is a typical timer instruction symbol we will encounter (depending on which manufacturer we choose) and how to use it. Remember that, while they may look different, they are all used basically the same way. If we can setup one, we can setup any of them.

This timer is the on-delay type and is named T. When the enable input is on the timer starts to tick. When it ticks Y (the preset value) times, it will turn on its contacts that we will use later in the program. Remember that the duration of a tick (increment) varies with the vendor and the time-base used (i.e., a tick might be 1ms or 1 second or...).

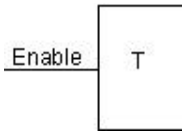


FIGURE 19: TIMER SYMBOL

It's important to note that, **in most PLCs, counters and timers can't have the same name because they typically use the same registers**.

REVIEW 3

Answer the following questions without referring to the material just presented. Begin the next section when you are confident that you understand what you've already read.

1. What three things do you need to know before using a counter?

   _____

   _____

   _____

2. High speed counters are typically _____ devices.

3. Typical counters are _____ counters and therefore do not physically exist.

4. Define the following terms:

   On-delay timer: _____

   Retentive or Accumulating timer: _____

   Off-delay timer: _____

5. What two things do you need to know before using a timer?

   _____

   _____

6. Generally, timers tick from _____ to _____ in _____ and _____ msec increments.

HOW PLCs GATHER DATA

PLC Registers

PLC Registers are storage locations within the device. The numbering system computers and PLCs use to manipulate this information is the binary system. The binary system uses the same basic principles as the decimal system. In decimal we have 10 digits. (0—9) In binary we only have 2 digits (0 and 1). The 0 or 1 indicate whether an instruction is False or True.

Obviously, it is far easier to design a system in which only 2 numbers (0 and 1) are manipulated (i.e. used). In decimal we count: 0,1,2,3,4,5,6,7,8,9, and instead of going back to zero, we start a new digit and then start from 0 in the original digit location. Binary works the same way. We start with 0 then 1. Since there is no 2 in binary we must create a new digit.

Let's refer back to a ladder diagram that we used earlier:

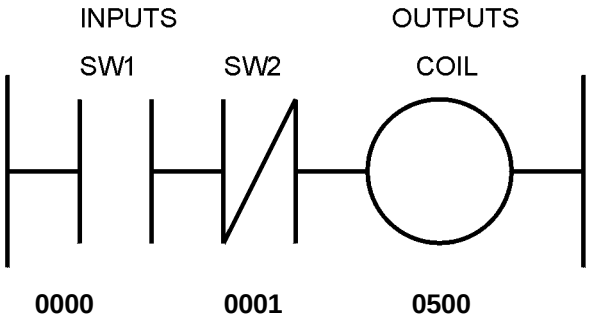


FIGURE 20: LADDER DIAGRAM

We now also give each symbol (or instruction) an **address**. This address sets aside a certain storage area in the PLCs data files so that the status of the instruction (i.e. true/false) can be stored. Many PLCs use 16 slot or bit storage locations. What PLC Registers store is ons and offs. In the example above we are using two different storage locations or registers.

REGISTER 0000															
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
														1	0

REGISTER 0500															
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
															0

PLC Registers (continued)

In the tables above we can see that, in register 00, bit 00 (i.e. input 0000) was a logic 0 and bit 01 (i.e. input 0001) was a logic 1. Register 05 shows that bit 00 (i.e. output 0500) was a logic 0. Remember, the logic 0 or 1 indicates whether an instruction is False or True.

Although most of the items in the register tables above are empty, they should each contain a 0. They were left blank to emphasize the locations we were concerned with.

LOGICAL CONDITION OF SYMBOL			
LOGIC BITS	INPUT 1 SW1	INPUT 2 SW2	OUT COIL
Logic 0	False	True	False
Logic 1	True	False	True

The PLC will only energize an output when *all* conditions on the rung are TRUE. So, looking at the table above, we see that in the previous example SW1 has to be logic 1 and SW2 must be logic 0. Then and ONLY then will the coil be true (i.e. energized). If any of the instructions on the rung before the output (coil) are false then the output (coil) will be false (not energized).

Let's now look at a *truth table* of our previous program to further illustrate this important point. Our truth table will show ALL possible combinations of the status of the two inputs.

Inputs		Outputs
SW1 (INPUT 1)	SW2 (INPUT 2)	COIL (OUT)
False	True	False
False	False	False
True	True	True
True	False	False

PLC Registers (continued)

Register Logic Bits		
SW1 (INPUT 1)	SW2 (INPUT 2)	COIL (OUT)
0	0	0
0	1	0
1	0	1
1	1	0

Notice from the chart that as the inputs change their states over time, so will the output. The output is only true (energized) when all preceding instructions on the rung are true.

How the Program Is Scanned

Now that we've seen how *registers* work, let's process a program like PLCs do to enhance our understanding of how the program gets *scanned.*

Let's consider the following application. We are controlling lubricating oil being dispensed from a tank. This is possible by using two sensors. We put one near the bottom and one near the top, as shown in the picture below.

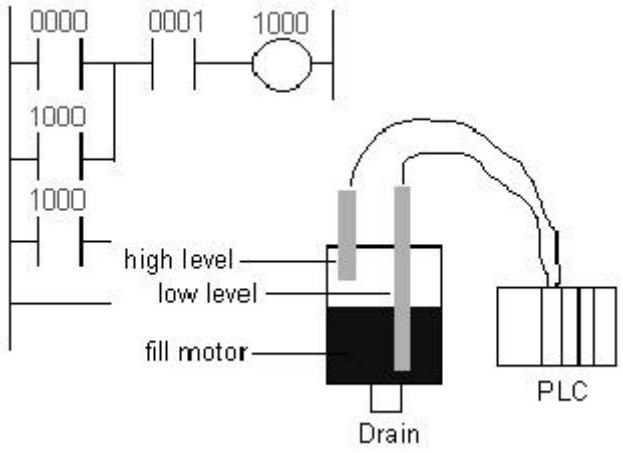


FIGURE 21: DISPENSING OIL FROM A TANK

We want the fill motor to pump lubricating oil into the tank until the high level sensor turns on. At that point we want to turn off the motor until the level falls below the low level sensor. Then we should turn on the fill motor and repeat the process.

How the Program Is Scanned (continued)

Here we have a need for 3 I/O (i.e. Inputs/Outputs). 2 are inputs (the sensors) and 1 is an output (the fill motor). Both of our inputs will be NC (normally closed) fiber-optic level sensors. When they are NOT immersed in liquid they will be ON. When they are immersed in liquid they will be OFF.

We will give each input and output device an address. This lets the PLC know where they are physically connected. Please note that each manufacturer uses a different addressing format. (Check the PLC manufacturer's manuals for details on their addressing methods.) The addresses for this example are shown in the following table:

INPUTS	ADDRESS	OUTPUT	ADDRESS	INTERNAL UTILITY RELAY
Low	0000	Motor	0500	1000
High	0001			

Below is what the ladder diagram will look like.

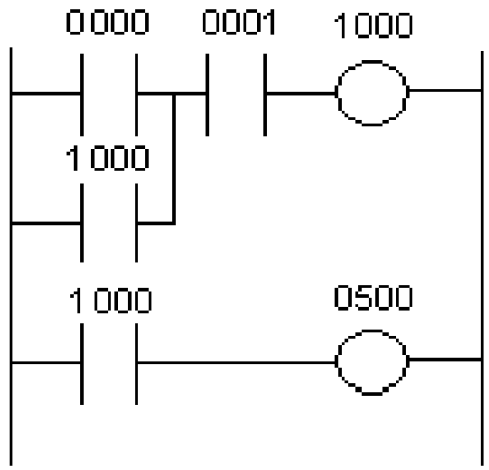


FIGURE 22: THE COMPLETED LEVEL CONTROL LADDER DIAGRAM

Notice that we are using an internal utility relay in this example. You can use the contacts of these relays as many times as required. Here they are used twice to simulate a relay with 2 sets of contacts. Remember, these relays *DO NOT* physically exist in the PLC but are bits in a register that you can use to SIMULATE a relay.

The program is continuously scanned from left to right and top to bottom. The time it takes to scan the program is called ***scan time.*** Scan time depends on the size of the program being scanned; it also varies from manufacturer to manufacturer, and computer to computer

How the Program Is Scanned (continued)

We should always remember that replacing real-world relays is the most common reason for using PLCs in our applications. The internal utility relays make this action possible. It's impossible to indicate how many internal relays are included with each brand of PLC. Some include hundreds, others include thousands, while still others include tens of thousands. Typically, PLC size (not physical size, but I/O size) is the deciding factor. If we are using a micro-PLC with a few I/O we don't need many internal relays. If, however, we are using a large PLC with hundreds or thousands of I/O we'll certainly need many more internal relays.

If ever there is a question as to whether or not the manufacturer supplies enough internal relays, consult their specification sheets. In all but the largest of large applications, the supplied amount should be MORE than enough.

Getting and Moving Data

Let's now start working with some data. This can be considered as among the "advanced" functions of a PLC. This is also the point where we'll see some marked differences between PLCs in functionality and implementation.

Why do we want to get or acquire data? The answer is simple. Let's say that we are using one of the manufacturer's optional modules. Perhaps it's an A/D module. This module acquires analog signals from the outside world (a varying voltage or current) and converts the signal to something the PLC can understand (a digital signal; i.e., 1's and 0's). Manufacturers automatically store this data in memory locations for us. However, we have to get the data out of there and move it some place else. If we don't, the next analog sample will replace the one we just took. In other words, move it or lose it! Other things we might want to do include store a constant (a fancy word for a number), get some binary data off the input terminals (maybe a thumb-wheel switch is connected there, for example), or do some math and store the result in a different location.

There are typically 2 common instruction "sets" for gathering and manipulating data. Some manufacturers use a single instruction to do the entire operation while others use two separate instructions. The two are used together to accomplish the final result. Let's now look briefly at each instruction.

The single instruction is commonly called MOV (move). Some vendors also include a MOVN (move not). It has the same function of MOV but it transfers the data in inverted form. (i.e. if the bit was a 1, a 0 is stored/moved or if the bit was a 0, a 1 is stored/moved). The MOV typically looks like that shown to the right.

FIGURE 23: MOV SYMBOL

The paired instruction typically is called **LDA (Load Accumulator) and STA (Store Accumulator)**. The accumulator is simply a register inside the CPU where the PLC stores data temporarily while its working. The LDA and STA instructions typically look like those shown to the right.

FIGURE 24: LDA SYMBOL

FIGURE 25: STA SYMBOL

The one symbol and two symbol instruction set work the same way—we have no control over which we use; it depends on whose PLC we use.

Getting and Moving Data (continued)

Let's see the single instruction first. The MOV instruction needs to know two things from us: Source, and Destination.

- **Source**—This is **where the data we want to move is located**. We could write a constant here (2222, for example). This would mean our source data is the number 2222. We could also write a location or address for where the data we want to move is located. If we wrote DM100 this would move the data that is located in data memory 100.

- **Destination**—This is the location **to which the data will be moved.** We write an address here. For example, if we write DM201 the data would be moved into data memory 201. We could also write 0500 here. This would mean that the data would be moved to the physical outputs. 0500 would have the least significant bit, 0501 would have the next bit, and so on. This would be useful if, for example, we had a binary display connected to the outputs and we wanted to display the value inside a counter for the machine operator at all times.

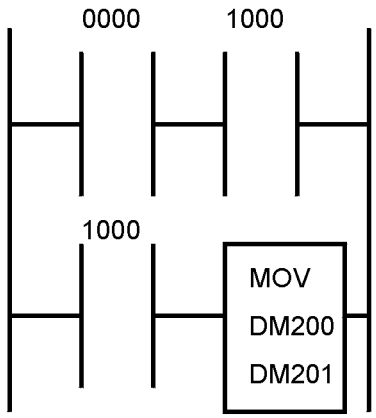


FIGURE 26: MOV LADDER DIAGRAM (SINGLE SYMBOL)

The ladder diagram to do this would look similar to that shown above.

Getting and Moving Data (continued)

The two symbol instruction works in the same way but, as you can see from the diagram below, looks different.

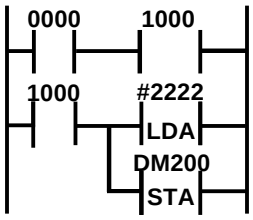


FIGURE 27: LDA/STA LADDER DIAGRAM (TWO SYMBOL)

To use the two symbol instruction we must also supply two things, one for each instruction:

- **LDA**—This instruction is **similar to the source of a MOV instruction**. This is where the data we want to move is located.
- **STA**—This instruction is **similar to the destination of a MOV instruction**. We write an address here.

REVIEW 4

Answer the following questions without referring to the material just presented. Begin the next section when you are confident that you understand what you've already read.

1. What is a PLC register? _____

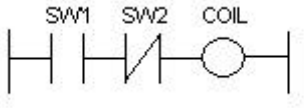
2. Why would you use a binary system instead of a decimal system for data storage?

3. For a Normally Open contact, a 0 value is _____ and a 1 value is _____.

4. In an initialized register, the default setting for each bit is zero, unless otherwise programmed.

   TRUE FALSE

5. Complete the Truth Table and Register for the following circuit:



Truth Table

INPUTS		OUTPUTS
SW1	**SW2**	**COIL**
TRUE		TRUE
TRUE	FALSE	
	TRUE	FALSE
FALSE	FALSE	

Register

INPUTS		OUTPUTS
SW1	**SW2**	**COIL**
0	0	
	1	0
1	1	
1		1

6. Complete the following sentence so it explains the manner in which ladder is scanned. Ladder is scanned is from _____ to _____ and _____ to _____.

MATH INSTRUC-TIONS

In our applications we often must execute some type of mathematical formula on our data. In fact, it's a rare occurrence when our data is actually *exactly* what we needed.

As an example, let's say we are manufacturing widgets. We don't want to display the total number we've made today. Instead, we want to display how many more we need to meet our daily quota of 1000 pieces. We'll say X is our current production. Therefore, we can figure that 1000-X = widgets left to make. To implement this formula we obviously need some math capability.

In general, PLCs almost always include these math functions:

- **Addition**—The capability to add one piece of data to another. It is commonly called ADD.
- **Subtraction**—The capability to subtract one piece of data from another. It is commonly called SUB.
- **Multiplication**—The capability to multiply one piece of data by another. It is commonly called MUL.
- **Division**—The capability to divide one piece of data by another. It is commonly called DIV.

As we saw with the MOV instruction, some manufacturers use a single instruction to do the entire operation while others use two separate instructions. The single instruction method typically requires the following few key pieces of information:

- Source A—This is **the address of the first piece of data we will use in our formula**. In other words, it's the location in memory of where the first "number" is that we use in the formula.
- Source B—This is **the address of the second piece of data we will use in our formula**. In other words, it's the location in memory of where the second "number" is that we use in the formula.

NOTE: Typically we can only work with 2 pieces of data at a time. In other words, we can't work directly with a formula like 1+2+3. Instead, we would have to break it up into pieces; for example, 1+2=X, then X+3= our result.

- Destination—This is **the address where the result of our formula will be put**. For example, if 1+2=3, the 3 would automatically be put into this *destination* memory location.

MATH INSTRUC-TIONS (CONTINUED)

The instructions above typically have a symbol that looks like the one on the right. Of course, the word ADD could be replaced by SUB, MUL, DIV, etc. In this symbol, the source A is DM100, the source B is DM101, and the destination is DM102. Therefore, the formula is simply whatever value is in DM100 + whatever value is in DM101. The result is automatically stored into DM102.

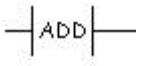


FIGURE 28: ADD SYMBOL

Many PLCs also include **other math capabilities**. Some of these functions could include: Square roots, Scaling, Absolute value, Sine, Cosine, Tangent, Natural logarithm, Base 10 logarithm , X^Y (X to the power of Y), Arcsine (tan, cos), and more. Check with the manufacturer to be sure.

BOOLEAN MATH

Let's now take a look at some simple *Boolean Math*. **Boolean Math lets us do some basic functions with the bits in our registers. These basic functions typically include AND, OR and XOR functions**. Each is described below.

- **AND**—This function enables us to use the truth table below. As you can see, the only time the Result is true (i.e. 1) is when both operators A *AND* B are true (i.e. 1) (1 AND 1 = 1, 0 AND 0= 0) *Result = A AND B*

A	B	Result
0	0	0
1	0	0
0	1	0
1	1	1

BOOLEAN MATH (CONTINUED)

- **OR**—This function is based on the truth table below. As you can see, the only time the Result is true (i.e. 1) is when operator A OR B is true (i.e. 1). Obviously, when they are both true the result is true. *Result = A OR B*

A	B	Result
0	0	0
1	0	1
0	1	1
1	1	1

- **XOR**—This function enables us to use the truth table below. An easy way to remember the results of this function is to think that A and B must be opposites of each other. When they are both the same (i.e. A=B) the result is false (i.e. 0). This function can be useful when you want to compare bits in 2 registers and highlight which bits are different.

  Result = A XOR B

A	B	Result
0	0	0
1	0	1
0	1	1
1	1	0

As we saw with the MOV instruction, some manufacturers use a single instruction to do the entire operation while others use two separate instructions. The single instruction method typically requires the following few key pieces of information:

- **Source A**—This is the address of the first piece of data we will use. In other words, it's the location in memory of where the A is.
- **Source B**—This is the address of the second piece of data we will use. In other words, it's the location in memory of where the B is.
- **Destination**—This is the address where the result will be put. For example, if A AND B = 0 the result (0) would automatically be put into this *destination* memory location.

BOOLEAN MATH (CONTINUED)

The instructions above typically have a symbol that looks like the one on the right. Of course, the word AND could be replaced by OR or XOR. In this symbol, the source A is DM100, the source B is DM101 and the destination is DM102. Therefore, we have simply created the equation DM100 AND DM101 = DM102. The result is automatically stored into DM102.

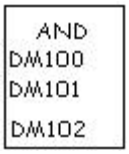


FIGURE 29: AND SYMBOL

The Boolean functions on a ladder diagram are shown below.

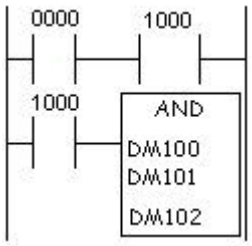


FIGURE 30: AND LADDER DIAGRAM

Note that, once again, we are using a one-shot instruction. As we've seen before, if we didn't use it, we would execute the instruction on every scan. The odds are good that we'd only want to execute the function one time when input 0000 becomes true.

The dual instruction method would use a symbol similar to the one shown on the right. If we use this method, we give this symbol only the Source B location. The Source A location is given by the LDA instruction. As the ladder diagram below shows, the Destination would be included in the STA instruction.

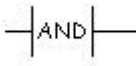


FIGURE 31: AND SYMBOL

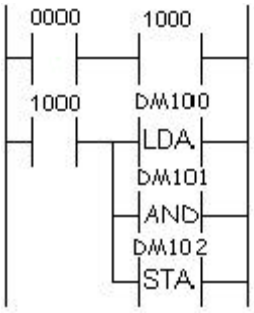


FIGURE 32: AND LADDER DIAGRAM

The results are the same as with the single instruction method. Although the symbol and ladder diagram above show the AND instruction, OR or XOR can be used as well. Simply replace the word “AND” within the instruction with either “OR” or “XOR.”

PLC COMMUNI-CATIONS

The great majority of installed PLCs “service” a moderate amount of I/O (probably less than 128 I/O points). Furthermore, most of the I/O devices are wired onto PLC I/O modules that are installed in a “local” rack or chassis structure. In that arrangement, the I/O modules can communicate directly to the CPU module (which runs the PLC logic) via a wired backplane structure that connects all modules within the chassis.

Communication Between the CPU Module and I/O Devices

But, what if the input and output devices need to be at great distances (thousands of feet) from the CPU module? In such cases, major PLC manufacturers such as Allen-Bradley, General Electric and Groupe Schneider have created proprietary, high-speed networks to connect their PLC’s CPU module to chassis units containing I/O modules, which may be thousands of feet away. These proprietary PLC networks are sometimes referred to as “remote I/O networks”, which provides a reasonable description of their purpose. It is also possible to use new non-proprietary networks such as DeviceNet to allow a PLC to service I/O devices located at a distance. At the present time, the PLC acts as a “master” to the distantly-located “slave” devices in both of these categories of networks.

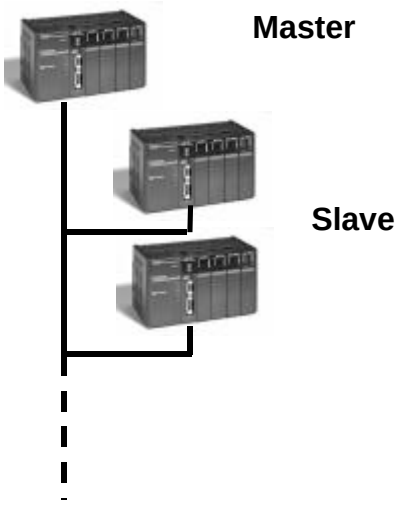


FIGURE 33: MASTER TO SLAVE COMMUNICATIONS

This simply means that the CPU always initiates and controls all communication to remote racks or other devices on the *network*. The communication details of such networks are beyond the scope of this PLC overview.

Communication Between Multiple PLCs and Other Devices

Major PLC manufacturers have also created proprietary networks to permit multiple PLCs of their own brand, plus certain other devices such as PCs and operator stations, to share data. Examples of such networks include Allen-Bradley's Data Highway Plus and Groupe Schneider's Modbus Plus. (For other examples, see the "Glossary" at the end of this module.) Unlike the "remote I/O networks" mentioned earlier, there are not racks of I/O devices directly on these networks. Instead, these networks exist to connect the CPU's of multiple PLCs to each other and to PCs and other devices. These networks permit sharing and exchanging data collected by each individual PLC.

Since the CPU on each PLC may need to exchange data with any one of a dozen (or more) other PLCs on the network, each network must have a method of managing the communication traffic. Data must be sent between multiple PLCs or other devices without data "collisions" or confusion. Each network type has a unique protocol that establishes the "rules" of how communication will take place. If all devices on the network have the ability to initiate the transmission of data, the network is referred to has having "peer to peer" communication, rather than the "master/slave" arrangement that characterizes remote I/O networks.

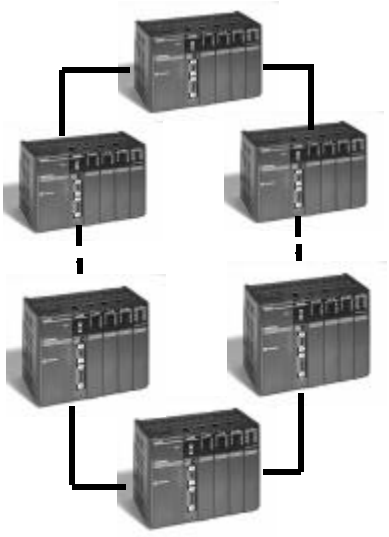


FIGURE 34: PEER TO PEER COMMUNICATIONS

Many types of devices (PLCs, PCs, programming devices, *operator interfaces*) can operate simultaneously on these types of networks. These networks are often used to report data from PLCs "up" to computers that are collecting plant-wide information. Many other types of "bridging" can exist between these networks and other networks or devices, but the discussion of those details is beyond the scope of this overview.

A Note about Electronic Operator Interface Products

PLCs can communicate with operator personnel via an electronic operator interface device (O/I). O/I products function just as their name implies - they allow the "operator" of a machine to "interface" with the PLC. This interface may include seeing the status of a counter, changing the set point on a timer, converting numerical data from Fahrenheit to Celsius, or any number of other operations.

FIGURE 35: OPERATOR INTERFACE PRODUCT

Electronic O/Is can also replace standard control devices like pushbuttons, lamps and selector switches, thus decreasing the number of input and output devices that have to be wired to the PLC. Operator interface products are available to connect to the PLC via a wide variety of communication options, including connection to:

- a port on the PLC's CPU module
- a general-purpose proprietary network like Data Highway Plus
- a PLC remote I/O network
- a non-proprietary network like DeviceNet

The only wiring required for PLC-to-O/I communication is a single cable that links a port on the O/I to a port or node connection on the PLC or the network.

SUMMARY

This module has provided you with a brief introduction to PLC history, application, and operation. It is important that you grasp the theories that have been presented to you. Once you have mastered the basics, it will be possible for you to use anybody's PLC. The manufacturer's documentation will provide the details to assist you with a specific PLC application.

REVIEW 5

Answer the following questions without referring to the material just presented.

1. List the four math functions common to most PLCs.

2. Complete the following tables:

Result = A AND B

A	B	RESULT
0	0	
1	0	
0	1	
1	1	

Result = A OR B

A	B	RESULT
	0	0
1		1
0		1
1	1	

Result = A XOR B

A	B	RESULT
0	0	
	0	1
0		1
1	1	

GLOSSARY

Analog	Any type of input or output that has more than two states; on and off (see Digital). An analog signal can vary in magnitude from “off” to a high-end value or between two non-zero values. An example of an analog device would be a level sensor that returns a voltage somewhere between 0 and 10 V that can vary over time.
Bit	A single digit that only has two possible values – 0 or 1. Multiple bits can be combined to form bytes or words.
Boolean Math	A general term used to describe several different types of comparative logic functions. Specific Boolean Math functions include, but are not limited to, AND, OR, XOR, etc.
Central Processing Unit (CPU)	The main processor of information in your computer. This single chip performs all of the logic and math operations of the PLC.
Digital	Any type of input or output signal that has exactly two states, on and off. An example of a digital device would be a pushbutton, which can either be pressed (ON) or released (OFF).
Expander	A module connected to block I/O via a cable connection that increases the number of I/O controlled by a CPU. Expanders do not contain a CPU and therefor are often called "dumb I/O blocks."
I/O	Inputs and Outputs
Ladder Diagram	The result of ladder programming used to control a PLC. The ladder language is modeled after relay wiring schematics. The fundamental theories behind ladder are consistent among all manufacturers. However, each PLC manufacturer generally has a proprietary ladder software package.
Logic	A series of directives or boundaries created to allow a process to be controlled. Logic can be programmed via hardwiring (as is the case with relay logic) or via a PC (as is the case with a PLC).
Network	Several devices connected together, through electrical means, for data acquisition and/or control.
Non-retentive	All values are resent to zero after powering down the unit.
Off-Delay Timer	Will turn an output OFF after X amount of seconds has passed.

On-Delay Timer	Will turn an output ON after X amount of seconds has passed.
Operator Interface (O/I)	A device that allows the operator of a machine to monitor and control devices attached to a PLC.
Register	A storage area, within the PLC, for information. Registers can have a one or two (or more) word capacity.
Relays	A type of switch that can control AC or DC loads.
Relay Circuits	Devices often used in control. Can be opened and closed electronically to complete logic circuits.
Retentive	Will store data in memory so that it remains intact after powering down the unit.
Sensor	A sensing element. The basic element that usually changes some physical parameter to an electrical signal.
Solenoid	A type of output device and a specific type of coil. Both coils and solenoids utilize voltage to convert electrical energy to mechanical energy via magnetic fields. A solenoid is an actual physical device, where as a coil is a generic description for any type of electrical output.
Starter	A control device usually consisting of a contact and overload. With DeviceNet, it will also contain a communication module used for starting and stopping loads.
Transistors	A solid-state, electronic switch. It is fast, switches a small current, has a long lifetime, and works with DC only.
Triacs	Or silicon controlled rectifiers (SRCs) act as a mediator between the PLC and the AC output device. The triac or SCR functions as a switch that that responds to the commands of the PLC logic.

REVIEW 1 ANSWERS

1. 4A, 1B, 3C, 5D, 2E

2. Any three of the following:
   PLCs take up less space
   PLCs have fewer hardwired devices and their inner workings are solid state
   With PLCs it's simpler to write and modify programs
   PLCs have a longer life
   PLCs require less maintenance

REVIEW 2 ANSWERS

1. A PLC works by continually scanning a program.

2. A CPU, or Central Processing Unit, holds the processor that defines what the PLC can and cannot do.

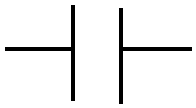
3. One scan time is the time it takes to check the input status, execute the program, and update the output status.

4. Counters count pulses. They do not physically exist.
   Timers are instructions that wait a specified time before doing something. They do not physically exist.
   Input relays receive signals from switches, sensors, etc. They physically exist.
   Internal Utility Relays are simulated relays that enable a PLC to external relays. They do not physically exist.
   Output Relays send On/OFF signals to solenoids, lights, etc. They physically exist.
   PLCs contain registers assigned to store data. They do not physically exist.

5. Because PLCs can't understand schematic diagrams; they only understand code.

6. A ladder diagram consists of individual rungs, each of which must contain one or more inputs and one or more outputs.

7. a. Translate all the items being used into symbols the PLC understands.
   b. Tell the PLC where everything is located by giving the devices addresses.
   c. Convert the schematic into a logical sequence of events.

8. 

9. 

REVIEW 3 ANSWERS

1. a. Where the pulses we want to count are coming from
   b. How many pulses we want to count before we react
   c. When and how we will reset the counter so it can count again
2. hardwired
3. software
4. a. A timer that delays turning a device on
   b. A timer that holds the current elapsed time when a device turns off midstream
   c. A timer that delays turning a device off
5. a. What will enable the timer
   b. How long we want to delay before we react
6. 0, 9999, 10, 100

REVIEW 4 ANSWERS

1. A PLC register is a storage location within the device.
2. Because it's easier to design systems in which only two numbers have to be manipulated.
3. False, True
4. True
5. Left to right and top to bottom: False, True, False, False, 0, 0, 0, 0
6. Left, right, top, bottom

REVIEW 5 ANSWERS

1. Addition, Subtraction, Multiplication, Division
2. Left to right and top to bottom:
   a. 0, 0, 0, 1
   b. 0, 0, 1, 1
   c. 1, 1, 0, 0

Cutler-Hammer
Milwaukee, Wisconsin U.S.A.

Publication No. TR.09E.02.T.E
February 1999
Printed in U.S.A. (GSP)