CHAPTER 12

Relay Logic, Programmable Logic Controllers, and Motion Controllers

OBJECTIVES

After studying this chapter, you should be able to:

- Explain the operation of electromechanical relays, time-delay relays, counters, and sequencers.
- Explain the purpose and operation of a ladder diagram.
- Explain the operation of a relay-based controller.
- Understand the concept and purpose of a programmable logic controller (PLC).
- Understand the hardware and wiring required in a PLC-based system.
- List the steps that must be taken to make a PLC control system operational.
- Understand the basic instructions used in a PLC program.
- Differentiate the ways that a PLC can be programmed.

- Understand how PLCs are used with networks.
- Explain the purpose and operation of a motion controller.

INTRODUCTION

The automatic control of repetitious mechanical or physical processes is a common control problem, which abounds in major appliances, office machines, the manufacturing industry, and industry in general. Traditionally, this kind of process control was done with electromechanical devices such as relays, timers, and sequencers. With this approach, however, the circuit must be rewired if the control logic changes, which was a particular problem to the automotive industry because of the annual model changeovers. In response to this problem, in the late 1960s, General Motors developed the specifications for a programmable electronic controller that could replace the hard-wired relay circuits. Based on those specifications, Gould Modicon Company developed the first programmable logic controller (PLC). The PLC is a small, microprocessor-based process-control computer

that can be connected directly to such devices as switches, small motors, relays, and solenoids, and it is built to withstand the industrial environment. This chapter will introduce the principles of relay logic control and then describe the general operation and programming of the PLC. For those who want a detailed working knowledge of a PLC, a tutorial (published by Allen-Bradley) for connecting, programming, and operating the Allen-Bradley MicroLogix 1000 and SLC 500 PLCs, is included in the appendix. The chapter concludes with discussions on PLC networks and motion controllers.

12.1 RELAY LOGIC CONTROL

Relay Logic

Relays as a device are covered in Chapter 4; however, a quick review of the basics is appropriate here. An **electromechanical relay** (EMR) is a device that uses electromagnetic force to close (or open) switch contacts—in other words, an electrically powered switch. Figure 12.1 shows a diagram of a relay. Relay contacts come in two basic configurations—**normally open contacts** (NO), which are open in the unenergized state (and close when the relay coil is energized), and **normally closed contacts** (NC), which are closed in the unenergized state (and open when the relay is energized). By convention, *the relay symbol always shows the contacts in the unenergized state*. Relays are available with a variety of multiple-contact configurations, two of which are shown in Figure 12.2.

No doubt, relays were first developed to satisfy two electrical control needs: remote control (the ability to turn devices on and off from a remote location) and power amplification. An example of power amplification is the starting relay in a car (a low-current ignition switch energizes a relay, which in turn passes a high current to the starting motor). Electrical designers soon recognized that relays could be used to implement control logic. AND, OR, and NOT gates, as well as flip-flops, can be wired from relays (Figure 12.3). For the AND gate in Figure 12.3(a), relays A *AND* B must be energized for a voltage to be at X. Figure 12.3(b) shows an OR gate. Here, if either relay A *OR* B is energized, the output X receives a voltage. Figure 12.3(c) shows a relay NOT gate. If relay A is ener- gized, the output is 0 V; if it is *NOT* energized, the output receives a voltage.

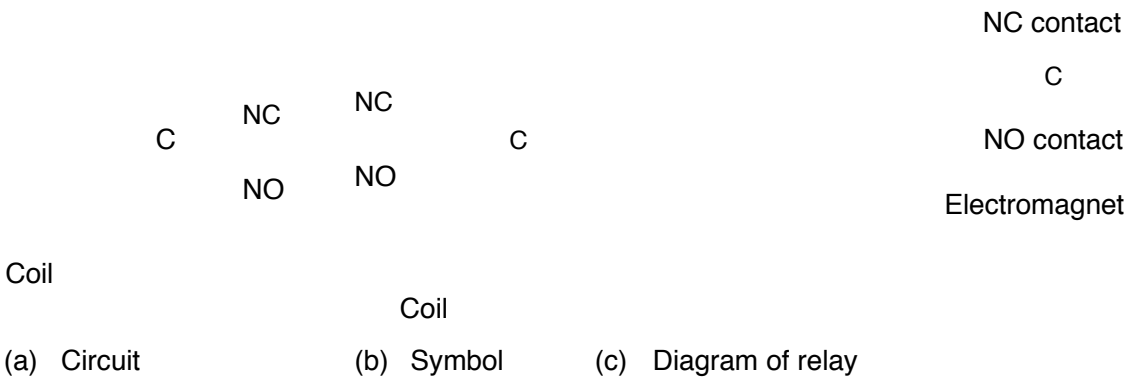


Figure 12.1
A relay: showing normally closed (NC) and normally open (NO) contacts.

Figure 12.2
Multiple-contact relays.

Circuit Symbol

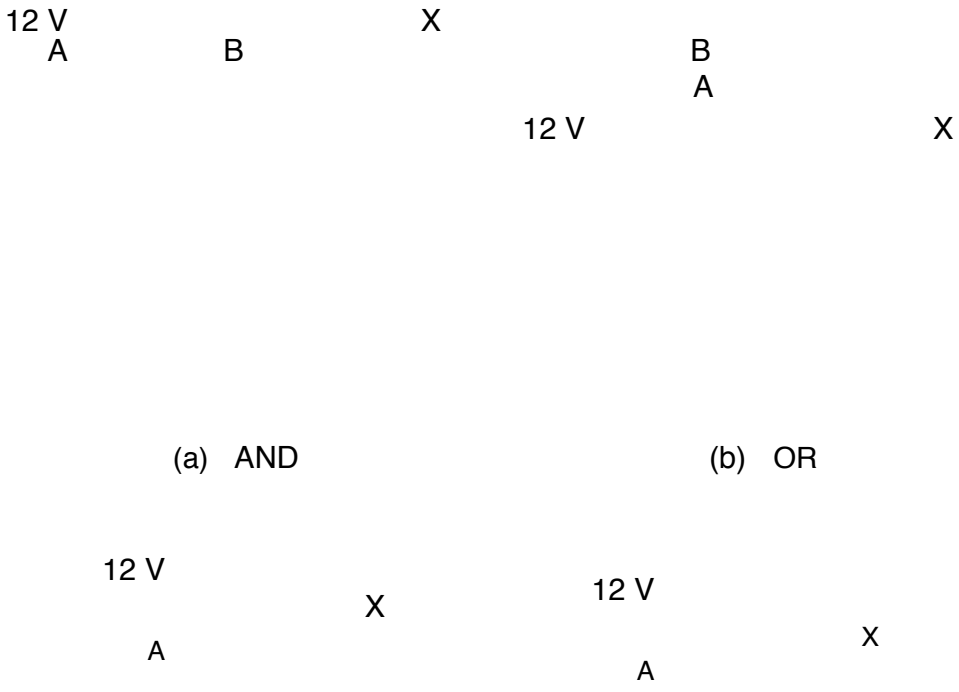
(a) Double-pole/double-throw (DPDT)

Circuit Symbol

(b) Triple-pole/double-throw (3PDT)

(c) General-purpose relay (Courtesy of Rockwell Automation)

Figure 12.3
Logic functions from relays.



(c) NOT

(d) Scaling or latching relay (flip-flop)

A flip-flop can be made from a relay using a principle called **sealing.** As seen in Figure 12.3(d), once the relay is energized, an electrical path through the contacts takes over the job of providing power to the coil, and the original A voltage can be removed. (This process is frequently called **latching.** However, a relay catalog uses the term *latching relay* to mean a relay that has a built-in mechanical-latching mechanism.) To unseal the relay, the power coming through the contacts must be (temporarily) removed.

Ladder Diagrams

Switches and relays became widely used in industry for controlling motors, machines, and processes. A switch can turn a single machine on and off, but a relay logic network can control an entire process—turning on one machine, waiting until that operation is done, then turning on the next operation. Decisions can be made by the logic—for example, if a part is too tall, it goes in one bin; otherwise, it goes in the other bin.

Eventually, the **ladder diagram,** a special type of wiring diagram, was developed for relay-and-switching control circuits. Figure 12.4 shows a ladder diagram (notice it resembles a ladder). The ladder diagram consists of two **power rails,** which are placed vertically on each side of the diagram, and *rungs*, which are placed horizontally between the power rails. The power rails are the source of power in the circuit (AC or DC), where the left rail is the "hot" side (voltage) and the right rail is neutral (AC) or ground (DC). Therefore, each rung is connected across the voltage source and is an independent circuit. A rung typically contains at least one set of switch or relay contacts and usually only one load such as a relay coil or motor. When the contacts in a particular rung

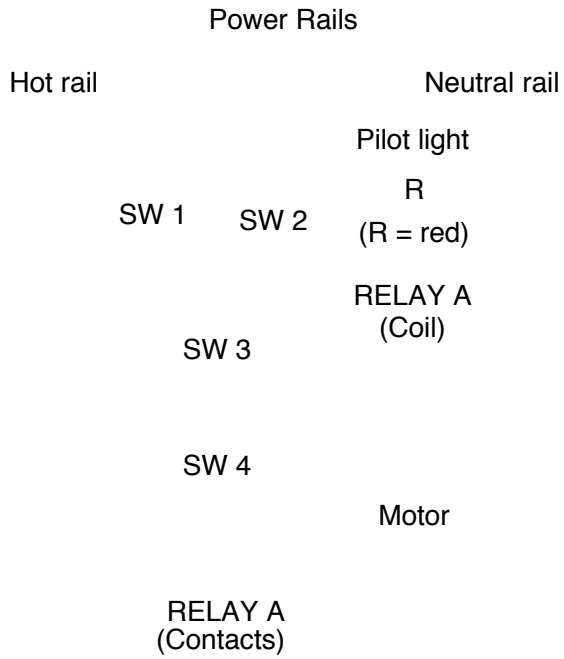


Figure 12.4
A relay logic ladder diagram.

close to make a continuous path, then that rung becomes active, and its load is energized. For example, the top rung in Figure 12.4 contains two switches and a pilot light in series. For the rung to become active, both switches must close, which will then apply voltage to the light. The middle rung has two switches in parallel, so only one of these switches must be closed to make the rung active. The load in this rung is a relay coil (RELAY A). The bottom rung contains a set of NO contacts from RELAY A, and the load is a motor. Consequently, when either SW 3 or SW 4 of the middle rung closes, RELAY A coil is energized, and the motor (in the bottom rung) starts.

EXAMPLE 12.1

In a certain bank, each of three bank officers has a unique key to the vault. The bank rules require that two out of the three officers be present when the vault is opened. Draw the ladder diagram for a relay logic circuit that will unlatch the door and turn on the light when two of the three keys are inserted.

SOLUTION

Three combinations of keys will open the vault: A and B, A and C, and B and C. Each of the three keys—A, B, and C—fits in its own key switch that has two sets of NO contacts. Figure 12.5 shows the completed ladder diagram. The top rung has three branches of switch contacts, one for each acceptable possibility. At least one branch must have continuity so that the relay coil is energized. The bottom rung, activated when the relay contacts close, provides power to the door-latch solenoid and the vault light.

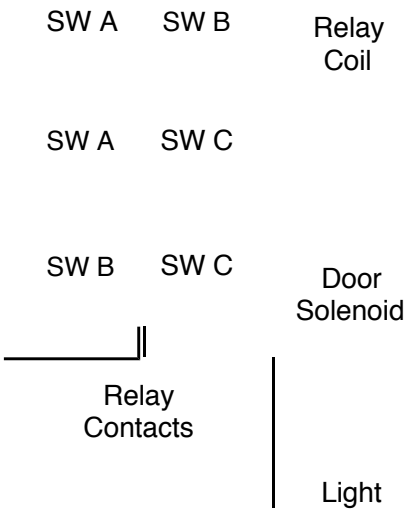


Figure 12.5
A ladder diagram (Example 12.1).

EXAMPLE 12.2

A simple pick-and-place robot picks up parts from one conveyer belt and places them on another belt, as shown in Figure 12.6(a). When a part moving along the lower conveyer belt activates Switch 1, a solenoid-powered gripper clamps on the part and carries it toward the upper conveyer belt. When the gripper reaches Switch 2, it releases the part and moves back (empty) to receive the next part. When the gripper reaches Switch 3, it halts and waits for the next part to start the cycle all over again. Draw the relay logic ladder diagram to control this operation.

SOLUTION

Figure 12.6(b) shows the completed ladder diagram. The cycle starts when a part on the lower conveyer belt activates Switch 1—momentarily closing contacts *SW 1-1* (shown in the top rung). This action energizes the *MOTOR* relay, and the motor starts receiving current through the *M-1* contacts (bottom rung). The *MOTOR* relay is sealed (latched) via relay contacts M-2 (now even when *SW 1-1* opens, the *MOTOR* relay will stay energized through the *M-2* contacts).

While the motor is getting started, the second set of Switch 1 contacts (*SW 1-2*) in rung 2 energizes the *DIRECTION* relay, causing the motor to drive the gripper toward the upper conveyer belt (the *DIRECTION* relay determines the motor direction with DPDT contacts D-3). Using the same technique as in rung 1, the *DIRECTION* relay is sealed "on" through one of its own contacts (*D-1*).

When the gripper arrives at the upper conveyer belt, it activates Switch 2, which opens the normally closed *SW 2-1* contacts. This action breaks the *DIRECTION* relay seal, allowing it to become de-energized, and the motor switches direction—now going back toward the bottom.

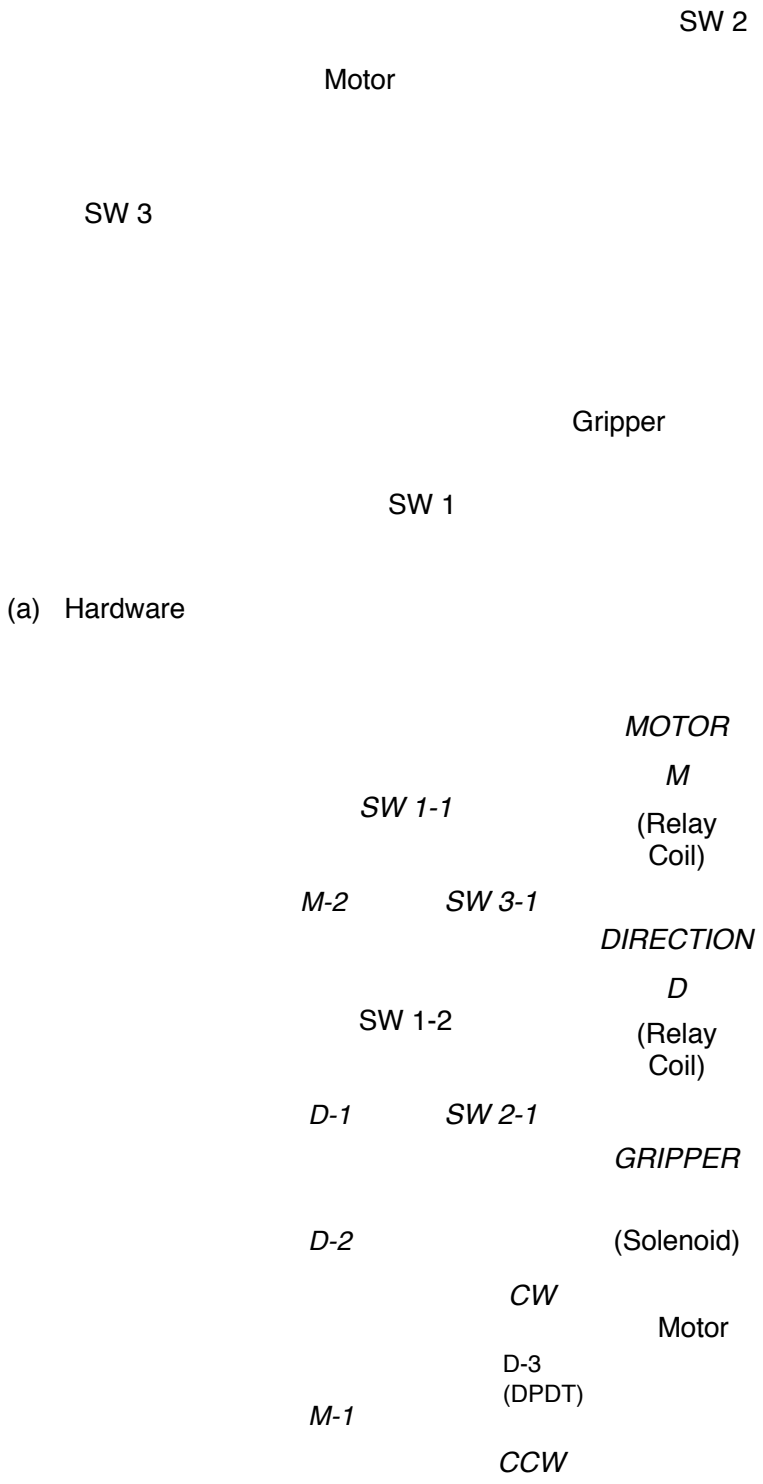
Rung 3 controls the *GRIPPER* solenoid. Controlling the gripper is simple because the *GRIPPER* needs to be activated for the same time period as the *DIRECTION* relay is energized—that is, from the time when the motor starts its upward journey, until just before it descends.

The final action in the cycle is when the gripper descends (empty) and activates Switch 3. This action opens the normally closed contacts (*SW 3-1*), breaking the seal in rung 1 (which had held the *MOTOR* relay energized), and the motor stops.

Timers, Counters, and Sequencers

Many control situations require that a time delay be inserted at some point in the process. For example, a mixing operation might take 90 s, or a conveyer belt might be allowed 10 s to reach speed before parts are placed on it. A relay control system would use a **time-delay relay** to create the time delay. Once activated, the time-delay relay will wait a predetermined period of time before the contacts open (or close)—the delay

Figure 12.6
Pick-and-place robot control (Example 12.2).



(b) Ladder diagram

may range from less than a second to minutes. Time-delay relays are categorized as being either *on-delay* or *off-delay* and are best explained by example. A 5-s **on-delay relay** with NO contacts will wait 5 s (after being energized) before closing its contacts. When the relay is de-energized, the contacts will open immediately. An application of the on-delay function is a security system that delays activation for a period of time after being turned on, to allow the occupants time to leave. The **off-delay relay** provides a time delay when the coil is de-energized. For example, a 5-s off-delay relay with NO contacts would close its contacts immediately when energized; when it is deenergized, however, the contacts would remain closed for 5 more seconds before opening. An application of the off-delay function is car lights that remain on for a period of time after being turned off (to provide light for the occupants as they are leaving the vehicle). Figure 12.7(a) shows the symbols used for time-delay relays.

Several different designs of delay timers, which use different operating principles, have evolved over the years. The **pneumatic time-delay relay** [Figure 12.7(b)] works as follows: When the relay is activated, a small spring-loaded bellows is squeezed closed, causing the air to escape through a check valve. The bellows then is allowed to slowly expand (the air being admitted through a small hole). When the bellows reaches its normal size again, the contacts close. These relays are available with a fixed or an adjustable delay.

A **thermal time-delay relay** [Figure 12.7(c)] uses a temperature-sensitive bimetallic strip. When the relay is energized, a small resistance heater warms the bimetallic

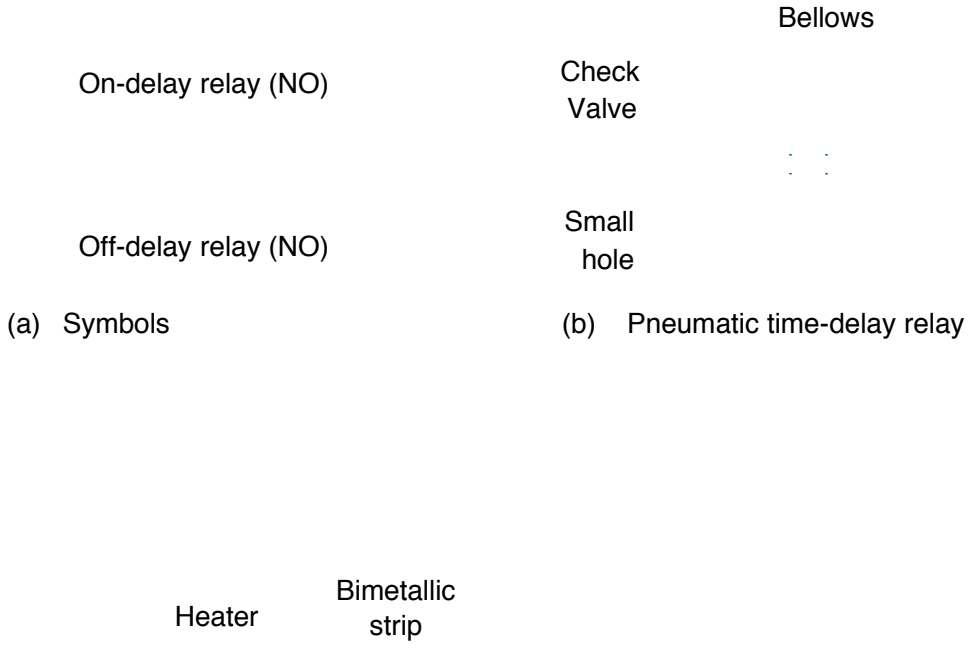


Figure 12.7
Time-delay relays.

(c) Thermal time-delay relay

(d) Solid-state time-delay relay

strip. As the strip heats, it bends and eventually closes the contacts. The flasher unit in a car, which controls the flashing directional signals, works on this principle.

Solid-state time-delay relays are used in most newer systems that require delay relays. An example is shown in Figure 12.7(d); the knob is for setting the delay. These units are based on the delay involved when (1) charging a capacitor or (2) counting high-speed clock pulses with a digital counter: the higher the count, the longer the delay.

EXAMPLE 12.3

A small electric furnace has two heating elements that are energized in stages 3 min apart. That is, when the furnace is turned on, the first heating element comes on right away, and the second element comes on 3 min later. A temperature sensor will shut down the furnace if it gets too hot. Draw the ladder diagram for the control circuit.

SOLUTION

On the top rung of the ladder diagram shown in Figure 12.8, a push-button switch energizes the *HEATER* relay, and the relay is sealed by the *H-1* contacts. This action supplies power to the first heating element (*ELEMENT-1*), via the relay contacts *H-2*, and energizes the *TMDY* delay relay (on-delay type). After 3 min, the time-delay relay activates the second heating element via the *TMDY-1* contacts (bottom rung). At any time if the stop button is pushed or the *OVER-TEMP* sensor contacts open, the seal is broken, and the *HEATER* relay deenergizes, causing both heating elements to shut down.

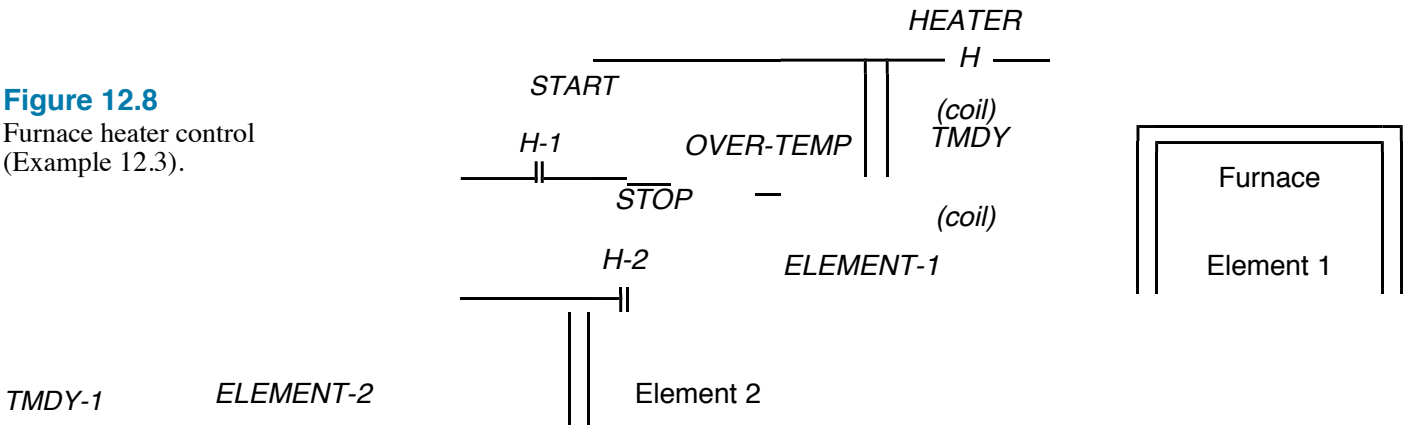
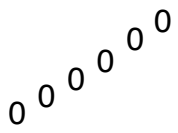


Figure 12.8
Furnace heater control (Example 12.3).

(a) Ladder diagram

(b) Furnace

Figure 12.9
An electromechanical counter.



Electromechanical counters are devices that count events. The counter, which has a readable output similar to a car's odometer (Figure 12.9), increments once for each electric pulse received. Counters usually simply keep track of the number of times an operation is performed. For example, it could count the number of products coming down the assembly line. Some models will close a set of switch contacts when the count gets to a preset value.

Electromechanical sequencers control the process that has a sequence of timed operations. An example of this is a dishwasher controller. The water is admitted for so many seconds, then the wash cycle is activated for so many seconds, and then the water is pumped out for so many seconds, and so on. As shown in Figure 12.10, the sequencer (known as a *drum controller*) consists of a small timing motor that slowly rotates a cluster of cams, and the cams activate switches. Each switch controls one of the timed operations, and the whole sequence repeats itself with each revolution of the drum. Some sequencers use a solenoid ratchet mechanism to rotate the cams in small discrete steps— one step for each input pulse.

Cams

Figure 12.10
An electromechanical sequencer.

Motor

Switches

12.2 PROGRAMMABLE LOGIC CONTROLLERS

Introduction

A **programmable logic controller** (PLC) is a small, self-contained, rugged computer designed to control processes and events in an industrial environment—that is, to take over the job previously done with relay logic controllers. Figure 12.11 shows a number of different types of PLCs. Physically, they range in size from that of a camera to that of a shoe box (and sometimes larger). Wires from switches, sensors, and other input devices are attached directly to the PLC; wires driving lights, small motors, motor starters, and other output devices are also connected directly to the PLC (Figure 12.12). Each PLC contains a microprocessor that has been programmed to drive the output terminals in a specified manner, based on the signals from the input terminals. The PLC program is usually developed on a separate computer such as a personal computer (PC), using special software provided by the PLC manufacturer. Once the program has been written, it is transferred, or downloaded, into the PLC. From this point on, the PLC may operate on its own, as a completely independent controller.

PLC Hardware

Figure 12.13 shows the block diagram for a PLC and includes the fundamental parts found in any microprocessor system (as discussed in Chapter 2). These major functional blocks are explained next.

Figure 12.11
A selection of PLCs. (Allen-Bradley products courtesy of Rockwell Automation)

F
i
A

Power Supply

PLCs are usually powered directly from 120 or 240 Vac. The power supply converts the AC into DC voltages for the internal microprocessor components. It may also provide the user with a source of reduced voltage to drive switches, small relays, indicator lamps, and the like.

Process or

The **processor** is a microprocessor-based CPU and is the part of the PLC that is capable of reading and executing the program instructions, one-by-one (such as the rungs of a ladder logic program).

PowerSupply

Processor

All

F
i
B
l

Program Memory

The **program memory** receives and holds the downloaded program instructions from the programming device. If this memory is standard RAM, the program will be lost every time the power is turned off, requiring it to be reloaded. To avoid this bother, the program memory may use an EEPROM (electrically erasable programmable ROM) or a battery-backup RAM, both of which are capable of retaining data even when the power is off. An EPROM (UV erasable PROM) can also store the program, but this device requires a special programming unit that is not part of the PLC system.

Data Memory

Data memory is RAM memory used as a "scratch pad" by the processor to temporarily store internal and external program-generated data. For example, it would store the present status of all switches connected to the input terminals and the value of internal counters and timers.

Programming Port

The **programming port,** an input/output (I/O) port, receives the downloaded program from the programming device (usually a PC). Remember that the PLC does not have an elaborate front panel or a built-in monitor; thus, to "see" what the PLC is doing inside (for debugging or troubleshooting), you must connect it to a PC, as illustrated in Figure 12.12.

Input and Output Modules

The **I/O modules** are interfaces to the outside world. These control ports may be built into the PLC unit or, more typically, are packaged as separate **plug-in modules,** where each module contains a set of ports (see Figures 12.11 and 12.12). The most common type of I/O is called **discrete I/O** and deals with on-off devices. *Discrete input modules* connect real-world switches to the PLC and are available for either AC or DC voltages (typically, 240 Vac, 120 Vac, 24 Vdc, and 5 Vdc). As shown in Figure 12.14, circuitry within the module converts the switched voltage into a logic voltage for the processor. Notice that an AC switch voltage must first be converted into a DC voltage with a rectifier. Also, an input module will usually include an opto-isolator to prevent possible high-voltage spikes from entering the PLC. A typical discrete input module would have 4, 8, or 16 inputs.

Discrete output modules provide on-off signals to drive lamps, relays, small motors, motor starters, and other devices. As shown in Figure 12.15, several types of output ports are available: Triac outputs control AC devices, transistor switches control DC devices, and relays control AC or DC devices (and provide isolation as well). A typical discrete output module would have 4, 8, or 16 outputs.

F
i
P
L

uit

Switch

120 Vac

Optoisolator

5 V

5-V Logic
(to processor)

(a) AC input module circuit

Switch

Optoisolator

5–24 Vdc V

5-V Logic
(to processor)

(b) DC input module circ

Analog I/O modules allow the PLC to handle analog signals. An *analog input module* [Figure 12.16(a)] has one or more ADCs (analog-to-digital converters), allowing analog sensors, such as temperature, to be connected directly to the PLC. Depending on the module, the analog voltage or current is converted into an 8-, 12-, or 16-bit digital word. An *analog output module* [Figure 12.16(b)] contains one or more DACs (digital- to-analog converters), allowing the PLC to provide an analog output—for example, to drive a DC motor at various voltage levels.

Specialized modules that perform particular functions are available for many PLCs. Examples incluye :

- *Thermocouple module*—Interfaces a thermocouple to the PLC.
- *Motion-control module*—Runs independently to control muti-axis motion in a device such as a robot (covered later in this chapter).
- *Communication module*—Connects the PLC to a network.

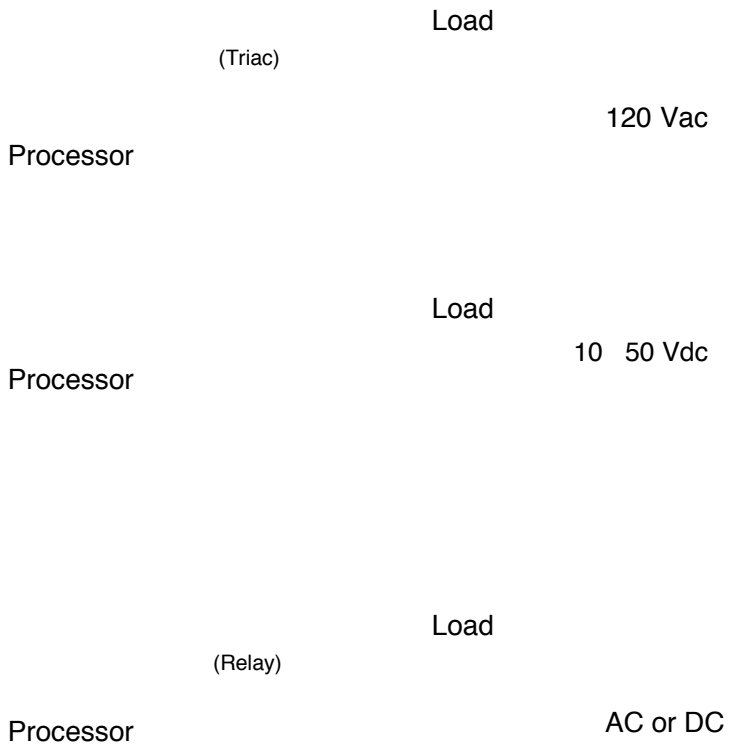
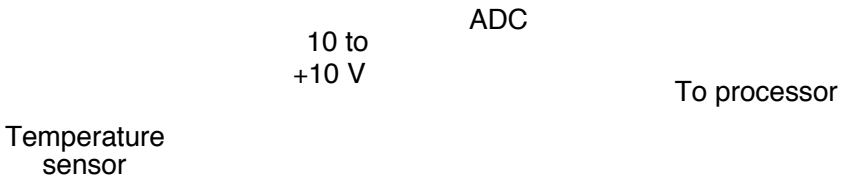


Figure 12.15
PLC output module circuits.

Figure 12.16
Analog I/O modules.



(a) Analog input module

(b) Analog output module

- *High-speed counter module*— Counts the number of input pulses for a fixed period of time.
- *PID module*— An independently running PID self-contained controller (PID control can also be implemented with software, as described later in this chapter).

PLC Bus

The **PLC bus** (Figure 12.13) are the wires in the *backplane* of a PLC modular system, which contains the data bus, address bus, and control signals. The processor uses the bus to communicate with the modules.

PLC Setup Procedure

The PLC evolved as a replacement for hard-wired relay logic and was designed to be installed, programmed, used, and maintained by technicians who were familiar with industrial relay and switching circuits. For this reason, manufacturers of PLCs developed a programming language that allows the user to program the desired control logic in the form of a ladder diagram. The ladder diagram may be entered into the system by either drawing it on the PC screen or using a specially adapted switch box (called a *hand-held programmer* (HHP)). If the PC is used, special software must be installed before the ladder diagram can be entered.

Assuming that the ladder diagram has been entered into a PC (or PC-like terminal device), the next step is to connect the PC to the PLC. In most cases, simply connect the two units with a cable or an interface circuit. A more sophisticated system may have a number of PLCs connected to a single PC on a LAN (local area network). With this system, the PC can communicate with any PLC on the network. Once connection has been established, the program is downloaded from the PC into the PLC.

With the program loaded, the PLC is ready to operate, and in any real application, the next step is testing and debugging. Testing can be done one of two ways: the total software approach, where the inputs are simulated from the programming terminal (PC), or a more real-world approach, where the inputs and outputs are connected to their actual destinations. In either case, the PC screen becomes a "window" into the workings of the program; if a problem becomes apparent, the program can be corrected immediately and testing continued until all bugs are worked out. When the testing is complete, the PC can be disconnected, and the PLC will continue to execute its stored program as an independent unit.

PLC Operation

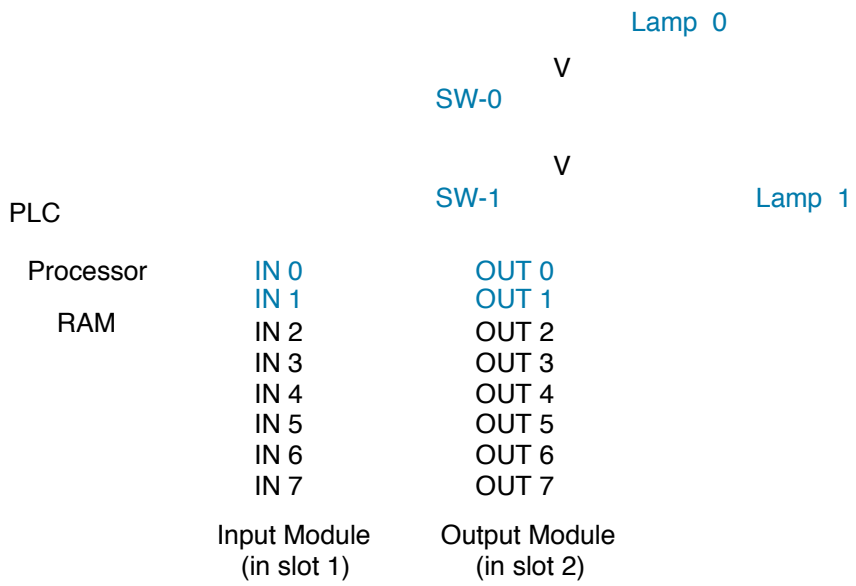
The PLC accomplishes its job of implementing the ladder logic control program in typical computer fashion—one step at a time. To see how it does this, let's examine a simple PLC application of turning lamps on and off [Figure 12.17(a)]. Notice that the PLC has one input module and one output module. Two external switches (SW-0 and SW-1) are connected to the PLC via terminals *IN-0* and *IN-1* of the input module. Two terminals of the output module (*OUT-0* and *OUT-1*) drive two indicator lamps (LAMP-0 and LAMP-1). The ladder diagram for this application has only two rungs [Figure 12.17(b)]. The top rung will light LAMP-0 if *both* SW-0 and SW-1 are closed. The bottom rung will light LAMP-1 if *either* SW-0 or *OUT-0* are closed. (The *OUT-0* contacts can be thought of as NO relay contacts of coil *OUT-0* in rung 1.)

Assume that the program has been loaded into the PLC and is residing in the

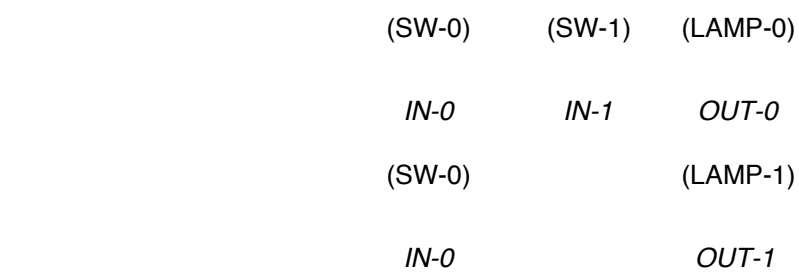
program section of memory. When the PLC is set to **run mode,** program execution begins. Execution is accomplished as a series of steps, which are repeated over and over. One complete cycle of all these steps is a **scan.** The steps are as follows:

1. The PLC reads in data from the input module and stores the values of all eight inputs, as a word, in the Input Data section of memory (Figure 12.18). Each bit in the word represents the present status of one external switch.

Figure 12.17
A simple PLC application and ladder diagram.

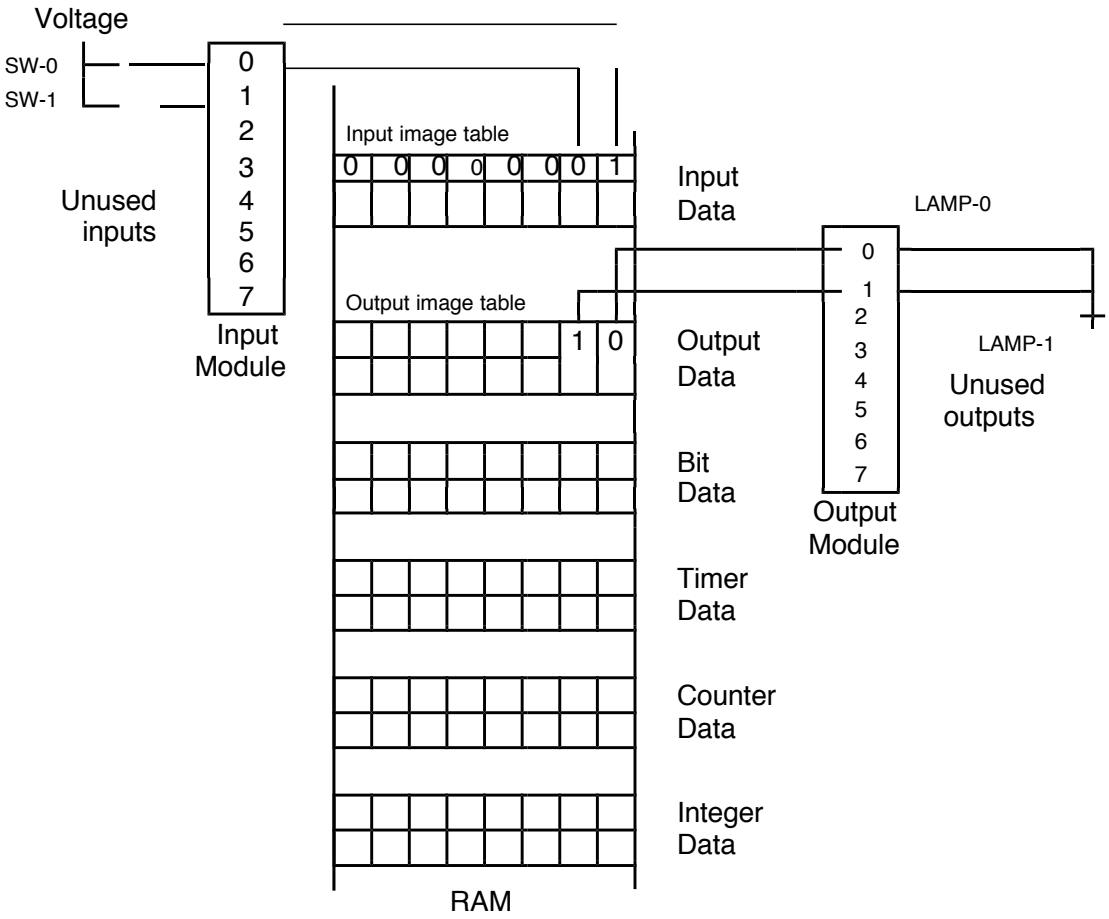


(a) Hardware



(b) Ladder diagram

Figure 12.18
Data in memory, showing the relationship between memory position and the I/O terminals.



2. The PLC turns its attention to the ladder diagram, starting with the first rung. The first two symbols in the rung are input switches, so the processor reads their status *from RAM*. It then evaluates the logic indicated by the rung. In this case, SW-0 is on, and SW-1 is off, so *OUT-0* will *not* be energized. This status of *OUT-0* is stored as a 0 bit in the Output Data section of RAM (Figure 12.18).
3. After completely dealing with the first rung, the PLC turns its attention to the second rung. The status of *IN-0* and *OUT-0* are read from RAM. Using these data, the PLC performs the rung logic, which is an OR operation between *IN-0* and *OUT-0* (recall that *OUT-0* is the output of the top rung and is presently off). In this case, because SW-0 is on, *OUT-1 will* be energized, and this result is stored as a 1 bit in the Output Data section of RAM. This completes action on the second rung. If there were more rungs, each would be evaluated in sequence until all had been executed.
4. As a final action of the scan, the updated output data in RAM are sent to the output module, causing all eight output terminals to be updated at once. In this case, LAMP- 0 would be off, and LAMP-1 would be on. Now the PLC will go into a rest state for a few seconds or less until the designated scan time has elapsed, and then the whole process starts over again with step 1.

Looking over the process just described, some observations are in order. First, the PLC accomplishes its control mission in the same way that all digital controllers work — by executing a loop (scan) over and over again. (For the PLC, however, the looping action is automatic and does not have to be programmed.) For each scan, inputs are read in, outputs are calculated based on the inputs, and then the outputs are sent out to the real world. This means that as much as one scan time's delay may be between an input and an output. Such a delay is not present in a true hard-wired relay circuit where the only delay would be the propagation time of the relays. This leads to the second observation: The order of the rungs in the ladder diagram may be important because the rungs are executed sequentially from top to bottom.

A final observation of the PLC hardware operation is that, for all practical purposes, it can do many independent control operations at the same time. We may be conditioned to think of a computer as being able to do only one thing at a time, albeit very rapidly. And, as we have seen, the PLC does execute its program one rung at a time. However, each rung is a separate circuit, and each control application (as far as the PLC is concerned) is only a set of rungs. Therefore, you could have one PLC running three different control applications at the same time. Each application would have its own assigned I/O terminal and its own set of rungs in the program. Within one scan cycle of the program, the PLC would update the three sets of outputs. This is similar to the chess master who plays three opponents simultaneously by going down the line, making one move on each board.

12.3 PROGRAMMING THE PLC

Ladder Diagram Programming

The most common way to program the PLC is to design the desired control circuit in the form of a relay logic ladder diagram and then enter this ladder diagram into a programming terminal (which could be either a PC, a dedicated PLC programming terminal, or a special handheld switch box). The programming terminal is capable of converting the ladder diagram into digital codes and then sending this program to the PLC where it is stored in memory. There are other ways to program a PLC besides ladder diagrams, and these will be introduced later in this chapter.

There are currently many manufacturers of PLCs, and though they all have basic similarities, the details of the hardware and software will, of course, be different. The following discussion on programming will be kept as general as possible but is patterned after the Allen-Bradley line of PLCs. This seems a logical choice because Allen-Bradley is one of the pioneers in the PLC field and its products are respected and well used by industry.

The PLC deals with two kinds of data that are maintained in memory: a **program file** in RAM or EEPROM, which contains the ladder logic program, and **data files** in

RAM, which hold the changing data from the control application, such as switch settings. Table 12.1 lists some of the more common types of data files. Each data file occupies a portion of RAM memory and consists of any number of 8- or 16-bit words. This data arrangement is illustrated in Figure 12.18.

The Input and Output Data files are handled in a special way. The PLC automatically transfers the on-off condition of all input switches directly to the Input Data file in RAM as part of the scan. This is called making an *input image table* (as indicated in Figure 12.18). Notice that a closed switch becomes a 1 in RAM and an open switch becomes a 0. Similarly, data in the Output Data file (called the *output image table*) are automatically transferred to the output-module terminals.

Thus, the PLC program actually deals with only the image data in their various data files. That is why each device on the ladder diagram—be it switch, relay, timer, or whatever—is identified by its address in RAM. Even though the programmer may think of a certain switch as being, say, an overflow limit switch, the software thinks of it as being a particular bit in a particular data file.

There is an important fundamental difference between a relay logic ladder diagram and a PLC ladder diagram. *The relay ladder diagram is basically a wiring diagram*, and a rung becomes active when current can flow from one power rail to the other; that is, if there is a path for the current through the contacts, then the load will be energized. On the other hand, *the PLC ladder diagram is basically a logic diagram* consisting of symbols (called input and output instructions), and a rung becomes continuous when there is a logical TRUE path from rail to rail. Referring to Figure 12.17(b), if there is a continuous path of TRUE Input instructions on the left side of a rung, then the Output instruction (on the right side) will become TRUE. Therefore, writing a PLC program involves placing instructions in rungs so as to obtain the desired result. Naturally, you can only use instructions that are supported by the language being used, but most PLCs support the basic components such as switches, relays, timers, counters, and sequencers. We will discuss each type of instruction next.

TABLE 12.1

Types of Data Files

	Data file	Description
I	Input Data	The current status of externally connected switches
O	Output Data	The status of those devices specified as outputs
B	Bit Data	The “scratch pad” to store individual bits
T	Timer Data	The data associated with timers
C	Counter Data	The data associated with counters
R	Control Data	The data associated with sequencers and other devices
N	Integer Data	Numbers, such as temperatures (in binary form)

Bit Instructions

Switches and relays are referred to as **bit instructions** because it takes only 1 bit to describe if a switch is open or closed. There are two kinds of switch instructions: one represents a NO switch and the other a NC switch. The following are the symbol and instruction for the NO switch:

NO instruction: —] [— (EXAMINE IF ON, Allen-Bradley)

When this symbol is placed in the ladder diagram, it behaves as a NO switch, push button, or set of relay contacts. It can be activated either by a real-world switch or another logical switch on the ladder diagram. In either case, when activated, the NO instruction in the ladder diagram goes TRUE. The state of the NO instruction is maintained by a particular bit in memory, where a 0 means off (FALSE) and a 1 means on (TRUE). The address of this bit is placed on the ladder diagram next to the instruction symbol.

The following are the symbol and instruction for the NC switch:

NC instruction: —]/[— (EXAMINE IF OFF, Allen-Bradley)

When this symbol is placed in the ladder diagram, it behaves logically in the same way any NC switch does—that is, it is normally TRUE and when activated goes FALSE (it behaves as a logical inverter). Care must be taken when using this instruction in conjunction with a real-world NC switch because it may not behave as you think. The problem is that people lose sight of the fact that a real-world NC switch and the NC instruction are *not* the same entity but, rather, *one drives the other;* that is, the physical switch controls the state of the NC instruction. Suppose you actually wired a real NC switch to the PLC and used it to control an NC instruction. Then, if the real switch was activated, it would send a 0 to the PLC, and the logical NC instruction would invert this FALSE to a TRUE, which may be backward from what you expected. To avoid this pitfall, remember that the real-world switch may drive the logical switch, but they are *not* the same thing.

The following symbol represents a relay coil or simply an output signal:

OUTPUT —( )— (OUTPUT ENERGIZE, Allen-Bradley)

If there is a continuous TRUE path through the rung, then the OUTPUT will go TRUE; thus, if this symbol represents a relay coil, the relay will energize. What really happens inside the PLC is that the bit in memory corresponding to this instruction goes to a 1. This bit may be used to drive an actual hardware relay through an output module or to control an instruction in another rung, or both. In fact, there is no limit to the number of times this bit can be used by other instructions in the program (which is another advantage PLCs have over hard-wired relay logic).

The operation of the three bit instructions are summarized in Table 12.2.

TABLE 12.2

Summary of Bit Instructions

If the data file bit Is	NO instruction —] [—	NC instruction —]/[—	OUTPUT — ( )—
Logic 0	FALSE	TRUE	FALSE
Logic 1	TRUE	FALSE	TRUE

In a PLC ladder diagram, each symbol is accompanied by its RAM address (Figure 12.19) because this address tells the PLC where to find the present logical state of the symbol. Recall that the logical state of each symbol is always maintained in RAM. Titles of real-world components that the symbols represent, such as *Overload Sw*, are frequently added to improve clarity, but the PLC itself deals strictly with addresses. The address of an individual bit may be specified by three quantities: the file type, word number, and position of the bit within the word.* As previously stated, I/O signals usually have a direct relationship between their address and the physical connection point on an I/O module. For example, in the top rung of Figure 12.19, the address of *Flow sw* is given as I : 1/0. The I stands for Input Data file, the 1 means that the input module is plugged into slot 1 of the PLC chassis, and the 0 means that the switch is wired to terminal 0 of the module. The following is a more general interpretation of the *Flow sw* instruction address:

Type of data file (Input in this case)
Address of word within the data file (or I/O module slot #)
Bit position within the word, if appropriate (or terminal # of I/O module)
I:1/0

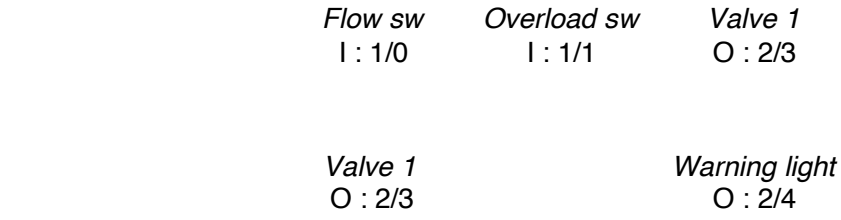


Figure 12.19
A simple PLC program.

*The address system used in this chapter is a simplified version of the Allen-Bradley system.

The second NO instruction on the first rung (Figure 12.19) represents a switch wired to terminal 1 of the same input module used by *Flow sw*. The OUTPUT instruction activates terminal 3 of the output module plugged into slot 2.

The operation of the top rung in Figure 12.19 is as follows: The *Valve 1* output will go TRUE when there is continuity through the rung. Continuity will be established when both NO instructions are TRUE—in other words, when the contacts of both *Flow sw* and *Overload sw* are closed.

Operation of the second rung illustrates two important concepts. First, notice that the address of the NC instruction is the same as the OUTPUT instruction in the top rung, meaning that the NC instruction will be controlled by the *Valve 1* bit (located at address 0:2/3). Second, because it is an NC instruction, it will go TRUE only when *Valve 1* is FALSE. So *Warning light* will come on when the valve goes off.

Timers

The Timer instruction provides a time delay, performing the function of a time-delay relay. Examples of using time delays include controlling the time for a mixing operation or the duration of a warning beep. The length of time delay is determined by specifying a preset value. The timer is enabled when the rung conditions become TRUE. Once enabled, it automatically counts up until it reaches the Preset value and then goes TRUE (and stays TRUE). The symbol for the Timer instruction is a box or a circle with the pertinent information placed in or about it, as shown and explained in Figure 12.20. Notice that the Timer instruction is placed in the rung. When *Switch* I : 1/3 goes TRUE, the timer starts counting. Five seconds later, when the Accumulator value reaches the Preset value, the DN (done) bit goes TRUE. Notice that the NO instruction in the second rung has the DN bit from Timer T : 1 as its address. Therefore, it will go true after 5 s, causing the OUTPUT instruction at 0 : 2/4 to turn on a light.

In Figure 12.20, the timer is acting as an on-delay relay because it starts the delay when the rung goes TRUE. Off-delay timer instructions are also available.

EXAMPLE 12.4

A batch process—which involves filling a vat with a liquid, mixing the liquid, and draining the vat—is automated with a PLC. Figure 12.21(a) shows the hardware. The specific sequence of events is as follows: When the start button near the process is pushed:

1. A fill valve opens and lets a liquid into a vat until it's full.
2. The liquid in the vat is mixed for 3 min.
3. A drain valve opens and drains the tank.

Draw the ladder diagram for the PLC program.

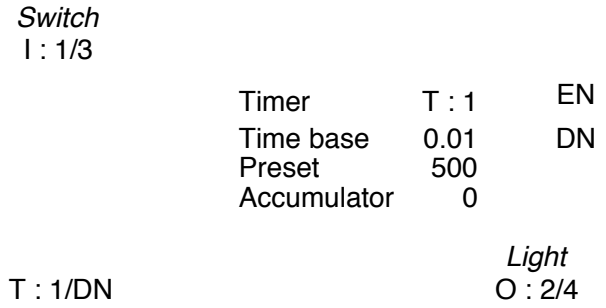


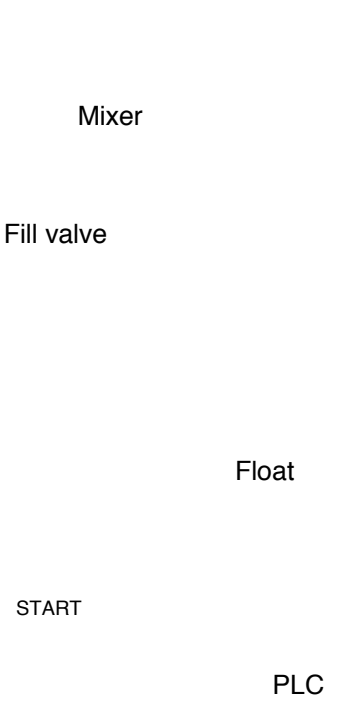
Figure 12.20
The timer instruction.

T : 1	The timer address (actually the first of three addresses in RAM) where the first address holds the status bits EN, TT, and DN; the second address holds the preset value; and the third address holds the accumulator value.
Time base	The value of 0.01 means that each count corresponds to 0.01 seconds.
Preset	The value of 500 means that the delay will last 500 counts, which in this case is 5 s (0.01s · 500 = 5 s).
Accumulator	Holds the value of the current count.
EN	A bit that is TRUE as long as the timer rung is TRUE.
TT	A bit that is TRUE as long as the timer is counting, that is, is TRUE for the time delay.
DN	A bit that goes TRUE when the timer is done—in other words, when the count gets to the preset value.

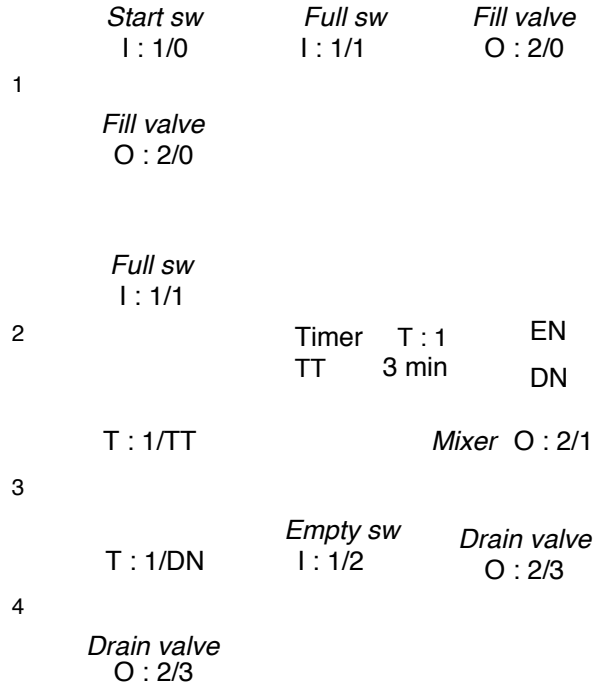
SOLUTION

Figure 12.21(b) shows the completed ladder diagram, which is explained rung- by-rung:

1. Rung 1 activates the fill valve and will go TRUE (and seal) when the local-control start switch is momentarily activated *and* the tank is less than full. The physical float-activated NO switch (Full switch) drives the *Full sw* NC instruction in rung 1; therefore, the NC instruction will be TRUE as long as the vat is not full (it changes to FALSE when the vat is full). As long as the entire rung is TRUE, the OUTPUT instruction (*Fill valve*) will be TRUE, which turns on the actual fill valve. (During the fill period, none of the other rungs are TRUE.)



(a) Hardware



(b) Ladder diagram

Fi
Us

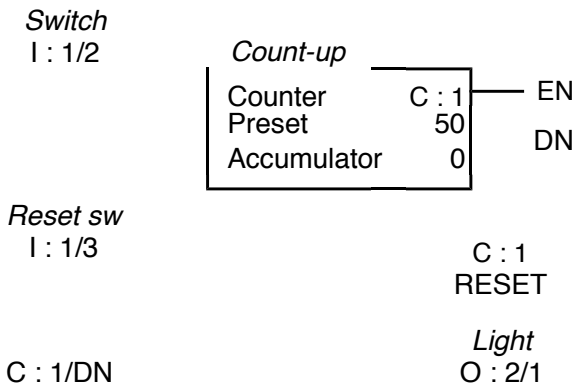
2. Rung 2 activates the 3-min mixing timer. When the vat is full, the physical full switch closes (goes to on), so the NO instruction (*Full sw*) goes TRUE, which starts Timer T : 1 (which has been programmed to provide the 3-min mixing time).
3. Rung 3 controls the mixer motor. As long as the TT bit of the timer is TRUE, the OUTPUT instruction *Mixer* will be TRUE, which activates the mixer motor.
4. Rung 4 activates the drain valve. After 3 min, the timer "times out," and the DN bit (T : 1/DN) goes TRUE, making the first instruction in rung 4 go TRUE. The NC instruction (*Empty sw*) will also be TRUE because the vat is still full (that is, the Empty switch remains unactivated). Therefore, rung 4 will be continuous, causing the OUTPUT instruction *Drain valve* to go TRUE, which opens the drain valve, allowing the liquid to leave. Here's where it gets interesting: As soon as the liquid starts to drain, the full switch will deactivate, making rung 2 go FALSE, which causes the timer's DN bit to go FALSE. This latter action will cause the first instruction in rung 4 to go FALSE. So, to keep rung 4 TRUE until the vat has completely drained, the T : 1/DN instruction has a parallel branch instruction, which is activated by *Drain valve* itself. In effect, rung 4 is latched on until *Empty sw* activates and breaks the seal.

Counters

A Counter instruction keeps track of the number of times some event occurs. The count could represent the number of parts to be loaded into a box or the number of times some operation is done in a day. Counters may be either count-up or count-down types. The Counter instruction is placed in a rung and will increment (or decrement) every time the rung makes a FALSE-to-TRUE transition. The count is retained until a RESET instruction

(with the same address as the Counter) is enabled. The Counter has a Preset value associated with it. When the count gets up to the Preset value, the output goes TRUE. This allows the program to initiate some action based on a certain count; for example, after 50 items are loaded in a box, a new box is moved into place. The symbol for the Counter may be a box or a circle, with all pertinent data placed in or about it, as shown and explained in Figure 12.22.

The ladder diagram in Figure 12.22 is a simple circuit that counts the number of times that *Switch* (address I : 1/2) is opened and closed. The Counter can be reset at any



C : 1	The counter address (actually the first of three addres the first address holds such bits as CU and DN; the s address holds the preset value; and the third address accumulated value.
Preset	When the counter gets to the Preset value, it makes t TRUE and keeps on counting.
Accumulator	Holds the value of the current count.
EN	Goes TRUE when the counter rung is TRUE.
DN	Stands for *done*—goes TRUE when the count meets the PRESET value.
RESET	A seperate instruction with the same counter address goes TRUE, the Accumulator resets to zero (resets th

time by closing *Reset sw* because this action activates the RESET instruction. The third rung uses the DN bit from the Counter to turn on a light when the count gets to 50 or higher.

Sequencers

The Sequencer instruction is used when a repeating sequence of outputs is required (recall the example of a dishwasher that cycles through predetermined steps). Traditionally, electromechanical sequencers (Figure 12.10) were used in this type of application (where a drum rotates slowly, and cams on the drum activate switches). The Sequencer instruction allows the PLC to implement this common control strategy.

Figure 12.23 shows the PLC Sequencer instruction. Operation is as follows: The desired output-bit patterns for each step are stored (sequentially) as words in memory—as usual, each bit in the word corresponds to a specific terminal of the output module. Every time the Sequencer instruction steps, it connects the next output pattern in memory to the designated output module. The address of the first pattern (word) in the sequence is given in the Sequencer instruction as the Sequencer File Adr (which is address B : 1 in Figure 12.23). The Sequencer shown in Figure 12.23 steps when the rung makes a FALSE-to-TRUE transition. Other versions of the Sequencer instruction allow the programmer to insert a time value for each step.

EXAMPLE 12.5

A process for washing parts requires the following sequence:

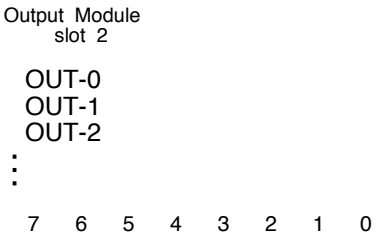
1. Spray water and detergent for 2 min (*Wash cycle*).
2. Rinse with water spray only for 1 min (*Rinse cycle*).
3. Water off, air blow dry for 3 min (*Drying cycle*).

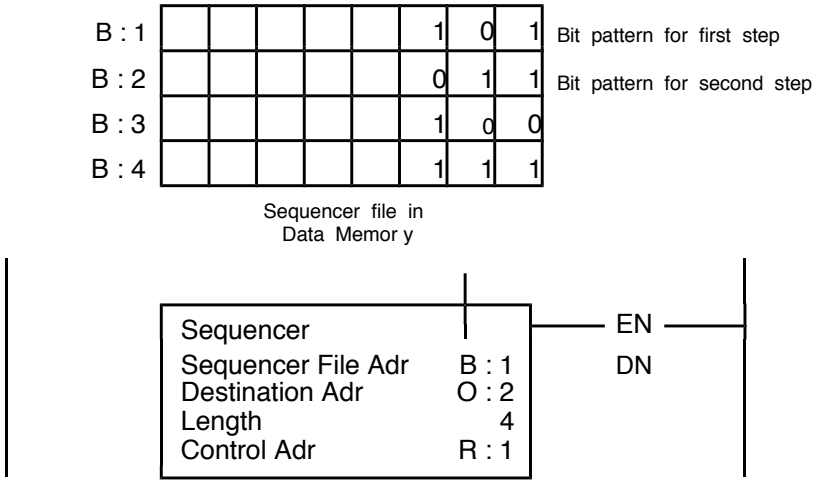
The sequence is started with a toggle switch. Draw the ladder diagram for this process (using the Sequencer instruction).

SOLUTION

The completed ladder diagram (Figure 12.24) consists of six rungs. Rungs 1 and 2 generate the 2-min time delay for *Wash cycle*. Rungs 3 and 4 generate the 1-min time delay for *Rinse cycle*, and rung 5 generates the 3-min time delay for *Drying cycle*. Rung 6 has the Sequencer instruction. The Sequencer instruction specifies that the Sequence file starts at address B : 1, that the file is three words long, and that the output module is in slot 2. Looking at the Sequence file, we see that bit 0 controls the water valve, bit 1 controls the detergent valve, and bit 2 controls the blow-dry fan.

Figure 12.23
The Sequencer instruction.



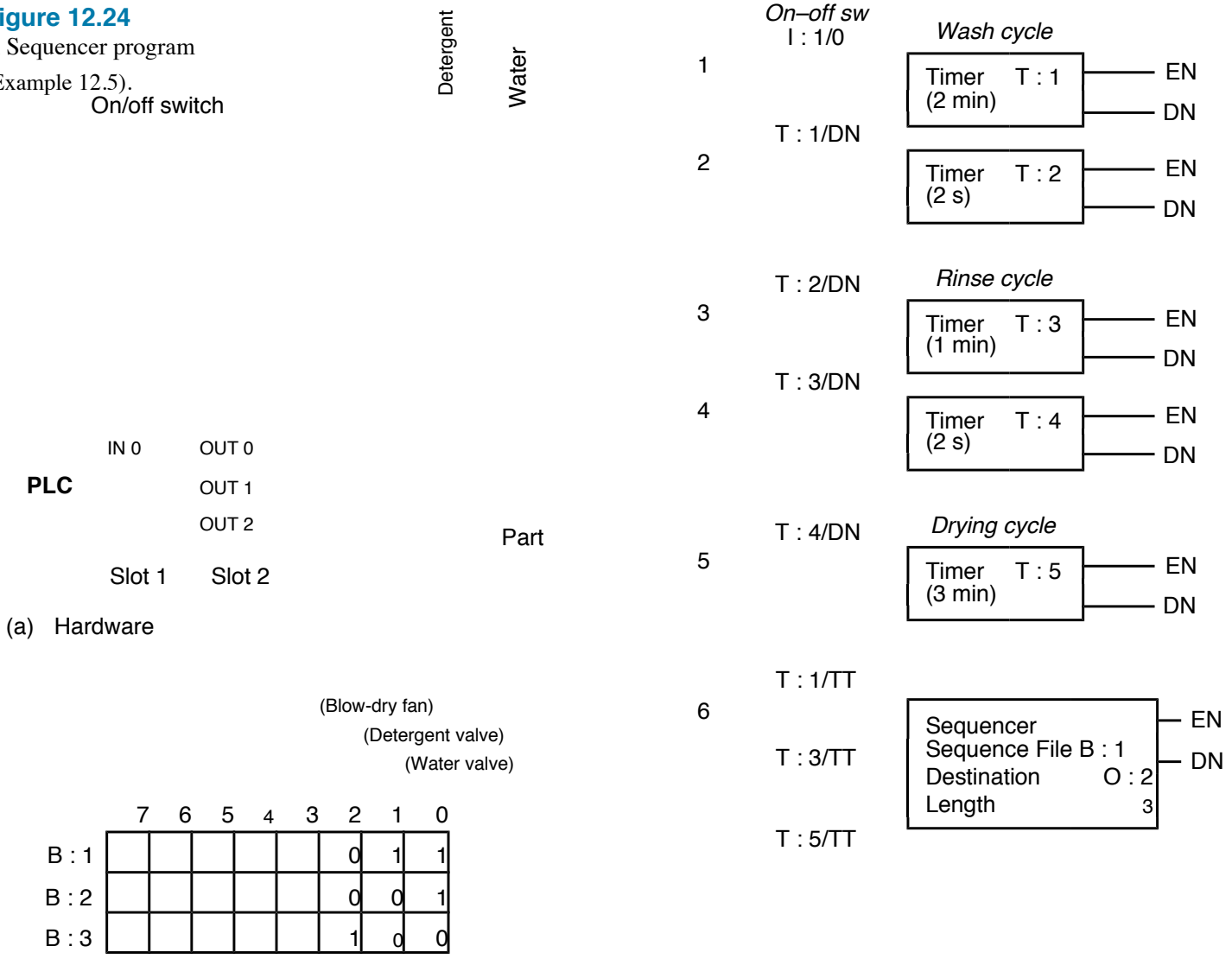


Sequencer File Adr	The first address of the Sequence file, which stores the sets of outputs.
Destination Adr	The output address (or slot) to which the Sequence file words are transferred with each step.
Length	The length of the Sequence file, that is, the number of steps in the sequence.
EN	Goes TRUE when the Sequencer rung is TRUE.
DN	Stands for *done*—goes TRUE when it has operated on the last word in the Sequence file.
Control Adr	Address that stores the control bits (EN, DN) and words (length) of the sequencer.

Operation Overview

The program consists of five timer rungs and a sequencer rung. The timers are activated, one after the other, down the program—that is, each timer, when fin- ished, starts the timer next in line. The outputs of Timers T : 1, T : 3, and T : 5 are used to step the Sequencer instruction. The Sequencer instruction presents three output-bit patterns to module 2.

Figure 12.24
A Sequencer program (Example 12.5).



(b) Sequencer File

(c) Ladder diagram

Detailed Operation

Operation of the ladder program is as follows: The action starts on rung 1 when the operator switches the on-off toggle switch to on. This provides a FALSE-to-TRUE transition that starts Timer T : 1 (*Wash cycle*), causing its timing bit (TT) to go TRUE for 2 min. This same timing bit (T : 1/TT), makes rung 6 go TRUE, stepping the sequencer to its first position (connecting the word at address B : 1 to output module 0 : 2). Notice that bit 0 and bit 1 of word B : 1 are 1s, which cause terminals OUT-0 and OUT-1 to go on, turning on the water and detergent valves.

After 2 min, the DN bit of Timer T : 1 (T : 1/DN) goes TRUE, which starts the next Timer (T : 2 in rung 2). This timer is only 2 s long, and its purpose is to create a short gap between the steps (to allow the sequencer rung to reset).

When Timer T : 2 is done, its DN bit (T : 2/DN) starts the 1-min Timer T : 3 (*Rinse cycle*). Timer T : 3 timing bit (T : 3/TT) causes the Sequencer instruction to advance to the second step (B : 2). Notice that in word B : 2, only bit 0 is a 1, causing OUT-0 to remain on (leaving the water on, but turning off the detergent).

Following in the same manner, Timer T : 4 provides a short gap, and then Timer T : 5 advances the sequencer to the third and last position (B : 3) for the 3-min *Drying cycle*. Notice that bit 2 of word B : 3 becomes OUT-2, and turns on the blow-dry fan.

There are many possible ways to program this sequence. The method chosen (Figure 12.24) is not necessarily the shortest, but it is a clear, uncomplicated solution that in real life is more valuable than a clever short program (that no one can understand except the programmer!). Also, remember that extra rungs on a PLC ladder program do not mean more hardware (it does mean a slightly longer scan time and memory usage, which in most cases is not an issue).

Using a PLC as a Two-Point Controller

Example 12.6 shows how a PLC can be used to implement a control strategy that requires decisions based on analog values. Although it is a simple example, it demonstrates the use of analog inputs and instructions that deal with analog values.

EXAMPLE 12.6

The temperature in an electric oven is to be maintained by a 16-bit PLC at approximately 100°C, using two-point control (actual range: 98-102°). Figure 12.25(a) shows the system hardware: an oven with an electric heating element driven by a *contactor* (high-current relay), an LM35 temperature sensor (produces 10 mV/°C—see Chapter 6), an operator on-off switch, and the PLC. The PLC has a processor and three I/O modules: a discrete input module (slot 1), a 16-bit analog input module (slot 2), and a discrete output module (slot 3).

Draw the ladder diagram for this system.

SOLUTION

We are asking the PLC to perform the job of a thermostat: When the oven temperature falls below 98°, the heating element will turn on and stay on until the

temperature reaches 102°. Thus, the PLC will be giving a discrete output (to the heater), based on an analog input (temperature).

The ladder diagram [Figure 12.25(b)] consists of three rungs that implement the control logic; however, recall that *before* the PLC executes the rungs, it reads in all input module data. In this case, it will read in the condition of the on-off switch from the input module in slot 1 (storing it to memory at address I : 1/0), and it will read in the value of the temperature through the analog input module in slot 2. The analog module contains an ADC that converts the analog voltage to a 16-bit word. The resolution of the module is given as

$$\text{ADC resolution} = 0.305176 \text{ mV} = 1 \text{ LSB}^*$$

Using this resolution, we can find the digital equivalent (in binary) of the lower- and upper-limit temperatures, 98° and 102°:

$$\text{input voltage for } 98° = 98° \cdot \frac{10 \text{ mV}}{1°\text{C}} \text{ Analog} = 980 \text{ mV}$$

$$\text{ADC output for } 98° = 980 \text{ mV} \cdot \frac{\text{LSB}}{0.305176 \text{ mV}} \text{ Output of LM35} = 3211 \text{ LSB} = 110010001011$$

$$\text{input voltage for } 102° = 102° \cdot \frac{10 \text{ mV}}{1°\text{C}} \text{ Analog} = 1020 \text{ mV}$$

$$\text{ADC output for } 102° = 1020 \text{ mV} \cdot \frac{\text{LSB}}{0.305176 \text{ mV}} \text{ Output of LM35} = 3342 \text{ LSB}$$

$$= 110100001110$$

We now know the digital values of the upper- and lower-limit temperatures. We will need these numbers in the program. The operation of the ladder program follows [refer to Figure 12.25(b)]:

1. Rung 1 determines if the temperature is below 98°C and will go TRUE when the conditions of the Compare instruction are met. This particular Compare

*This ratio would be found on the specification sheet and comes from the following analysis: Input voltage range = ±10 V = 20 V; output is 16 bits = 65,535 states (2s complement); LSB = 20 V/65,535 = 0.305176 mV.

F
i
A

ON

OFF

PLC

(a) Hardware

1

2

On–off sw
I : 1/0

3

(b) Ladder diagram

instruction compares two 16-bit words and goes TRUE if the value of word A is less than word B. In this case, word A is the actual temperature (as converted by the ADC and stored at address I : 2), and word B is the binary equivalent of the lower-limit temperature of 98°. Therefore, the rung will go TRUE if the oven temperature is less than 98°. The value of this rung is stored in bit 0 : 3/0.

2. Rung 2 determines if the temperature is above 102°C. This rung has another Compare instruction, but this one goes TRUE if word A is greater than word B. Notice that word A is again the actual temperature (from the ADC), and word B is the upper-limit temperature of 102°. Therefore, the rung will go TRUE if the oven temperature is greater than 102°. The value of this rung is stored in bit 0 : 3/1.
3. Rung 3 activates the heating element via the control logic that turns the heating element on and off—that is, the OUTPUT instruction *Heater* (0 : 3/2) directly controls the heating element. The rung will go TRUE if the on-off switch is on, *and* the temperature is not over 102° (notice this is an NC instruction), *and* the temperature is less than 98°. Once *Heater* is on, it is sealed on with the parallel branch 0 : 3/2 so that, even when the oven temperature rises above 98°, the rung will stay TRUE. With the heating element on, the oven temperature will eventually rise to above 102°, and so the NC instruction (*Temperature > 102°*) will go FALSE, breaking the seal and turning *Heater* off. The rung will stay FALSE until the temperature drops below 98°, and then the cycle starts over.

Advanced Instructions

So far, we have just examined the basic instructions supported by virtually any PLC. PLCs are getting more sophisticated, and as they do, their instruction sets expand. In fact, many newer PLCs more closely resemble a real-time process-control computer than simply a substitute for relay logic. Many of the advanced instructions operate on a digital word instead of a single bit. Some of these words may be digitized analog data (such as temperature) used with an analog I/O module, which is an ADC or a DAC. Counter and timer accumulation values are stored as words. In many cases, the purpose of processing these number quantities is to arrive at a yes-no decision, where the actual output of the program is still in terms of individual bits (that is, turning a motor on or off). Then too, it is also possible to output a whole digital word to be converted to an analog voltage by an I/O module. Such a voltage could be used to control a motor's speed.

The higher level instructions come in various categories:

- **Comparison instructions** compare two words and yield a single bit as a result. For example, a word may be tested to see if it is larger than another word or to see if it is equal to another word. These instructions are demonstrated in Example 12.6.

- **Math instructions** perform mathematical operations on words in memory. These operations may include addition, subtraction, multiplication, division, and others. A math instruction might be used to multiple a sensor input word by a constant, for scaling purposes.
- **Logical and shift instructions** perform logical operations such as AND, OR, and NOT and Right and Left Shifts on words stored in memory. For example, the AND instruction could be used to mask out certain bits in a word.
- **Control instructions** allow for such things as jumps and subroutines. For example, a Jump instruction, when TRUE, will cause the execution to jump ahead (or back) to any designated rung.
- **PID control** is available on some PLCs. In some cases, it is built into an I/O module; in other cases, it is programmed as an instruction in the ladder diagram. The PID instruction is covered in the next section.
- **I/O message and communications instructions** allow for PLCs to send and receive messages and data to other PLCs or a terminal (PC), for those applications where PLCs are networked together.

PID Instruction

PID control is becoming more commonly available on PLCs. In most cases, the PID function is implemented by an instruction and uses regular I/O (analog) channels. Another possibility is a separate PID hardware module, in which case it is actually an independently operating controller running under supervisory control of the PLC.

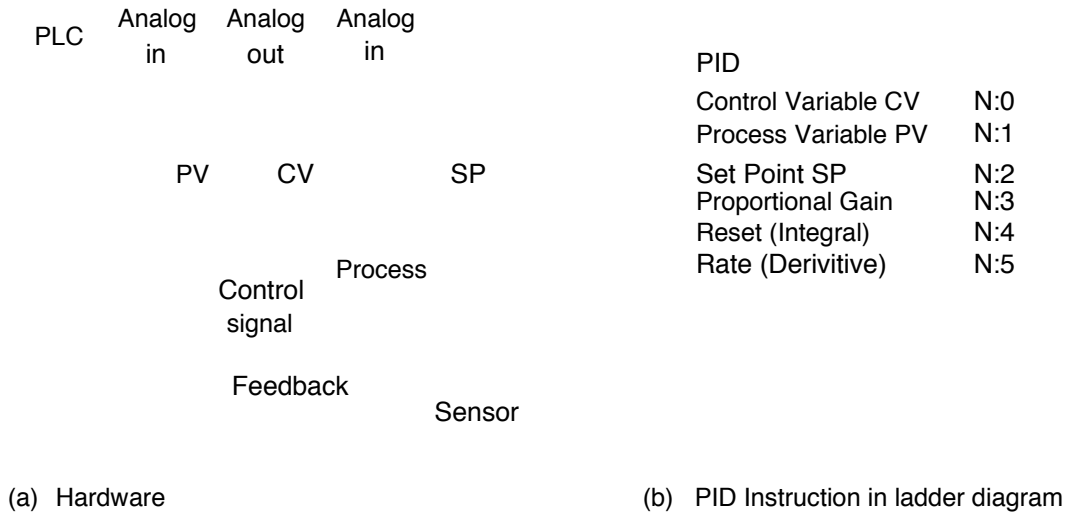
The PID instruction implements the PID control strategy discussed in Chapter 11. It would be used in a closed-loop system to control some physical parameter such as temperature, liquid level, or even motion (although a motion problem may require faster responses than the PLC can deliver).

A sample PID instruction is shown in Figure 12.26(b). It consists of a box or circle with the pertinent information placed in or around it. It is usually on a rung all by itself (there is no conditional logic on the left side of the rung), which means the PID instruction is always TRUE. Thus, as long as the PLC is running, the PID function will be trying to maintain some physical parameter at its set point. The hardware to support the PID instruction is shown in Figure 12.26(a) and consists of

- One I/O analog channel for the output signal *CV* (control variable)
- One input I/O channel (analog) for the feedback signal *PV* (process variable)
- Possibly an input I/O channel (analog) for the set point *SP* (However, the SP does not necessarily need a physical input, because it can be set by another instruction or is simply a constant.)

The instruction symbol specifies the addresses in RAM of the PID parameters. As shown in Figure 12.26(b), these memory locations store (at a minimum) the current values of *CV*, *PV*, and *SP*, as well as the PID constants for proportional, integral and derivative (K_P, K_I, K_D).

Figure 12.26
PID instruction.



(a) Hardware (b) PID Instruction in ladder diagram

Tuning the PID controller means finding the values of K_P, K_I and K_D that cause the system to remain stable, respond quickly to disturbances without excessive overshoot, and have a minimum of steady-state error. The method of tuning suggested by Allen-Bradley is similar to the continuous-cycle method discussed in Chapter 11 (see Example 11.6). This method is as follows:

1. Initially set $K_P = 1, K_I = 0, K_D = 0$
2. Slowly increase the gain (K_P) until the system goes into a steady oscillation, and record the period of oscillation (period = T_C)
3. Initial settings: set K_P to half of what was needed to go into oscillation, set $K_I = 1/T_C$, and set $K_D = T_C/8$.

Other PLC Programming Languages

Ladder diagram programs are highly symbolic and are the result of years of evolution of industrial control circuit diagrams. The PLC is a relative latecomer, and it was adapted to use traditional ladder logic. However, as we have noted, a PLC program is not actually a wiring diagram but a way to describe the logical relationship between inputs and outputs. There are now more concise ways to convey control logic than ladder diagrams. These programs look more like conventional computer programs, using words and mathematical symbols instead of pictures.

One type of PLC language used by Allen-Bradley is called simply *ASCII programming*, or "creating an ASCII archive file." Basically, this is a method of conveying a ladder diagram by using words instead of pictures. Each instruction is given a three-letter designation called a **mnemonic.** Table 12.3 lists the more common instructions. In Table 12.3, the first three mnemonics represent such devices as switches and relays, and the last three mnemonics specify how the devices are interconnected logically. In an ASCII

TABLE 12.3

ASCII Programming Instructions

Symbol	Mnemonic	Explanation
—] [—	XIC (examine if closed)	Goes TRUE when the contacts close.
—] / [—	XIO (examine if open)	Goes TRUE when the contacts open.
— ( )—	OTE (output energize)	Energizes the relay when TRUE.
——I——	BST (branch start)	Starts a parallel branch.
I_ -I	NXB (next branch)	Ends previous branch, and starts new branch.
————I	BND (branch ends)	Ends the parallel branches.

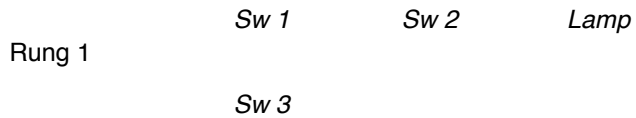


Figure 12.27
A ladder diagram and the corresponding ASCII program.

Program Instructions (ASCII)			Explanation
!RUNG 1			Identifies the rung.
SOR BST	XIC Sw 1 NXB		Start of the rung; branch starts and XIC Sw 1 is on the top branch, the next branch follows.
	XIC Sw 3 BND		On the next branch is XIC Sw 3, and then this branching ends.
	XIO Sw 2	OTE Lamp	On the main rung is XIO Sw 2, then a relay driving a lamp.
EOR			This rung is finished.

Program Instructions (Boolean)		Explanation
START	Sw 1	Switch 1 is first logical variable.
OR	Sw 3	Switch 3 is in parallel with Switch 1 (which is the same as an OR operation).
AND		The previous result will be ANDed with
NOT	Sw 2	the inverse of the state of Switch 2.
OUT	Lamp	The result of the logic equation is used to activate the Lamp output.

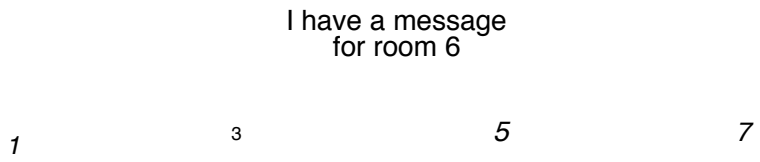
program, the ladder diagram is conveyed by placing the mnemonics and addresses in a certain order. These concepts are best shown by Figure 12.27, a simple ladder diagram followed by the corresponding ASCII program.

Another type of PLC programming language uses Boolean logic directly. This involves listing commands such as AND, OR, and NOT in such a way as to describe the program logic. An example of this approach is shown in Figure 11.27. Still others PLCs can be programmed with high level languages such as BASIC or C.

12.4 PLCS AND NETWORKS

Networks are increasingly being used to interconnect the components of control systems with each other and the rest of the factory. Physically, a network is a wire acting as an "electronic highway" that can pass messages between **nodes** (PCs and other electronic devices). Each node on the network has a unique address, and each message—called a **data packet**—includes the address of where it's going and where it came from. All data on the network is sent serially (one bit at a time) on one wire. The most common type of network uses the **bus topology,** which means that all the nodes tap into a single cable (see Figure 12.31). The bus type of network is analogous to a long hall in a hotel, with room doors on either side, as illustrated in Figure 12.28. Each room has a room number (address), and a message could be passed from one room to another by shouting it down the hall. For example, if the person in room 3 wanted to send a message to the person in room 6, he or she would open the door and shout, "I have a message for room 6 from room 3. The message is blah, blah, blah." All the other occupants would be listening at their doors, but only the occupant of room 6 would actually take note and receive the message. Besides conveying the general concept, this analogy illustrates two other points about a bus type of

Figure 12.28
Analogy for a bus type of network: a long hall in a hotel, where each room is a node.



It’s for me!

network: Every message is broadcast to every door, and only one message can be communicated at a time.*

There are three levels of networks that could be used by industrial organizations (see Figure 12.29), and all three levels can be interconnected so that any node can communicate with any other node in the building. The highest level is Ethernet. **Ethernet** is the traditional network used for interconnecting PCs and mainframe computers for the purpose of exchanging data and data files; this is known as the *information level*. Ethernet is good for communicating financial, inventory, and production data, but it may have difficulty dealing with a real-time process like a PLC responding quickly to a change in motor speed. The problem is that Ethernet is not a deterministic network. A **deterministic network** is able to assign priority to a message and guarantee that it will arrive in a given time. Ethernet may slow down significantly in times of heavy traffic.

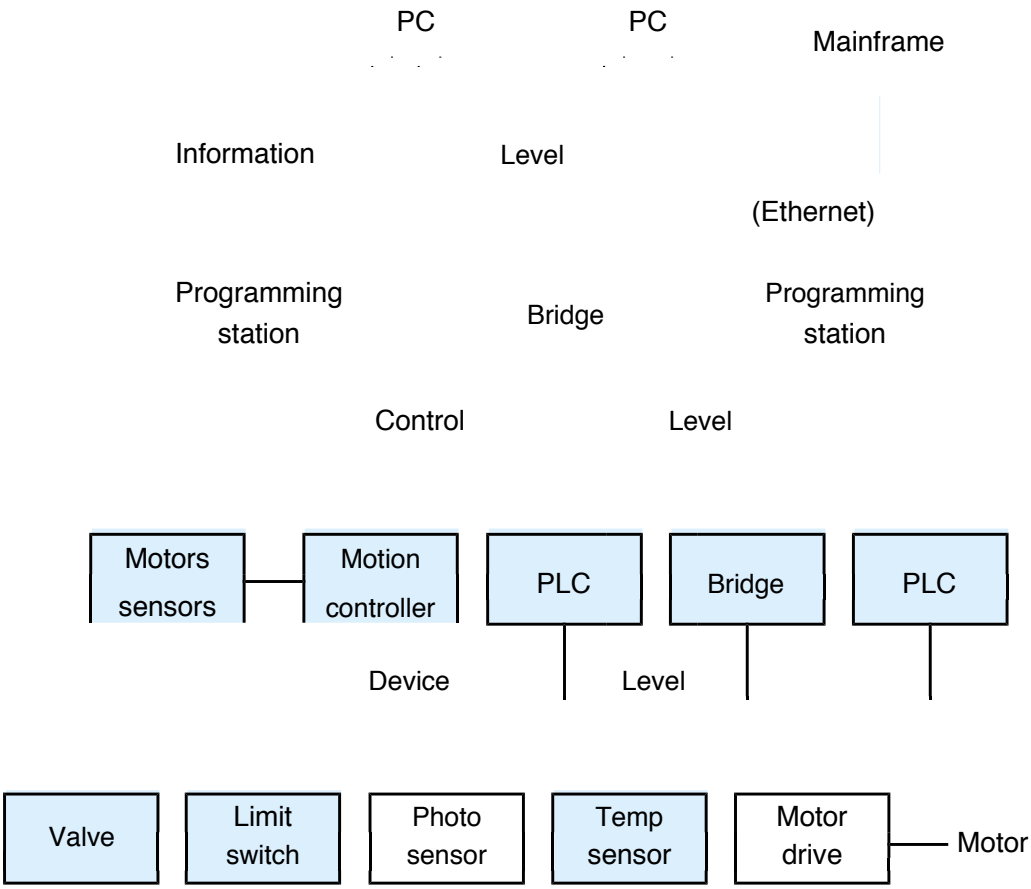


Figure 12.29
Three levels of networks.

*The integral constant K_I is usually called $Reset = 1/T_C$, and the derivative constant K_D is usually called $Rate = T_D$

*This is true for most networks that use a digital signal in *baseband* (no frequency multiplexing).

The second network level is known as the *control level*. A network at the control level provides deterministic communication and would be used to interconnect PLCs with other PLCs and with their supervisory computers and programming stations (typically PCs). A network at the control-level may use a *token* system* to guarantee that each node that needs to transmit will get an opportunity to do so. An example of this type of network is Control Net (Allen-Bradley).

The third network level is known as the *device level*. A **device-level network** is used to connect low-level devices such as sensors and actuators to a PLC. Traditionally, individual sensors are connected directly to the PLC, and it might seem like "overkill" to use a network for this job, but there are three good reasons for using a network:

1. A device network simplifies wiring. Figure 12.30 compares the traditional point-to-point sensor wiring with the device network. Clearly the network is a simpler system that uses less wire. In fact, many new cars use a device network to connect electrical devices (locks, controls, and the like); this reduces the amount of wiring needed.
2. With a network, the sensor data arrives in better shape. In the traditional point-to-point system, a low-level analog voltage may have to travel many feet. The signal is subject to attenuation and noise. Even the current-loop system is subject to noise and other losses. With a device network, the analog voltage is converted to digital *at the source* and transmitted as a digital message with error detection. There is no chance that the signal will be attenuated (as long as the digital data gets through), and any corrupted data can be identified as such and re-transmitted.
3. Network devices tend to be more "intelligent." Once a device has electronics built in to communicate on the network, it's a small step to build in more functionality with little additional cost. For example, a photo cell could send a message saying the light level has diminished, (indicating that the lens may be getting dirty or that some- one has bumped it out of position).

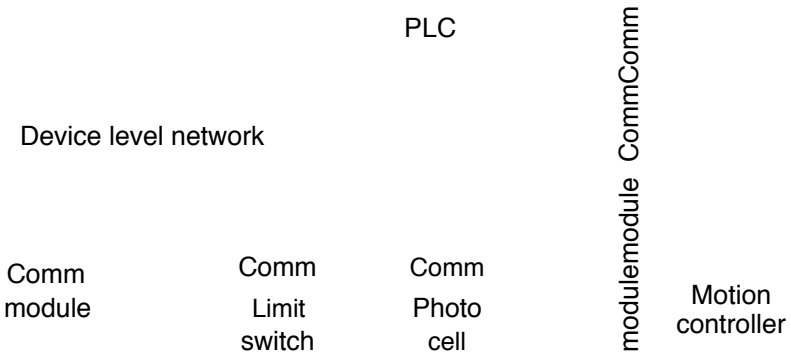
a) PLC connected to sensors and actuators using point-to-point wiring

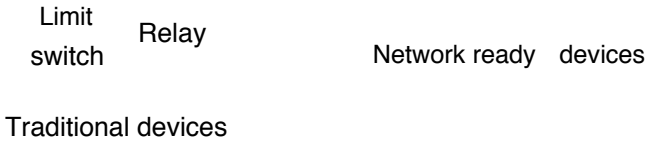
(b) PLC connected to sensors and actuators using a device-level network

Fi
Co

*A token is a small data packet that is passed from node to node. There is only one token, and a node can transmit one chunk of data only if it has the token. This system guarantees that each node gets a turn to send data.

Figure 12.31
Each device connected to the network must use a network interface circuit (communication module).





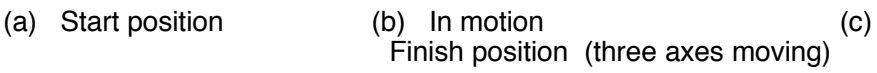
A device-level network may be able to communicate in different ways, depending on the needs of the application. Probably the most common system is polled I/O. An example of **polled I/O** occurs when a PLC asks (polls) a sensor for data only when it needs it. In a **change-of-state** system, a sensor would initiate a data transfer only when there was a change to report. For example, a switch on a furnace door would only report when the door was actually opened. In a *cyclic* system, the sensor would send its data at a predefined rate, such as once every 100 ms. Finally, in a *strobe* system, one message is delivered to many devices simultaneously.

More equipment and procedural steps are required for setting up a device network and getting it operational, compared with the traditional wiring system of Figure 12.30(a). First, each device must connect to the network through an interface unit (see Figure 12.31). Traditional simple devices such as limit switches, photocells, and temperature sensors must connect to the network through a communication module. The communication module enables the operator to set the address of the device, perhaps with a thumbwheel switch. More sophisticated devices, such as motor drive controllers and other "network ready" devices, have the network communication circuitry built in. Also, to make the system work, the PLC program may have to include instructions to send initialization commands at start-up to those devices that need it, and it must to be programmed to communicate with its devices through the network. Examples of device- level networks include **DeviceNet** (Allen-Bradley), and the **Fieldbus**.

12.5 MOTION CONTROLLERS

Motion controllers are control circuits specifically made to coordinate movement. A typical application would be a robot picking up an object from one place, moving to another place, and putting the object down. As shown in Figure 12.32, this action could involve

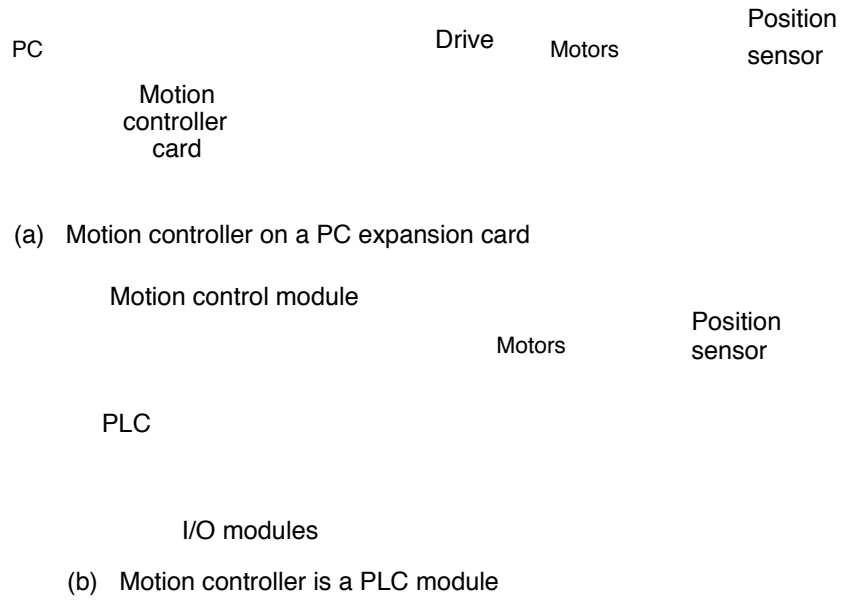
Figure 12.32
In this example of motion control, three axes are moving at the same time.



moving two or more axes at the same time. The motor for each axis would accelerate up to speed, travel at a fixed speed to the vicinity of the destination, and then decelerate to a stop. The speeds of both motors would be controlled so that they both arrived at the destination at the same time , even if that meant that they moved at different speeds.

A motion controller is basically a programmable digital controller similar to those already discussed. A typical controller is built on a circuit card and may be able to control 4 to 8 axes. The controller can be either open-loop or closed-loop. Open-loop systems have to use stepper motors. Closed-loop systems use DC or AC servo motors and require feedback position sensors, which are typically optical encoders. Outputs from the motion controller card provide the speed or torque signal to the motors (or motor drives); inputs to the controller come from the sensors. Two possible hardware configurations are shown in Figure 12.33. The system shown in Figure 12.33(a) is a PC with

Figure 12.33
Motion controller hardware.



a motion controller expansion card. The card can be programmed with special software that runs under Windows. In another configuration, shown in Figure 12.33(b), the motion controller is a module that runs under a PLC. In this case the PLC would download the motion parameters into the motion controller, and the motion controller would then work on its own.

Like virtually all digital controllers, a motion controller is basically a computer executing a program. What makes it a "motion controller" is that the hardware and software have been adapted specifically to facilitate controlling *motion*. A processor in a motion controller could be either a general-purpose microcontroller or a digital-signal processor (DSP). A **DSP** is a processor that is designed to support high-performance, repetitive, numerically intensive tasks. For example, DSPs may be able to do two multiply operations at the same time, can perform tight, quick loops, and have irregular instructions sets that allow several instructions to be encoded in the same instruction. In other words, a DSP is good for control operations where multiple-axis operation and quick response are required. However, regardless of the type of processor used by the motion controller, the basic setup and operation are the same:

1. By means of a PC and proprietary software supplied or specified by the manufacturer of the motion controller, a series of commands are entered that describe the desired motion.
2. The motion commands are processed by the software into "machine language" and downloaded into memory on the motion processor card.
3. When activated, the processor on the motion controller card executes the machine-language program loop . It is this program that takes over the job of actually processing the feedback signals and sending motor-control signals to the motor.

The motion controller may apply several control strategies in a sequence when moving to its target. A typical move would start by accelerating to a designated speed, then moving toward the target at that speed, and finally decelerating and possibly using PID to close in on the final position. This particular sequence is sometimes called *trapezoid* motion because a graph of the motion resembles a trapezoid (see Figure 12.34). Other ways of getting to the target (depending on the system) might be called *rapid* , *circular*, or *S-curve*. The programmer can specify whether more than one axis is to move at the same time, and if so, the controller will move them in a coordinated fash-

Figure 12.34
Graph of "trapezoid" motion

Time

ion, all finishing at the same time. If more than one move is specified in succession with no pause in between, the first move will blend smoothly into the second, and so on, until the target is reached.

There is not yet a standard accepted programming language for motion controllers, which means that most systems have their own unique language. An example of a simplified program for the Delta-Tau PMAC system follows.

Command	Explanation
#1->1000X	Assign motor #1 to *X*-axis.
#2->1000Y	Assign motor #2 to *Y*-axis.
CLEAR	Erase existing contents.
LINEAR	Specify motion mode (trapezoid).
TA500	Set acceleration value.
TS200	Set speed.
X0 Y0	Move to start position.
X10 Y0	Move to $x = 10$ and $y = 0$.
DWELL	Wait 500 ms.
X0 Y10	Move to $x = 0$ and $y = 10$.
X0 Y0	Move back to start position.

SUMMARY

One common type of control system governs the repetitious sequence of events. Traditionally, this kind of control was implemented with electromechanical relays, timers, and sequencers. Relays can be interconnected to provide some control logic—for example, two relays in series become an AND function. Time-delay relays will delay activating their contacts for some period of time (after being energized). Sequencers use a rotating cam to open and close contacts in some sequence.

A ladder diagram, a special kind of wiring diagram, was developed to document electromechanical control circuits. This type of diagram (which resembles a ladder) has two vertical wires (rails) on either side of the drawing; these wires supply the power. Each rung of the ladder diagram connects from one rail to the other and is a separate circuit, which typically consists of some combination of switches, relay contacts, relay coils, and motors. It is common for the coil of a relay to be in one rung and the contacts to be in another.

In the late 1960s, the programmable logic controller (PLC) was developed to replace electromechanical controllers. The PLC is a small, microprocessor-based, process-control computer that can be connected directly to switches, relays, small motors, and so on, and is built to withstand the industrial environment. The PLC exe- cutes a program that must be written for each specific application. The program is always in the form of a loop, which has the following pattern: The PLC reads and stores

the input switch and sensor data; it determines what the output(s) should be; it sends the output control signals to the devices to be controlled.

To use a PLC requires setting up the hardware and software. The hardware installation consists of wiring the PLC to all switches and sensors of the system and to such output devices as relay coils, indicator lamps, or small motors. The control program is usually developed on a PC, using software provided by the PLC manufacturer. This software allows the user to develop the control program in the form of a ladder diagram on the monitor screen. Once the program is complete, it is automatically converted into instructions for the PLC processor. The completed program is then downloaded into the PLC. Once the program is in the PLC's memory, the programming terminal can be disconnected, and the PLC will continue to function on its own.

Increasingly PLCs are being interconnected with networks. There are three levels of networks: the *information level* for office communications, the *control level* for interconnecting PLCs, and the *device level* for connecting sensors and actuators to the PLC.

Motion controllers are control circuits specifically made to coordinate movement. A typical motion controller works with a closed-loop system to coordinate the movement of two or more axes simultaneously.

GLOSSARY

analog I/O module A module that plugs into the PLC and converts real-world analog signals to digital signals for the PLC and vice versa.

bit instruction The basic PLC instruction that logically represents a simple switch or relay coil; the state of each switch or coil occupies 1 bit in memory.

bus topology A type of network wherein all the nodes tap into a single cable at different places, and the cable has a beginning and an end (that is, the cable is not in a ring).

change-of-state system A method of communicating on a network wherein a device sends a message only when something changes.

comparison instruction Compares two words and yields a single bit as a result.

control instruction Allows for such things as jumps and subroutines.

data file The collection of data in a PLC that represents the present condition of switches, counters, relays, and the like.

data memory The section of RAM memory in a PLC where the data file is stored.

data packet A message transmitted on a network; usually includes the address of where it's going, the address of where it came from, and the message itself.

deterministic network A network able to assign priority to a message and to guarantee that it will arrive in a given time.

device-level network A network used to connect low-level devices such as sensors and actuators to a PLC.

DeviceNet A type of device-level network developed by Allen-Bradley. See **device-level network.**

Digital-signal processor (DSP) A type of digital processor that is designed to support high-performance, repetitive, numerically intensive tasks.

discrete I/O I/O that deals with on-off devices. Discrete input modules are used to connect real-world switches to the PLC, and discrete output modules are used to turn on lamps, relays, motors, and so on.

electromechanical counter A device that counts events. Each time it receives a pulse, it increments a mechanical counter, which can then be read by an operator or used to activate some other process.

electromechanical relay (EMR) A device that uses an electromagnet to close (or open) switch contacts—in other words, an electrically powered switch.

electromechanical sequencer A device that provides sequential activation signals; it works by rotating a drum with cams, which activates switches.

EMR *See* **electromechanical relay.**

Ethernet The type of network used to interconnect PCs and mainframe computers for the purpose of exchanging data and data files.

Fieldbus A type of device-level network. See **device-level network**

I/O message and communications instructions Allow PLCs to send and receive messages and data to other PLCs or a terminal (PC), for those applications where PLCs are networked.

Input/Output modules The connection points where real-world switches, relays, and the like are connected to the PLC.

ladder diagram A type of wiring diagram (which resembles a ladder) used for control circuits; ladder diagrams typically include switches, relays, and the devices they control.

latching *See* **sealing.**

logical and shift instructions Perform logical operations such as AND, OR, and NOT and Right and Left Shifts on words stored in memory.

math instructions PLC mathematical operations on words in memory; these operations may include addition, subtraction, multiplication, division, and others.

mnemonic An English-sounding abbreviation for PLC logical instructions.

Motion controller A control circuit specifically made to coordinate movement. A typical motion controller works with a closed loop system to coordinate the movement of two or more axes simultaneously.

node Any unit connected to a network that can transmit and receive data.

normally closed (NC) contacts Relay contacts that are closed in the de-energized state and open when the relay is energized.

normally open (NO) contacts Relay contacts that are open in the de-energized state and closed when the relay is energized.

off-delay relay A time-delay relay that, after being de-energized, waits a specified period of time before activating.

on-delay relay A time-delay relay that, after being energized, waits a specified period of time before activating.

PID control A common control strategy. PID instructions are available on some PLCs.

PLC *See* **programmable logic controller.**

PLC bus The wires in the backplane of a modular PLC system that allows the microprocessor (within the PLC) to communicate with plugged-in modules.

plug-in module An interface circuit to the PLC, packaged as a separate module, which plugs into the PLC. Different modules support different I/O devices, from simple switch and relay output modules to ADC and DAC modules.

pneumatic time-delay relay A time-delay relay that generates the delay by allowing air to bleed out of a spring-loaded bellows.

polled I/O Probably the most common method of communicating on a network, wherein a PLC asks (polls) a sensor for data only when it needs it.

power rails Part of a ladder diagram, the two vertical lines on each side of the diagram, which represent power lines.

processor Part of the PLC that executes the instructions; it is a small, microprocessor- based computer.

program file Contains the program that the PLC executes. The program file is part of the PLC memory (the other part being data files).

programmable logic controller (PLC) A small, self-contained, rugged computer, designed to control processes and events in an industrial environment—to take over the job previously done with relay logic controllers.

program memory When applied to PLCs, the section of memory (RAM or EEPROM) where the PLC program is stored.

programming port A connector on a PLC through which the program is downloaded.

run mode The operational mode of the PLC when it is actually executing the control program and hence performing some control operation.

scan The event when the processor makes one pass through all instructions in the control program loop.

sealing A relay wired such that once energized its own contacts take over the job of providing power to the coil; it will stay energized, even if the original energizing signal goes away.

solid-state time-delay relay A time-delay relay that generates the delay with an elec- tronic circuit.

thermal time-delay relay A time-delay relay that generates the delay by allowing a bimetallic strip to heat and bend, which activates a switch.

time-delay relay A relay with an intentional delay action; that is, the contacts close (or open) in a specified period of time *after* the relay is energized (or deenergized).

EXERCISES

Section 12.1

Figure 12.35

1. Draw a relay wiring diagram for a circuit that implements the simple logic diagram shown in Figure 12.35. The circuit should be in the style of Figure 12.1(a).
2. Repeat Exercise 1 except draw the circuit as a ladder diagram.
3. A small house has three windows and two doors. Each window and door has a switch attached such that the contacts close when a door or window opens. Draw a ladder logic diagram that will turn on a light if one or more windows are open or if both doors are open.

4. In a processing plant, jars on a conveyer belt are cleaned out with a high-pressure air jet just prior to being filled (Figure 12.36). When a jar approaches the cleaning station, it activates a switch (with both NO and NC contacts). The conveyer belt stops for 10 s while the air jet is on; then the conveyer belt starts, and the jar moves

along. Draw a ladder logic diagram to control this process.

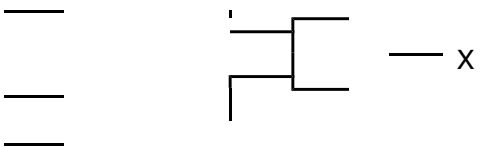
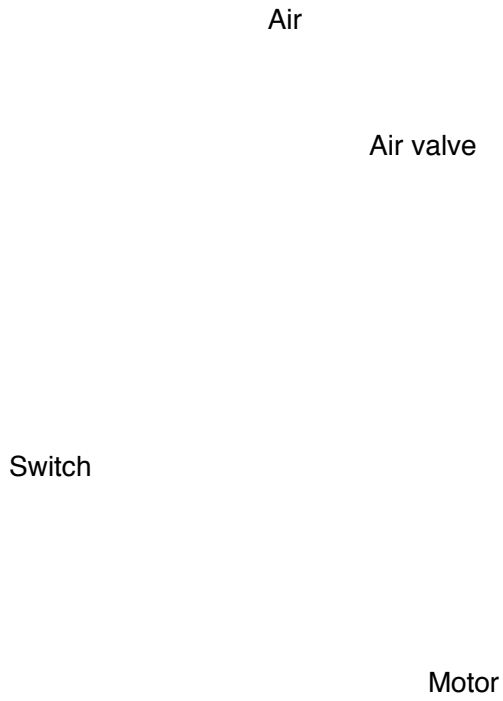


Figure 12.36



Section 12.2

5. You just bought a PLC, and it's still in the box. List the general steps you will have to take to get the PLC operational in a specific task.

6. A PLC has eight inputs and eight outputs as shown in Figure 12.17(a). Draw a wiring diagram for this PLC that will be used as the controller for the situation of Exercise 3.

7. A PLC has eight inputs and eight outputs as shown in Figure 12.17(a). Draw a wiring diagram for this PLC that will be used as the controller for the situation of Exercise 4.

8. List the steps the PLC takes to execute the ladder diagram program.

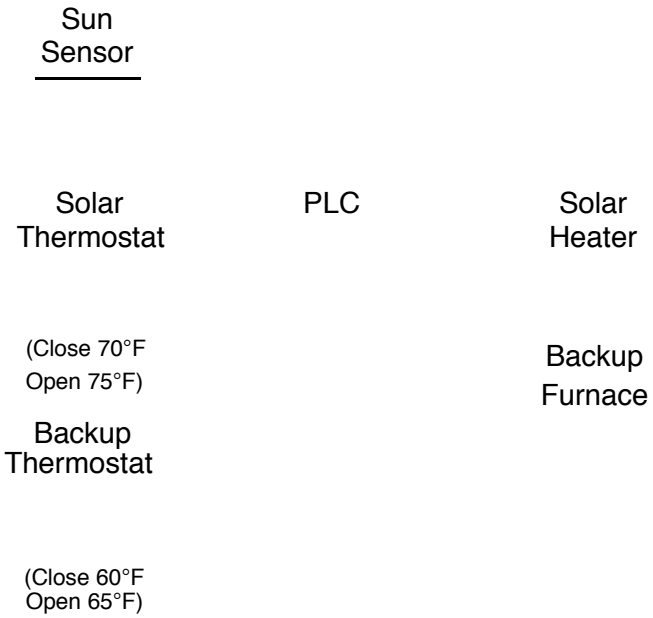
Section 12.3

9. Draw a PLC ladder diagram for the situation described in Exercise 3.

10. A motor is controlled by a NO start button, a NC stop button, and an overload device. The overload device is normally closed and opens if the motor overheats. Draw the PLC ladder diagram for the circuit. Include a light that comes on when the motor overheats.

11. A PLC is to control the solar heating system shown in Figure 12.37. The system has two interrelated parts: (a) A solar thermostat turns on and off the solar heater *if* the sun sensor says the sun is shining, and (b) a backup thermostat turns on and off a conventional furnace *if* the solar energy is insufficent. Both heating systems share the same ductwork, so if the backup thermostat closes, the PLC *must* turn on

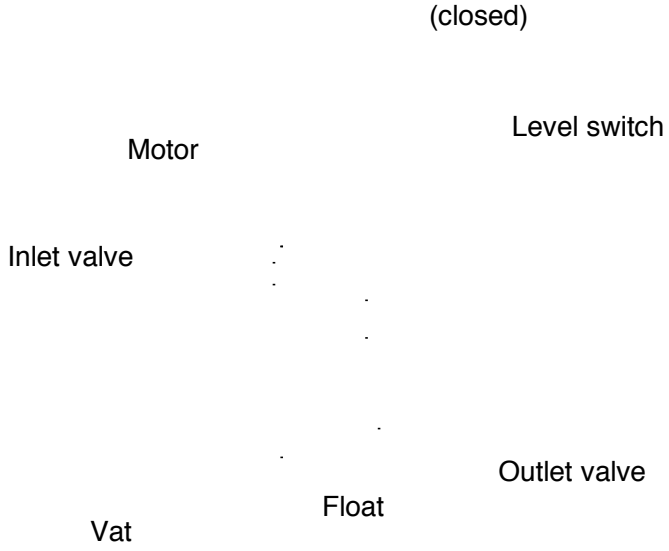
Figure 12.37



the backup furnace (and turn off the solar heater if it's on). Draw a PLC ladder diagram for this system.

12. A mixing vat has an inlet valve, an outlet valve, a mixing motor, and a single level- detecting switch (Figure 12.38). Both valves are opened by solenoids. The level switch closes when the vat is full and stays closed until the vat is empty. Draw a PLC ladder logic diagram to do the following:

Figure 12.38



a. When the start button is pushed, the inlet valve opens until the vat is full.
b. The mixer motor comes on for 5 min.
c. The outlet valve opens until the vat is empty.

13. Modify the two-point control program of Example 12.6 so that the oven temperature limits are 72-76°C. The system should be started and stopped with NO push-button switches (instead of toggle switches).

Section 12.4

14. What are the three levels of networks that might be found in a process-control industry.

15. What is a "device-level" network, and what advantages might this system offer over traditional point-to-point wiring?

Section 12.5

16. How is a "motion controller" different from a PLC?

17. Using the sample program in Section 12.5 as a guide, write a motion-control program for a three-axis system to do the following:
Start at $X = 0$, $Y = 0$, and $Z = 0$.
Move to $X = 5$, $Y = 5$, and $Z = 2$.
Move to $X = 4$, $Y = 4$, and $Z = 3$.
Pause 500 ms.
Move to start positio