chapter 8

Discrete Control Using Programmable Logic Controllers and Personal Computers

CHAPTER CONTENTS

Numerical control (Chapter 6) and industrial robotics (Chapter 7) are primarily concerned with motion control, because the applications of machine tools and robots involve the movement of a cutting tool or end effector, respectively. A more general control category is discrete control, defined in Section 4.2.2. In the present chapter, we provide a more-complete discussion of discrete control, and we examine the two principal industrial controllers used to implement discrete control: (1) programmable logic controllers (PLCs) and (2) personal computers (PCs).

DISCRETE PROCESS CONTROL

Discrete process control systems deal with parameters and variables that change at discrete moments in time. In addition, the parameters and variables themselves are discrete,

Scanned with OKEN Scanner

typically binary. They can have either of two possible values, 1 or 0. The values mean ON or OFF, true or false, object present or not present, high voltage value or low voltage value, and so on, depending on the application. The binary variables in discrete process control are associated with input signals to the controller and output signals from the controller. Input signals are typically generated by binary sensors, such as limit switches or photosensors that are interfaced to the process. Output signals are generated by the controller to operate the process in response to the input signals and as a function of time. These output signals turn on and off switches, motors, valves, and other binary actuators related to the process. We have compiled a list of binary sensors and actuators, along with the interpretation of their 0 and 1 values, in Table 8.1. The purpose of the controller is to coordinate the various actions of the physical system, such as transferring parts into the workholder, feeding the machining workhead, and so on. Discrete process control can be divided into two categories: (1) logic control, which is concerned with event-driven changes in the system; and (2) sequencing, which is concerned with time-driven changes in the system. Both are referred to as *switching systems* in the sense that they switch their output values on and off in response to changes in events or time.

8.1.1 Logic Control

A *logic control* system, also referred to as *combinational logic control*, is a switching system whose output at any moment is determined exclusively by the values of the inputs. A logic control system has no memory and does not consider any previous values of input signals in determining the output signal. Neither does it have any operating characteristics that perform directly as a function of time.

Let us use an example from robotics to illustrate logic control. Suppose that in a machine-loading application, the robot is programmed to pick up a raw workpart from a known stopping point along a conveyor and place it into a forging press. Three conditions must be satisfied to initiate the loading cycle. First, the raw workpart must be at the stopping point; second, the forge press must have completed the process on the previous part; and third, the previous part must be removed from the die. The first condition can be indicated by means of a simple limit switch that senses the presence of the part at the conveyor stop and transmits an ON signal to the robot controller. The second condition can be indicated by the forge press, which sends an ON signal after it has completed the previous cycle. The third condition might be determined by a photodetector located so as to sense the presence or absence of the part in the forging die. When the finished part is removed from the die, an ON signal is transmitted by the photocell. All three of these ON signals must be received by the robot controller to initiate the next work cycle. When these

ABLE 8.1 Binary Sensors and Actuators Used in Discrete Process Control

Sensor	*One/Zero Interpretation*	*Actuator*	*One/Zero Interpretation*
imit switch	Contact/no contact	Motor	On/off
hotodetector	On/off	Control relay	Contact/no contact
ush-button switch	On/off	Light	On/off
mer	On/off	Valve	Closed/open
ontrol relay	Contact/no contact	Clutch	Engaged/not engaged
rcuit breaker	Contact/no contact	Solenoid	Energized/not energized

Scanned with OKEN Scanner

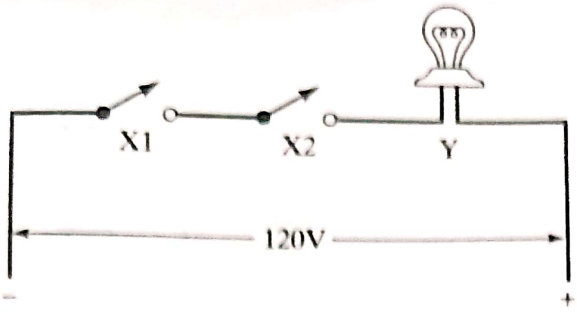


Figure 8.1 Electrical circuit illustrating the operation of the logical AND gate.

input signals have been received by the controller, the robot loading cycle is switched on. No previous conditions or past history are needed.

Elements of Logic Control. The basic elements of logic control are the logic gates AND, OR, and NOT. In each case, the logic gate is designed to provide a specified output value based on the values of the input(s). For both inputs and outputs, the values can be either of two levels, the binary values 0 or 1. For purposes of industrial control, we define 0 (zero) to mean OFF, and 1 (one) to mean ON.

The logical AND gate outputs a value of 1 if all of the inputs are 1, and 0 otherwise. Figure 8.1 illustrates the operation of a logical AND gate. If both switches X1 and X2 (representing inputs) in the circuit are closed, then the lamp Y (representing the output) is on. The truth table is often used to present the operation of logic systems. A *truth table* is a tabulation of all of the combinations of input values to the corresponding logical output values. The truth table for the AND gate is presented in Table 8.2. The AND gate might be used in an automated production system to indicate that two (or more) actions have been successfully completed, therefore signaling that the next step in the process should be initiated. The interlock system in our previous robot forging example illustrates the AND gate. All three conditions must be satisfied before loading of the forge press is allowed to occur.

The logical OR gate outputs a value of 1 if either of the inputs has a value of 1, and 0 otherwise. Figure 8.2 shows how the OR gate operates. In this case, the two input signals X1 and X2 are arranged in a parallel circuit, so that if either switch is closed, the lamp Y will be on. The truth table for the OR gate is presented in Table 8.3. A possible use of the OR gate in a manufacturing system is for safety monitoring. Suppose that two sensors are

TABLE 8.2 Truth Table for the Logical AND Gate

Inputs		*Output*
X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

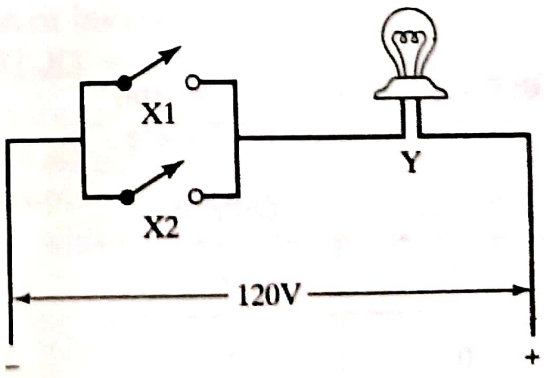


Figure 8.2 Electrical circuit illustrating the operation of the logical OR gate.

Scanned with OKEN Scanner

TABLE 8.3 Truth Table for the Logical OR Gate

Inputs		*Output*
X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

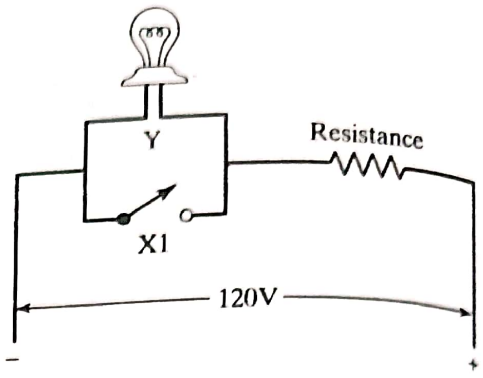


Figure 8.3 Electrical circuit illustrating the operation of the logical NOT gate.

utilized to monitor two different safety hazards. When either hazard is present, the respective sensor emits a positive signal that sounds an alarm buzzer.

Both the AND and OR gates can be used with two or more inputs. The NOT gate has a single input. The logical NOT gate reverses the input signal: If the input is 1, then the output is 0; if the input is 0, then the output is 1. Figure 8.3 shows a circuit in which the input switch X1 is arranged in parallel with the output so that the voltage flows through the lower path when the switch is closed (thus Y = 0), and through the upper path when the switch is open (thus Y = 1). The truth table for the NOT gate is shown in Table 8.4.

In addition to the three basic elements, there are two more elements that can be used in switching circuits: the NAND and NOR gates. The logical NAND gate is formed by combining an AND gate and a NOT gate in sequence, yielding the truth table shown in Table 8.5(a). The logical NOR gate is formed by combining an OR gate followed by a NOT gate, providing the truth table in Table 8.5(b).

Various diagramming techniques have been developed to represent the logic elements and their relationships in a given logic control system. The logic network diagram is one of the most common methods. Symbols used in the logic network diagram are illustrated in Figure 8.4. We demonstrate the use of the logic network diagram in several examples later in this section.

TABLE 8.4 Truth Table for the Logical NOT Gate

Inputs	*Output*
X1	Y
0	1
1	0

TABLE 8.5 Truth Tables for the Logical NAND Gate and Logical NOR Gate

(a) NAND

Inputs		*Output*
X1	X2	Y
0	0	1
0	1	1
1	0	1
1	1	0

(b) NOR

Inputs		*Output*
X1	X2	Y
0	0	1
0	1	0
1	0	0
1	1	0

Scanned with OKEN Scanner

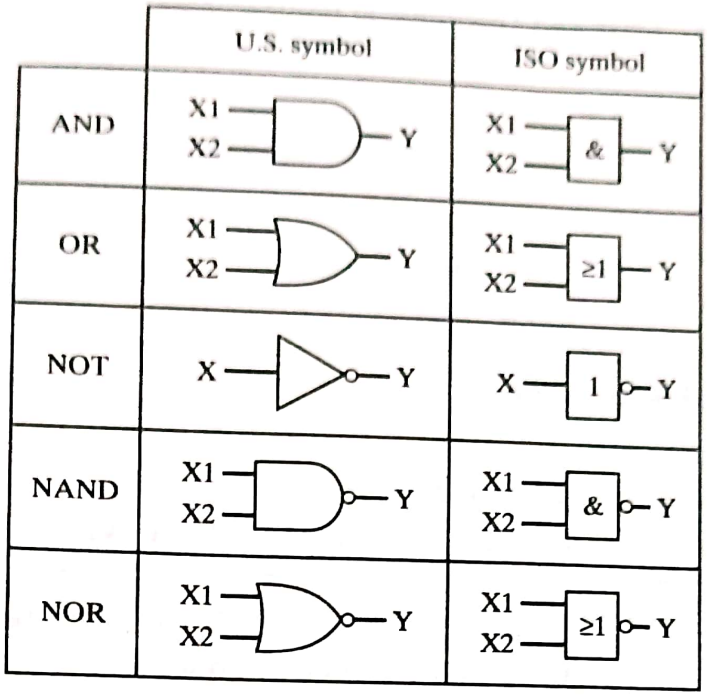


Figure 8.4 Symbols used for logical gates: U.S. and ISO.

Boolean Algebra. The logic elements form the foundation for a special algebra that was developed around 1847 by George Boole and that bears his name. Its original purpose was to provide a symbolic means of testing whether complex statements of logic were TRUE or FALSE. It was not until about a century later that Boolean algebra was shown to be useful in digital logic systems. We briefly describe some of the fundamentals of Boolean algebra here, with minimum elaboration. In Boolean algebra, the AND function is expressed as

$$Y = X1 \cdot X2 \tag{8.1}$$

This is called the logical product of X1 and X2. The results of the AND function for four possible combinations of two input binary variables are listed in the truth table of previous Table 8.2. The OR function in Boolean algebra notation is given by

$$Y = X1 + X2 \tag{8.2}$$

This is called the logical sum of X1 and X2. The output of the OR function for four possible combinations of two input binary variables are listed in the truth table of Table 8.3.

The NOT function is referred to as the negation or inversion of the variable. It is indicated by placing a bar above the variable (e.g., NOT X1 = $\overline{X1}$). The truth table for the NOT function is listed in Table 8.4.

There are certain laws and theorems of Boolean algebra. We cite them in Table 8.6. These laws and theorems can often be applied to simplify logic circuits and reduce the number of elements required to implement the logic, with resulting savings in hardware and/or programming time.

EXAMPLE 8.1 Robot Machine Loading

The robot machine loading example described at the beginning of Section 8.1.1 required three conditions to be satisfied before the loading sequence was

Scanned with OKEN Scanner

TABLE 8.6 Laws and Theorems of Boolean Algebra

Commutative Law:

$X + Y = Y + X$

$X \cdot Y = Y \cdot X$

Associative Law:

$X + Y + Z = X + (Y + Z)$

$X + Y + Z = (X + Y) + Z$

$X \cdot Y \cdot Z = X \cdot (Y \cdot Z)$

$X \cdot Y \cdot Z = (X \cdot Y) \cdot Z$

Distributive Law:

$X \cdot Y + X \cdot Z = X \cdot (Y + Z)$

$(X + Y) \cdot (Z + W) = X \cdot Z + X \cdot W + Y \cdot Z + Y \cdot W$

Law of Absorption:

$X \cdot (X + Y) = X + X \cdot Y = X$

De Morgan's Laws:

$\overline{(X + Y)} = \bar{X} \cdot \bar{Y}$

$\overline{(X + Y)} = \bar{X} + \bar{Y}$

Consistency Theorem:

$X \cdot Y + X \cdot \bar{Y} = X$

$(X + Y) \cdot (X + \bar{Y}) = X$

Inclusion Theorem:

$X \cdot \bar{X} = 0$

$X + \bar{X} = 1$

initiated. Determine the Boolean algebra expression and the logic network diagram for this interlock system.

Solution: Let X1 = whether the raw workpart is present at the conveyor stopping point (X1 = 1 for present, X1 = 0 for not present). Let X2 = whether the press cycle for the previous part has completed (X2 = 1 for completed, 0 for not completed). Let X3 = whether the previous part has been removed from the die (X3 = 1 for removed, X3 = 0 for not removed). Finally, let Y = whether the loading sequence can be started (Y = 1 for begin, Y = 0 for wait).

The Boolean algebra expression is:

$$Y = X1 \cdot X2 \cdot X3$$

All three conditions must be satisfied, so the logical AND function is used. All of the inputs X1, X2, and X3 must have values of 1 before Y = 1, hence initiating the start of the loading sequence. The logic network diagram for this interlock condition is presented in Figure 8.5.

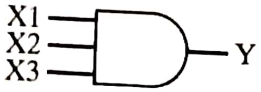


Figure 8.5 Logic network diagram for the robotic machine loading interlock system in Example 8.1.

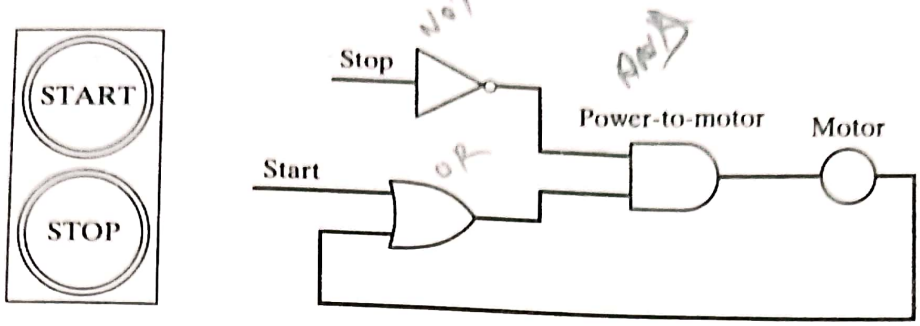
EXAMPLE 8.2 Push-Button Switch

A push-button switch used for starting and stopping electric motors and other powered devices is a common hardware component in an industrial control system. As shown in Figure 8.6(a), it consists of a box with two buttons, one for START and the other for STOP. When the START button is depressed momentarily by a human operator, power is supplied and maintained to the motor (or other load) until the STOP button is pressed. POWER-TO-MOTOR is the output of the push-button switch. The values of the variables can be defined as follows:

START = 0 normally open contact status

START = 1 when the START button is pressed to contact

Scanned with OKEN Scanner



(a) (b)

Figure 8.6 (a) Push-button switch of Example 8.2 and (b) its logic network diagram.

STOP = 0 is normally closed contact status

STOP = 1 when the STOP button is pressed to break contact

MOTOR = 0 when off (not running)

MOTOR = 1 when on

POWER-TO-MOTOR = 0 when the contacts are open

POWER-TO-MOTOR = 1 when the contacts are closed

The truth table for the push-button is presented in Table 8.7. From an initial motor off condition (MOTOR = 0), the motor is started by depressing the start button (START = 1). If the stop button is in its normally closed condition (STOP = 0), power will be supplied to the motor (POWER-TO-MOTOR = 1). While the motor is running (MOTOR = 1), it can be stopped by depressing the stop button (STOP = 1). The corresponding network logic diagram is shown in Figure 8.6(b).

TABLE 8.7 Truth Table for Push-Button Switch of Example 8.2

Start	*Stop*	*Motor*	*Power-to-Motor*
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	0
0	0	1	1
0	1	1	0
1	0	1	1
1	1	1	0

Scanned with OKEN Scanner

In a sense, the push-button switch of Example 8.2 goes slightly beyond our definition of a pure logic system because it exhibits characteristics of memory. The MOTOR and POWER-TO-MOTOR variables are virtually the same signal. The conditions that determine whether power will flow to the motor are different depending on the motor ON/OFF status. Compare the first four lines of the truth table with the last four lines in Table 8.7. It is as if the control logic must remember whether the motor is on or off to decide what conditions will determine the value of the output signal. This memory feature is exhibited by the feedback loop (the lower branch) in the logic network diagram of Figure 8.6(b).

8.1.2 Sequencing

A *sequencing system* uses internal timing devices to determine when to initiate changes in output variables. Washing machines, dryers, dishwashers, and similar appliances use sequencing systems to time the start and stop of cycle elements. There are many industrial applications of sequencing systems. For example, suppose an induction heating coil is used to heat the workpart in our previous example of a robotics forging application. Rather than use a temperature sensor, the heating cycle could be timed so that enough energy is provided to heat the workpart to the desired temperature. The heating process is sufficiently reliable and predictable that a certain duration of time in the induction coil will consistently heat the part to a certain temperature (with minimum variation).

Many applications in industrial automation require the controller to provide a prescheduled set of ON/OFF values for the output variables. The outputs are often generated in an open-loop fashion, meaning that there is no feedback verification that the control function has actually been executed. Another feature that typifies this mode of control is that the sequence of output signals is usually cyclical; the signals occur in the same repeated pattern within each regular cycle. Timers and counters illustrate this type of control component. They are briefly described in Table 8.8.

LADDER LOGIC DIAGRAMS

The logic network diagrams of the type shown in Figures 8.5 and 8.6(b) are useful for displaying the relationships between logic elements. Another diagramming technique that exhibits the logic and, to some extent, the timing and sequencing of the system is the ladder logic diagram. This graphical method also has an important virtue in that it is analogous to the electrical circuits used to accomplish the logic and sequence control. In addition, ladder logic diagrams are familiar to shop personnel who must construct, test, maintain, and repair the discrete control system.

TABLE 8.8 Common Sequencing Elements Used in Discrete Process Control Systems

Timer. This device switches its output on and off at preset time intervals.

Drum timer. A device with multiple on/off outputs, each of which can be independently set with its own time intervals.

Counters. The counter is a component used to count electrical pulses and store the results of the counting procedure. The instantaneous contents can be displayed or used in some control algorithm.

Scanned with OKEN Scanner

In a *ladder logic diagram*, the various logic elements and other components are displayed along horizontal lines or rungs connected on either end to two vertical rails, as illustrated in Figure 8.7. The diagram has the general configuration of a ladder, hence its name. The elements and components are contacts (usually representing logical inputs) and loads (representing outputs). The power (e.g., 120 V ac) to the components is provided by the two vertical rails. It is customary in ladder diagrams to locate the inputs to the left of each rung and the outputs to the right.

Symbols used in ladder diagrams for the common logic and sequencing components are presented in Figure 8.8. Normally open contacts of a switch or other similar device are symbolized by two short vertical lines along a horizontal rung of the ladder, as in Figure 8.8(a). Normally closed contacts are shown as the same vertical lines only with a diagonal line across them as in Figure 8.8(b). Both types of contacts are used to represent ON/OFF inputs to the logic circuit. In addition to switches, inputs include relays, on/off sensors (e.g., limit switches and photodetectors), timers, and other binary contact devices.

Output loads such as motors, lights, alarms, solenoids, and other electrical components that are turned on and off by the logic control system are shown as nodes (circles) as in Figure 8.8(c). Timers and counters are symbolized by squares (or rectangles) with appropriate inputs and outputs to properly drive the device as shown in Figure 8.8(d), (e). The simple timer requires the specification of the time delay and the input signal that activates the delay. When the input signal is received, the timer waits the specified delay time before switching on the output signal. The timer is reset (the output is set back to its initial value) by turning off the input signal.

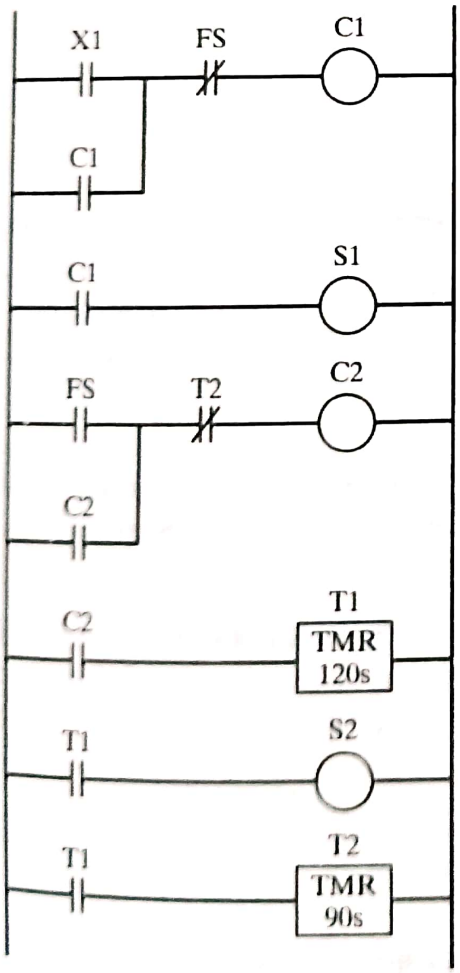


Figure 8.7 A ladder logic diagram.

Ladder symbol	Hardware component
(a)	Normally open contacts (switch, relay, other ON/OFF devices)
(b)	Normally closed contacts (switch, relay, etc.)
(c)	Output loads (motor, lamp, solenoid, alarm, etc.)
(d) TMR 3s	Timer
(e) CTR	Counter

Figure 8.8 Symbols for common logic and sequence elements used in ladder logic diagrams.

Scanned with OKEN Scanner

Counters require two inputs. The first is the pulse train (series of on/off signals) that is counted by the counter. The second is a signal to reset the counter and restart the counting procedure. Resetting the counter means zeroing the count for a count-up device and setting the starting value for a count-down device. The accumulated count is retained in memory for use if required for the application.

EXAMPLE 8.3 Three Simple Lamp Circuits

The three basic logic gates (AND, OR, and NOT) can be symbolized in ladder logic diagrams. Consider the three lamp circuits illustrated in Figures 8.1, 8.2, and 8.3.

Solution: The three ladder diagrams corresponding to these circuits are presented in Figure 8.9(a)-(c). Note the similarity between the original circuit diagrams and the ladder diagrams shown here. Notice that the NOT symbol is the same as a normally closed contact, which is the logical inverse of a normally open contact.

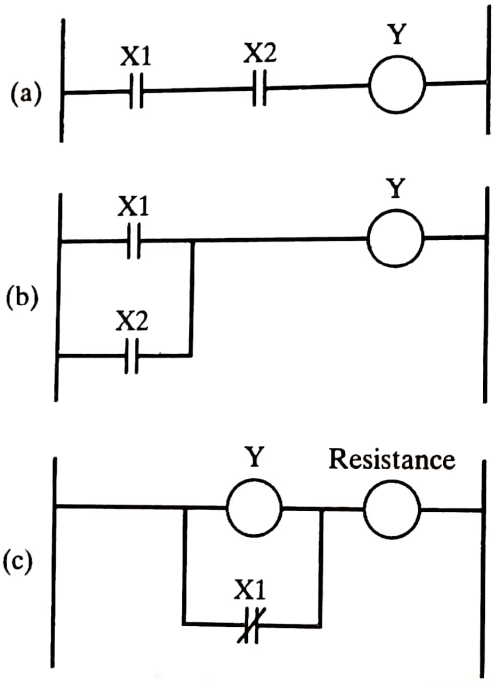


Figure 8.9 Three ladder logic diagrams for lamp circuits in (a) Figure 8.1, (b) Figure 8.2, and (c) Figure 8.3.

EXAMPLE 8.4 Push-Button Switch

The operation of the push-button switch of Example 8.2 can be depicted in a ladder logic diagram. From Figure 8.6, let START be represented by X1, STOP by X2, and MOTOR by Y.

Solution: The ladder diagram is presented in Figure 8.10. X1 and X2 are input contacts and Y is a load in the diagram. Note how Y also serves as an input contact to provide the POWER-TO-MOTOR connection.

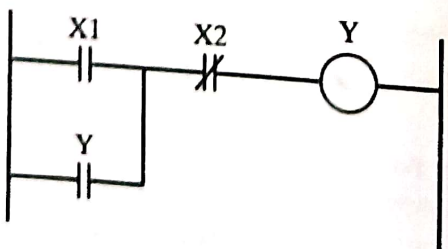


Figure 8.10 Ladder logic diagram for the push-button switch in Example 8.4.

Scanned with OKEN Scanner

EXAMPLE 8.5 Control Relay

The operation of a control relay can be demonstrated by means of the ladder logic diagram presented in Figure 8.11. A relay can be used to control on/off actuation of a powered device at some remote location. It can also be used to define alternative decisions in logic control. Our diagram illustrates both uses. The relay is indicated by the load C (for control relay), which controls the on/off operation of two motors (or other types of output loads) Y1 and Y2. When the control switch X is open, the relay is deenergized, thereby connecting the load Y1 to the power lines. In effect, the open switch X turns on motor Y1 by means of the relay. When the control switch is closed, the relay becomes energized. This opens the normally closed contact of the second rung of the ladder and closes the normally open contact of the third rung. In effect, power is shut off to load Y1 and turned on to load Y2.

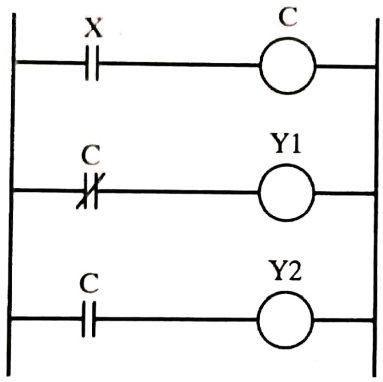


Figure 8.11 Ladder logic diagram for the control relay in Example 8.5.

Example 8.5 illustrates several important features of a ladder logic diagram. First, the same input can be used more than once in the diagram. In our example, the relay contact R was used as an input on both the second and third rungs of the ladder. As we shall see in the following section, this feature of using a given relay contact in several different rungs of the ladder diagram to serve multiple logic functions provides a substantial advantage for the programmable controller over hardwired control units. With hardwired relays, separate contacts would have to be built into the controller for each logic function. A second feature of Example 8.5 is that it is possible for an output (load) on one rung of the diagram to be an input (contact) for another rung. The relay C was the output on the top rung in Figure 8.11, but that output was used as an input elsewhere in the diagram. This same feature was illustrated in the push-button ladder diagram of Example 8.4.

EXAMPLE 8.6

Consider the fluid storage tank illustrated in Figure 8.12. When the start button X1 is depressed, this energizes the control relay C1. In turn, this energizes solenoid S1, which opens a valve allowing fluid to flow into the tank. When the tank becomes full, the float switch FS closes, which opens relay C1, causing the solenoid S1 to be deenergized, thus turning off the in-flow. Switch FS also activates timer T1, which provides a 120-sec delay for a certain chemical reaction to occur in the tank. At the end of the delay time, the timer energizes a second relay C2, which controls two devices: (1) It energizes solenoid S2, which opens a valve to allow the fluid to flow out of the tank; and (2) it initiates timer T2, which waits

Scanned with OKEN Scanner

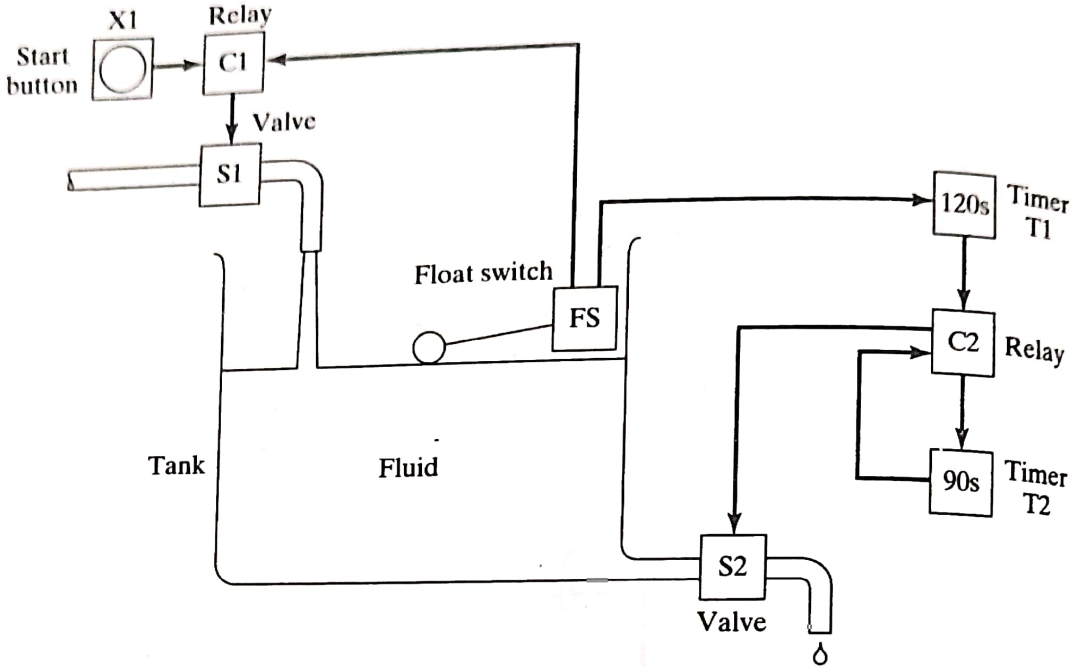


Figure 8.12 Fluid filling operation of Example 8.6.

90 sec to allow the contents of the tank to be drained. At the end of the 90 sec, the timer breaks the current and deenergizes solenoid S2, thus closing the outflow valve. Depressing the start button X1 resets the timers and opens their respective contacts. Construct the ladder logic diagram for the system.

Solution: The ladder logic diagram is constructed as shown in Figure 8.7.

The ladder logic diagram is an excellent way to represent the combinatorial logic control problems in which the output variables are based directly on the values of the inputs. As indicated by Example 8.6, it can also be used to display sequential control (timer) problems, although the diagram is somewhat more difficult to interpret and analyze for this purpose. The ladder diagram is the principal technique for setting up the control programs in PLCs.

PROGRAMMABLE LOGIC CONTROLLERS

A *programmable logic controller* can be defined as a microcomputer-based controller that uses stored instructions in programmable memory to implement logic, sequencing, timing, counting, and arithmetic functions through digital or analog input/output (I/O) modules, for controlling machines and processes. PLC applications are found in both the process industries and discrete manufacturing, but it is primarily associated with the latter industries to control machines, transfer lines, and material handling equipment. Before the PLC was introduced around 1970, hard-wired controllers composed of relays, coils, counters, timers, and similar components were used to implement this type of industrial control (Historical Note 8.1).

Scanned with OKEN Scanner

Historical Note 8.1 Programmable logic controllers [2], [6], [8], [9].

In the mid-1960s, Richard Morley was a partner in Bedford Associates, a New England consulting firm specializing in control systems for machine tool companies. Most of the firm's work involved replacing relays with minicomputers in machine tool controls. In January 1968, Morley devised the notion and wrote the specifications for the first programmable controller.[1] It would overcome some of the limitations of conventional computers used for process control at the time; namely, it would be a real-time processor (Section 4.3.1), it would be predictable and reliable, and it would be modular and rugged. Programming would be based on ladder logic, which was widely used in the industrial controls. The controller that emerged was named the Modicon Model 084. MODICON was an abbreviation of MOdular DIgital CONtroller. Model 084 was derived from the fact that this was the 84th product developed by Bedford Associates. Morley and his associates elected to start up a new company to produce the controllers, and Modicon was incorporated in October 1968. In 1977, Modicon was sold to Gould and became Gould's PLC division.

In the same year that Morley invented the PLC, the Hydramatic Division of General Motors Corporation developed a set of specifications for a PLC. The specifications were motivated by the high cost and lack of flexibility of electromechanical relay-based controllers used extensively in the automotive industry to control transfer lines and other mechanized and automated systems. The requirements included: (1) The device must be programmable and reprogrammable. (2) It must be designed to operate in an industrial environment. (3) It must accept 120 V ac signals from standard push-buttons and limit switches. (4) Its outputs must be designed to switch and continuously operate loads such as motors and relays of 2-A rating. (5) Its price and installation cost must be competitive with relay and solid-state logic devices then in use. In addition to Modicon, several other companies saw a commercial opportunity in the GM specifications and developed various versions of the PLC.

Capabilities of the first PLCs were similar to those of the relay controls they replaced. They were limited to on/off control. Within five years, product enhancements included better operator interfaces, arithmetic capability, data manipulation, and computer communications. Improvements over the next five years included larger memory, analog and positioning control, and remote I/O (permitting remote devices to be connected to a satellite I/O subsystem that was multiplexed to the PLC using twisted pair). Much of the progress was based on advancements taking place in microprocessor technology. By the mid-1980s, the micro PLC had been introduced. This was a down-sized PLC with much lower size (typical size = 75 mm by 75 mm by 125 mm) and cost (less than $500). By the mid-1990s, the nano PLC had arrived, which was still smaller and less expensive.

[1] Morley used the abbreviation PC to refer to the programmable controller. This term was used for many years until IBM began to call their personal computers by the same abbreviation in the early 1980s. The term PLC, widely used today for programmable logic controller, was coined by Allen-Bradley, a leading PLC supplier.

There are significant advantages in using a PLC rather than conventional relays, timers, counters, and other hardware elements. These advantages include: (1) programming the PLC is easier than wiring the relay control panel; (2) the PLC can be reprogrammed, whereas conventional controls must be rewired and are often scrapped instead; (3) PLCs take less floor space than do relay control panels; (4) reliability of the PLC is greater, and maintenance is easier; (5) the PLC can be connected to computer systems more easily than relays; and (6) PLCs can perform a greater variety of control functions than can relay controls.

Scanned with OKEN Scanner

In this section, we describe the components, programming, and operation of the PLC. Although its principal applications are in logic control and sequencing (discrete control), many PLCs also perform additional functions, and we survey some of these at the end of the section.

8.3.1 Components of the PLC

A schematic diagram of a PLC is presented in Figure 8.13. The basic components of the PLC are the following: (1) processor, (2) memory unit, (3) power supply, (4) I/O module, and (5) programming device. These components are housed in a suitable cabinet designed for the industrial environment.

The *processor* is the central processing unit (CPU) of the programmable controller. It executes the various logic and sequencing functions by operating on the PLC inputs to determine the appropriate output signals. The typical CPU operating cycle is described in Section 8.3.2. The CPU consists of one or more microprocessors similar to those used in PCs and other data processing equipment but are designed to facilitate I/O transactions. PLC microprocessors include a range of bit sizes and clock speeds. At the smaller end of the range are 8-bit devices operating at a clock speed of 4 MHz. Medium-sized and larger PLCs use 16- or 32-bit microprocessors running at 33 MHz or faster.

Connected to the CPU is the PLC *memory unit*, which contains the programs of logic, sequencing, and I/O operations. It also holds data files associated with these programs, including I/O status bits, counter and timer constants, and other variable and parameter values. This memory unit is referred to as the user or application memory because its contents are entered by the user. In addition, the processor also has a *system memory* that directs the execution of the control program and coordinates I/O operations. The contents of the system memory are entered by the PLC manufacturer and cannot be accessed or altered by the user. Typical PLC memory capacities range from less than 1K (1000) words for small controllers to more than 64K.

A *power supply* of 120 V alternating current (ac) is typically used to drive the PLC (some units operate on 240 V ac). The power supply converts the 120 V ac into direct current (dc) voltages of ±5 V. These low voltages are used to operate equipment that may

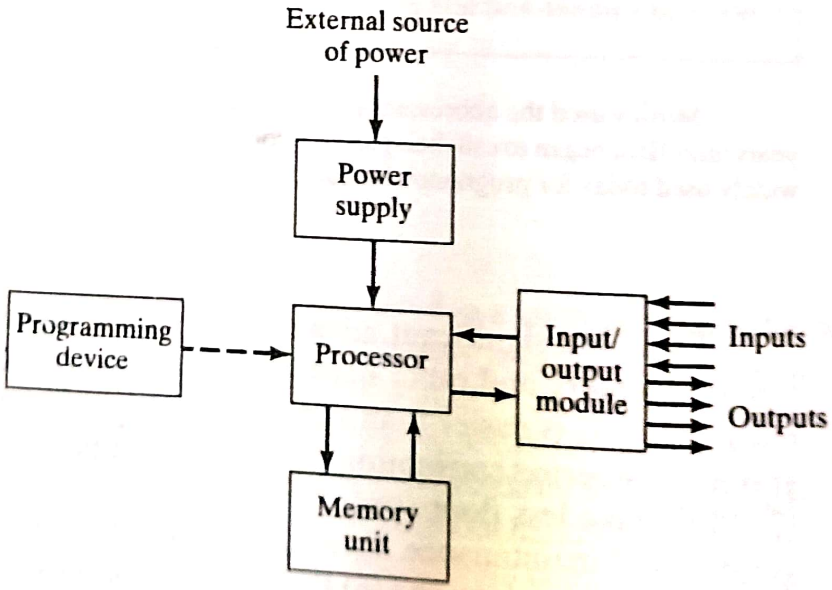


Figure 8.13 Components of a PLC.

Scanned with OKEN Scanner

TABLE 8.9 Typical Classification of PLCs by Number of Input/Output Terminals

PLC Size	*I/O Count*
Large PLC	≥ 1024
Medium PLC	< 1024
Small PLC	< 256
Micro PLC	≤ 32
Nano PLC	≤ 16

have much higher voltage and power ratings than the PLC itself. The power supply often includes a battery backup that switches in automatically in the event of an external power source failure.

The *input/output module* provides the connections to the industrial equipment or process that is to be controlled. Inputs to the controller are signals from limit switches, push-buttons, sensors, and other on/off devices. Outputs from the controller are on/off signals to operate motors, valves, and other devices required to actuate the process. In addition, many PLCs are capable of accepting continuous signals from analog sensors and generating signals suitable for analog actuators. The size of a PLC is usually rated in terms of the number of its I/O terminals, as indicated in Table 8.9.

The PLC is programmed by means of a programming device. The *programming device* is usually detachable from the PLC cabinet so that it can be shared among different controllers. Different PLC manufacturers provide different devices, ranging from simple teach pendant type devices, similar to those used in robotics, to special PLC programming keyboards and CRT displays. Personal computers can also be used to program PLCs. A PC used for this purpose sometimes remains connected to the PLC to serve a process monitoring or supervision function and for conventional data processing applications related to the process.

8.3.2 PLC Operating Cycle

As far as the PLC user is concerned, the steps in the control program are executed simultaneously and continuously. In truth, a certain amount of time is required for the PLC processor to execute the user program during one cycle of operation. The typical operating cycle of the PLC, called a *scan*, consists of three parts: (1) input scan, (2) program scan, and (3) output scan. During the *input scan*, the inputs to the PLC are read by the processor and the status of these inputs is stored in memory. Next, the control program is executed during the *program scan*. The input values stored in memory are used in the control logic calculations to determine the values of the outputs. Finally, during the *output scan*, the outputs are updated to agree with the calculated values. The time to perform the scan is called the *scan time*, and this time depends on the number of inputs that must be read, the complexity of control functions to be performed, and the number of outputs that must be changed. Scan time also depends on the clock speed of the processor. Scan times typically vary between 1 and 25 msec [5].

One of the potential problems that can occur during the scan cycle is that the value of an input can change immediately after it has been sampled. Since the program uses the

Scanned with OKEN Scanner

input value stored in memory, any output values that are dependent on that input are determined incorrectly. There is obviously a potential risk involved in this mode of operation. However, the risk is minimized because the time between updates is so short that it is unlikely that the output value being incorrect for such a short time duration will have a serious effect on process operation. The risk becomes most significant in processes in which the response times are very fast and where hazards can occur during the scan time. Some PLCs have special features for making "immediate" updates of output signals when input variables are known to cycle back and forth at frequencies faster than the scan time.

8.3.3 Additional Capabilities of a PLC

The logic control and sequencing functions described in Section 8.1 are likely to be the principal control operations accomplished by the PLC. These are the functions for which the programmable controller was originally designed. However, the PLC has evolved to include several capabilities in addition to logic control and sequencing. Some of these additional capabilities available on many commercial PLCs include:

- *Analog control.* Proportional-integral-derivative (PID) control is available on some programmable controllers. These control algorithms have traditionally been implemented on analog controllers. Today the analog control schemes are approximated using the digital computer, with either a PLC or a computer process controller.
- *Arithmetic functions.* These functions are addition, subtraction, multiplication, and division. Use of these functions permits more-complex control algorithms to be developed than what is possible with conventional logic and sequencing elements.
- *Matrix functions.* Some PLCs have the capability to perform matrix operations on stored values in memory. The capability can be used to compare the actual values of a set of inputs and outputs with the values stored in the PLC memory to determine if some error has occurred.
- *Data processing and reporting.* These functions are typically associated with business applications of PCs. PLC manufacturers have found it necessary to include these PC capabilities in their controller products. The distinction between PCs and PLCs is becoming less and less clear.

8.3.4 Programming the PLC

Programming is the means by which the user enters the control instructions to the PLC through the programming device. The most basic control instructions consist of switching logic, sequencing, counting, and timing. Virtually all PLC programming methods provide instruction sets that include these functions. Many control applications require additional instructions to accomplish analog control of continuous processes, complex control logic, data processing and reporting, and other advanced functions not readily performed by the basic instruction set. Owing to these differences in requirements, a variety of PLC programming languages have been developed. A standard for PLC programming was published by the International Electrotechnical Commission and released in 1992 entitled *International Standard for Programmable Controllers* (IEC 1131–3). This standard specifies three graphical languages and two text-based languages for programming PLCs, respectively: (1) ladder logic diagrams, (2) function block diagrams, (3) sequential functions

Scanned with OKEN Scanner

TABLE 8.10 Features of the Five PLC Languages Specified in the IEC 1131–3 Standard

Language	*Abbreviation*	*Type*	*Applications Best Suited for*
Ladder logic diagram	(LD)	Graphical	Discrete control
Function block diagram	(FBD)	Graphical	Continuous control
Sequential function chart	(SFC)	Graphical	Sequencing
Instruction list	(IL)	Textual	Similar to ladder diagrams
Structured text	(ST)	Textual	Complex logic, computations, etc.

charts, (4) instruction list, and (5) structured text. Table 8.10 lists the five languages along with the most suitable application of each. IEC 1131–3 also states that the five languages must be able to interact with each other to allow for all possible levels of control sophistication in any given application.

Ladder Logic Diagram. The most widely used PLC programming language today involves ladder diagrams (LDs), examples of which are shown in several previous figures. As indicated in Section 8.2, ladder diagrams are very convenient for shop personnel who are familiar with ladder and circuit diagrams but may not be familiar with computers and computer programming. The use of ladder logic diagrams does not require them to learn an entirely new programming language.

Direct entry of the *ladder logic diagram* into the PLC memory requires the use of a keyboard and CRT with graphics capability to display symbols representing the components and their interrelationships in the ladder logic diagram. The symbols are similar to those presented in Figure 8.8. The PLC keyboard device is often designed with keys for each of the individual symbols. Programming is accomplished by inserting the appropriate components into the rungs of the ladder diagram. The components are of two basic types: contacts and coils. *Contacts* are used to represent input switches, relay contacts, and similar elements. *Coils* are used to represent loads such as motors, solenoids, relays, timers, and counters. In effect, the programmer inputs the ladder logic circuit diagram rung by rung into the PLC memory with the CRT displaying the results for verification.

Function Block Diagrams. The *function block diagram* (FBD) provides a means of inputting high-level instructions. Instructions are composed of operational blocks. Each block has one or more inputs and one or more outputs. Within a block, certain operations take place on the inputs to transform the signals into the desired outputs. The function blocks include operations such as timers and counters, control computations using equations (e.g., proportional-integral-derivative control), data manipulation, and data transfer to other computer-based systems. We leave further description of these function blocks to other references, such as [5] and the operating manuals for commercially available PLC products.

Sequential Function Charts. The *sequential function chart* (SFC, also called the *Grafcet* method) graphically displays the sequential functions of an automated system as a series of steps and transitions from one state of the system to the next. The sequential function chart is described in Boucher [1]. It has become a standard method for documenting logic control and sequencing in much of Europe. However, its use in the United States is more limited, and we refer the reader to the cited reference for more details on the method.

Scanned with OKEN Scanner

Instruction List. Instruction list (IL) programming also provides a way of entering the ladder logic diagram into PLC memory. In this method, a low-level computer language is employed by the programmer to construct the ladder logic diagram by entering statements that specify the various components and their relationships for each rung of the ladder diagram. Let us explain this approach by introducing a hypothetical PLC instruction set. Our PLC "language" is a composite of various manufacturers' languages containing fewer features than most commercially available PLCs. We assume that the programming device consists of a suitable keyboard for entering the individual components on each rung of the ladder logic diagram. A CRT capable of displaying each ladder rung (and perhaps several rungs that precede it) is useful to verify the program. The instruction set for our PLC is presented in Table 8.11 with a concise explanation of each instruction. Let us examine the use of these commands with several examples.

EXAMPLE 8.7 Language Commands for AND, OR, and NOT Circuits

Using the command set in Table 8.11, write the PLC programs for the three ladder diagrams from Figure 8.10, depicting the AND, OR, and NOT circuits from Figures 8.1, 8.2, and 8.3.

Solution: Commands for the three circuits are listed below, with explanatory comments.

	Command	Comment
(a)	STR X1	Store input X1
	AND X2	Input X2 in series with X1
	OUT Y	Output Y
(b)	STR X1	Store input X1
	OR X2	Input X2 parallel with X1
	OUT Y	Output Y
(c)	STR NOT X1	Store inverse of X1
	OUT Y	Output Y

TABLE 8.11 Typical Low-Level Language Instruction Set for a PLC

STR	Store a new input and start a new rung of the ladder.
AND	Logical AND referenced with the previously entered element. This is interpreted as a series circuit relative to the previously entered element.
OR	Logical OR referenced with the previously entered element. This is interpreted as a parallel circuit relative to the previously entered element.
NOT	Logical NOT or inverse of entered element.
OUT	Output element for the rung of the ladder diagram.
TMR	Timer element. Requires one input signal to initiate timing sequence. Output is delayed relative to input by a duration specified by the programmer in seconds. Resetting the timer is accomplished by interrupting (stopping) the input signal.
CTR	Counter element. Requires two inputs: One is the incoming pulse train that is counted by the CTR element, the other is the reset signal indicating a restart of the counting procedure.

Scanned with OKEN Scanner

EXAMPLE 8.8 Language Commands for Control Relay

Using the command set in Table 8.11, write the PLC program for the control relay depicted in the ladder logic diagram of Figure 8.11.

Solution: Commands for the three circuits are listed below, with explanatory comments.

Command	Comment
STR X	Store input X
OUT C	Output contact relay C
STR NOT C	Store inverse of C output
OUT Y1	Output load Y1
STR C	Store C output
OUT Y2	Output load Y2

The low-level languages are generally limited to the kinds of logic and sequencing functions that can be defined in a ladder logic diagram. Although timers and counters have not been illustrated in the two preceding examples, some of the exercise problems at the end of the chapter require the reader to make use of them.

Structured Text. *Structured text* (ST) is a high-level computer-type language likely to become more common in the future to program PLCs and PCs for automation and control applications. The principal advantage of a high-level language is its capability to perform data processing and calculations on values other than binary. Ladder diagrams and low-level PLC languages are usually quite limited in their ability to operate on signals that are other than on/off types. The capability to perform data processing and computation permits the use of more-complex control algorithms, communications with other computer-based systems, display of data on a CRT console, and input of data by a human operator. Another advantage is the relative ease with which a complicated control program can be interpreted by a user. Explanatory comments can be inserted into the program to facilitate interpretation.

4 PERSONAL COMPUTERS USING SOFT LOGIC

In the early 1990s, PCs began to encroach into applications formerly dominated by PLCs. Previously, PLCs were always seen to have the advantage of being designed for the harsh environment of the factory, while PCs were designed for the office environment. In addition, with its built-in I/O interface, the PLC could be readily connected to external equipment, whereas the PC required special I/O cards to enable such connections. These advantages notwithstanding, the technological evolution of PLCs has not kept pace with the development of PCs, new generations of which are introduced with much greater frequency than PLCs are. There is much more proprietary software and architecture in PLCs than in PCs. Over time, this has resulted in a performance disadvantage for PLCs. At time of writing, PLC performance lags its PC counterpart by as much as two years, and the gap is increasing. PC speeds are typically doubling every 18 months or so, accordingly to Moore's Law (Section 4.4.6 and Table 4.5), much more rapidly than in PLC technology, which

Scanned with OKEN Scanner

requires that individual companies redesign their proprietary software and architectures for each new generation of microprocessors.

PCs are now available in more-sturdy enclosures for use in the plant. They can be equipped with membrane-type keyboards for protection against factory moisture, oil, and dirt. They can be ordered with I/O cards and related hardware to provide the necessary devices to connect to the plant's equipment and processes. They come with Windows NT or other operating system designed for implementing control applications in addition to traditional office software. And they can be programmed with *soft logic*, a term used to describe a family of software products that emulate the operations of the built-in control software used in PLCs. PLC makers are responding to the PC challenge by including PC components and features in their controller products, calling them *softPLC* or similar names to distinguish them from conventional PLCs. Nevertheless, the future is likely to see PCs used in increasing numbers in factory control applications where PLCs would have formerly been used.

An example of the soft logic products is *FloPro*[2], a software package for PCs that uses a flowchart-based language rather than ladder logic diagrams. The argument for flowchart programming is that most computer software is developed using flowcharts. Before writing any computer code, the programmer develops a logical and sequential plan using flowcharts that detail what decisions and actions the software is to accomplish. With FloPro, the flowchart is entered directly into the computer. In fact, the FloPro programming tools allow the flowchart to be developed on the computer rather than manually beforehand. The control program can be written, debugged, and simulated on a conventional office PC before being loaded into the industrial control computer for the given application.

FloPro permits the development of multiple flowcharts, each designed to accomplish a relatively simple control task. The flowcharts execute their respective tasks simultaneously and utilize a common database, so that a change in a parameter value is available to all of the flowcharts that use that parameter. The typical program, consisting of many separate flowcharts, executes in a very short scan cycle, typically less than 10 msec for several hundred I/O points and less than 85 msec for 11,000 I/O points [4]. This makes the execution of a FloPro control program on a modern PC comparable to or faster than executing the same control functions on a PLC.

There are three basic types of graphical symbols in a FloPro flowchart, as illustrated in Figure 8.14: (a) enable criteria, (b) action blocks, and (c) test blocks. The *enable criteria* is used to determine whether a given flowchart in the larger flowchart program is permit-

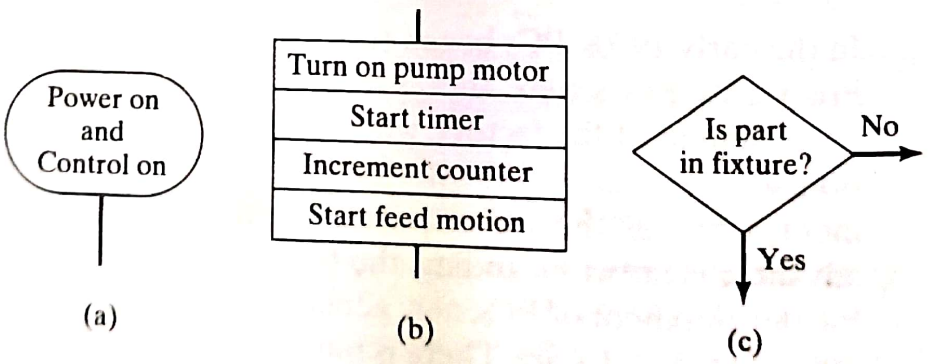


Figure 8.14 Three types of flowchart symbols in FloPro: (a) enable criteria, (b) action blocks, and (c) test blocks.

[2] FloPro is a software product of NemaSoft, a division of Nematron Corporation in Ann Arbor, Michigan.

Scanned with OKEN Scanner