Programmable Controllers
4th Edition

Thomas A. Hughes

Printed in the United States of America.
10 9 8 7 6 5 4 3

ISBN 1-55617-899-9

ISA
67 Alexander Drive
P.O. Box 12277
Research Triangle Park, NC 27709

Library of Congress Cataloging-in-Publication Data

Hughes, Thomas A.
Programmable controllers / Thomas A. Hughes.-- 4th ed.
p. cm.
ISBN 1-55617-899-9 (hardcover)
1. Programmable controllers. I. Title.
TJ223.P76H84 2004
629.8'9--dc22
2004024467

DEDICATION

This book is dedicated to my grandson, Ian Lovell Rager.

About the Author

Thomas A. Hughes, a Senior Member of ISA has 34 years of experience in the design and installation of instrumentation and control systems, including 20 years in the management of instrumentation and control projects for the process and nuclear industries. He is the author of two books: *Measurement and Control Basics, 3rd Edition, (2002)* and *Programmable Controllers, 4th Edition, (2005),* both published by ISA.

Mr. Hughes received a B. S. in engineering physics from the University of Colorado, and a M.S. in control systems engineering from Colorado State University. He holds professional engineering licenses in the states of Colorado and Alaska, and has held engineering and management positions with Dow Chemical, Rockwell International, EG&G Rocky Flats, Topro Systems Integration, and the International Atomic Energy Agency. Mr. Hughes has taught numerous courses in electronics, mathematics, and instrumentation systems at the college level and in industry. He is currently the Principal Consultant with Nova Systems Engineering Services in Arvada, Colorado.

Preface

Since 1989, this book has been used both as a textbook for programmable logic controller (PLC) courses and for self-study by thousands of professionals. This applications-based book provides a clear and concise presentation of the fundamental principles of programmable controllers for process and machine control. This fourth edition covers all phases of programmable controller applications from design and programming to installation, maintenance, and start-up. Coverage of all five standard PLC programming languages: Ladder Diagram, Function Block Diagram, Sequential Function Chart, Instruction List, and Structured Text has been increased in this fourth edition and numerous programming applications and examples have been added to more clearly explain each programming language.

The text provides a complete and comprehensive presentation on the design and programming of programmable controller–based control systems. The material also includes chapters on binary logic fundamentals, electrical and electronic principles, input and output systems, memory and addressing, programming languages, and data communication. Chapter 11 provides complete coverage of several typical PLC control applications. The final chapter covers design, installation, and maintenance of programmable controllers in detail.

All the chapters have been supplemented with new or improved example problems and exercises. Most of the illustrations in the book have been revised and improved. Answers to all the exercises have been added at the end of the book to assist students and instructors.

I would like to express my appreciation to my wife Ellen for the long hours spent reviewing all four editions. I would also like to thank the technical reviewers for making numerous constructive comments that improved the overall presentation of this fourth edition.

Contents

1 Introduction to Programmable Controllers

Introduction

Programmable controllers were originally designed to replace relay-based control systems and solid-state, hard-wired logic control panels. However, the modern programmable controllers system is far more complex and powerful.

The most basic function performed by programmable controllers is to examine the status of inputs and in response, control some process or machine through outputs. The logical combinations of inputs to produce an output or outputs are called *control logic*. Several logic combinations are usually required to carry out a control plan or program. This control plan is stored in memory using a programming device to input the program into the system. The processor, usually a high-speed microprocessor, in a predetermined sequential order, periodically scans the control plan in memory. The period required to examine the inputs and outputs, perform the control logic, and execute the outputs is called the "scan time."

A simplified block diagram of a programmable controller is shown in Figure 1-1. In this diagram, a level switch and panel-mounted pushbutton are wired to input circuits, and the output circuits are connected to an electric solenoid valve and a panel-mounted indicator light. The output devices are controlled by the control program in the logic unit.

This figure shows a typical configuration of the early programmable controller applications, which were intended to replace relay or hard-wired logic control systems. The input circuits are used to convert the various field voltages and currents to the low voltage signals [normally 0- to 5-V direct current (DC)] used by the logic unit. The output circuits

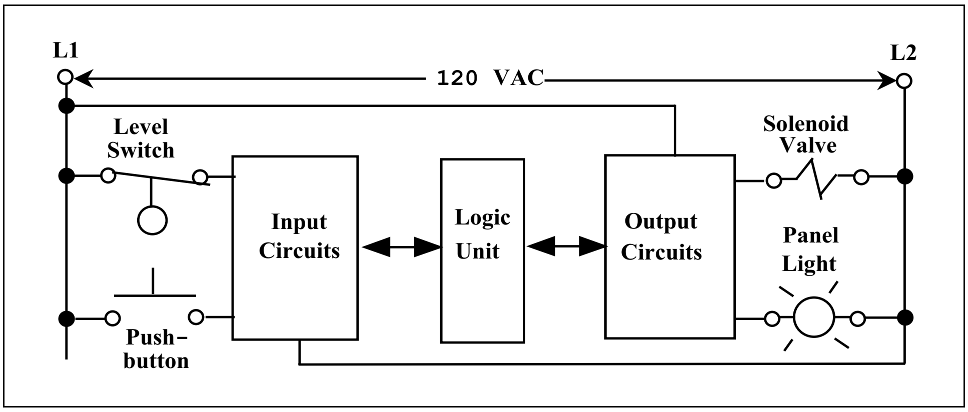


Figure 1-1. Simplified Diagram for a Programmable Controller System

convert the logic signals to a level that will drive the field devices. For example, in Figure 1-1, 120-VAC power is connected to the field input devices, so the input circuits are used to convert the 120 VAC to the 0- to 5-V logic signals used by the control unit.

Brief History of PLCs

In 1968, a major automobile manufacturer wrote a design specification for the first programmable controller. The primary goal was to eliminate the high cost associated with the frequent replacement of inflexible relay-based control systems. The specification also called for a solid-state industrial computer that could be easily programmed by maintenance technicians and plant engineers. It was hoped that the programmable controller would reduce production downtime and provide expandability for future production improvements and changes. In response to this design specification, several manufacturers developed computer-based control devices called *programmable controllers*.

The first programmable controller was installed in 1969, and it proved to be a vast improvement over relay-based control systems. They were easy to install and program, they used less plant floor space, and were more reliable than a relay-based control system. The initial programmable controller not only met the automobile manufacturer production needs, but further design improvements in later models led to widespread use of programmable controllers in other industries.

There were two main factors in the initial design of programmable controllers that led to their success. First, highly reliable solid-state components were used, and the electronic circuits were designed for the harsh industrial environment. The I/O circuits were designed and built to withstand electrical noise, moisture, oil, and the high temperature encountered in industry. The second important factor was that the initial programming language selected was based on standard electrical ladder logic design. Some earlier computer system applications had failed because plant technicians and engineers were not easily trained in standard computer software. However, most were already trained in relay ladder logic design so that programming in a language based on the familiar relay ladder diagrams was learned quickly.

When microprocessors were introduced in 1974 and 1975, the basic capabilities of programmable controllers were greatly expanded and improved. They were able to perform sophisticated math and data manipulation functions, which greatly increased the use of programmable controllers in more complex control applications.

In the late 1970s, improved communication components and circuits made it possible to place programmable controllers thousands of feet from the equipment they controlled, and several programmable controllers can now exchange data to more effectively control processes and machines. In addition, microprocessor-based input and output modules allowed programmable controller systems to evolve into the analog control world.

The introduction of the personal computer (PC) in the early 1980s greatly increased the power and utility of the programmable controller system in process and machine control. The low cost of PCs led to their extensive use as programming devices and operator interface control stations. The development of low-cost graphical control software packages on PCs led to the extensive use of PC-based human machine interfaces (HMIs) in programmable controller applications in the early 1990s.

The worldwide standardization of the very reliable and powerful Microsoft operating software systems had a dramatic impact on PLC programming, control software, and data collection during the mid 1990s. The PLC vendors and developers started to use Microsoft developed software as the basis for their programming and HMI software packages that could operate on inexpensive PCs.

In particular, a Windows copy-and-paste function called Object Linking and Embedding, or OLE, found wide use in PLC software applications and led to the establishment of the process control industry standard OLE for Process Control (OPC). OPC allows for fast and secure access to process data and information under Windows operating systems. Finally, the widespread use of the Internet and advanced communication systems and techniques has led to very powerful PLC systems.

Programmable controllers are found in thousands of industrial applications. They are used to control chemical, petrochemical, food, and pharmaceutical, wastewater treatment, water treatment, nuclear, natural gas, and mining processes. They are found in material transfer and storage systems that transport and store both the raw materials and the finished products. They are used with robots to perform hazardous industrial operations to allow for safer operations. Programmable controllers are used in conjunction with other computers to perform process and machine data collection and reporting functions, including statistical process control, quality assurance, and online diagnostics. They are utilized in energy management systems to reduce costs and improve environmental control of industrial facilities and office buildings.

Because of the wide use of the PC in control and business applications, the acronym PC is generally reserved for personal computers and the abbreviation PLCs is used for programmable controllers or programmable logic controllers. The abbreviation PLCs will be used in this book to represent programmable controllers.

Basic Components of Programmable Controllers

Regardless of size, cost or complexity, all PLCs share the same basic components and functional characteristics. A programmable controller will typically consist of the following hardware components: a processor, an input/output system, memory, a power supply and a communications device or port. A block diagram of typical PLC hardware components is shown in Figure 1-2.

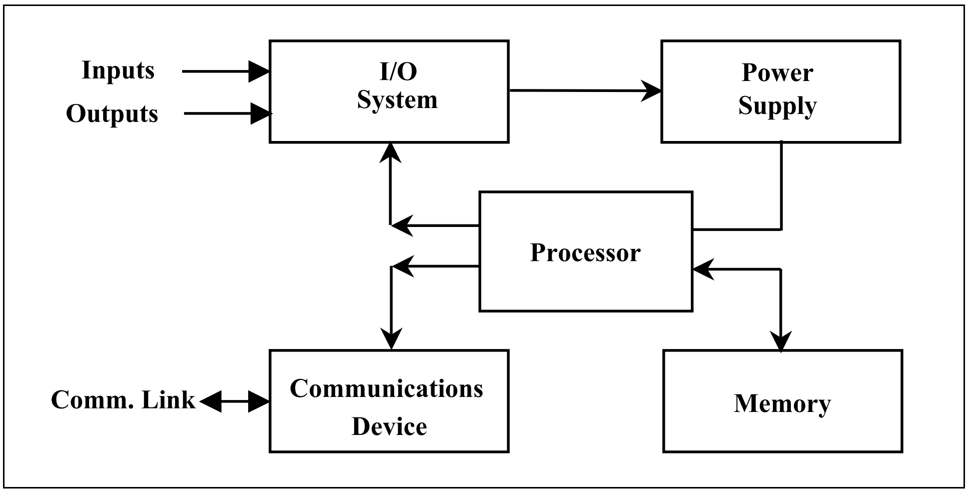


Figure 1-2. Block Diagram of a Typical PLC

Programmable controllers also require programming devices and software that will be discussed later. First, we will cover the basic PLC hardware components.

The Processor

The processor consists of one or more standard or custom microprocessors and other integrated circuits that perform the logic, control, and memory functions of the PLC system. The processor reads the inputs, executes logic as determined by the application program, performs calculations, and controls the outputs accordingly.

The processor controls the operating cycle or processor scan. This operating cycle consists of a series of operations performed sequentially and repeatedly. A typical PLC processor operating cycle is shown in Figure 1-3.

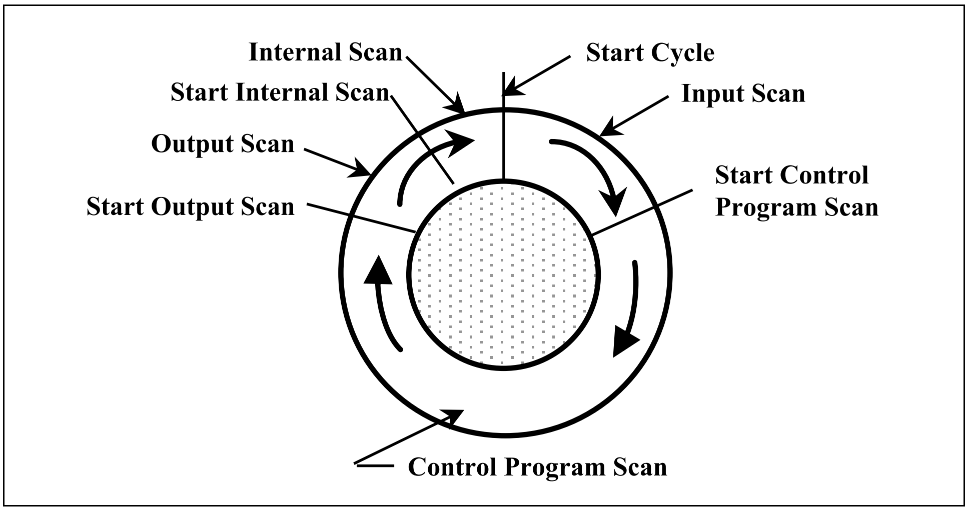


Figure 1-3. PLC Processor Operating Cycle

During the "input scan," the PLC examines the external input devices for a signal present or absent (i.e., an on or off state). The status of these inputs is temporarily stored in an input image table or memory file.

During the "program scan" cycle, the processor scans the instructions in the control program, uses the input status from the input image file, and determines if an output will or will not be energized. The resulting status of the outputs is written to the output image table or memory file.

Based on the data in the output image table, the PLC energizes or deenergizes its associated output circuits, controlling external devices during the “Output Scan” cycle.

During the “Internal Scan” cycle, the processor performs housekeeping functions such as internal diagnostics and communications. A typical diagnostic function would be to check for math operational errors. The PLC must also check for and perform communication operations with its programming device, other PLCs, or other devices connected to its communication port or ports.

This operating cycle typically takes 1 to 25 milliseconds (thousandths of a second). However, the operating cycle time depends on the complexity of the control logic written by the user so that a simple control program might take only 1 ms but a large and complex program may require a scan time as high as 250 ms. The input, output, and internal scans are normally very short compared to the time taken for the program scan. These scans are continually repeated in a looped process.

The PLC operating shown in Figure 1-3 is typical of most PLCs but the manufacturer’s instruction manual should be consulted for the PLC type used in a given control application to avoid programming errors.

I/O System

The I/O system provides the physical connection between the process equipment and the microprocessor. This system uses various input circuits or modules to sense and measure physical quantities of the process, such as motion, level, temperature, pressure, flow, and position. Based on the status sensed or values measured, the processor controls various output modules to drive field devices such as valves, motors, pumps, and alarms to exercise control over a machine or a process.

Input Types

The inputs from field instruments or sensors supply the data and information the processor needs to make logical decisions to control a given process or machine. These input signals from varied devices such as push buttons, hand switches, thermocouples, strain gages, etc., are connected to input modules to filter and condition the signal for use by the processor.

Output Types

The outputs from the PLC energize or deenergize control devices to regulate processes or machines. These output signals are control voltages from the output circuits, and they are generally not high power signals.

For example, an output module sends a control signal that energizes the coil in a motor starter. The energized coil closes the power contacts of the starter. These contacts then close to start the motor. The output modules are usually not directly connected to the power circuit but rather to devices such as motor starter and heater contactors that apply high power (greater than 10 amps) signals to the final control devices.

I/O Structure

PLCs are classified as micro, small, medium, and large mainly based on the I/O count. Micro PLCs generally have an I/O count of 32 or less, small PLCs have less than 256 I/O points, medium-size PLCs have an I/O count less than 1024, and large PLCs have an I/O count greater than 1024. Micro PLCs are self-contained units with the processor, power supply, and I/O all in one package. Because they are self-contained, micro PLCs are also called packaged controllers. A modular PLC is one that has separate components or modules.

The advantage of a packaged controller is that the unit is smaller, costs less, and is easy to install. A typical wiring diagram for a Micro PLC is shown in Figure 1-4. An Allen-Bradley Micro-1000 PLC with nine inputs and five outputs is shown. The unit is powered with 120 V of alternating current (AC) with an internal power supply to operate the internal I/O circuits and the built-in microprocessor, and to generate 24-V direct current (DC) for the field input switches and contacts.

In medium and large PLC systems, the I/O modules are normally installed or plugged into a slot in a "universal" modular housing. The term universal in this context means that any module can be inserted into any I/O slot in the housing. Some systems have special positions in the modular system for the communication modules, power supplies or processor, but the I/O modules can be placed in any position. Some "universal" modular I/O housings are designed so that the I/O modules can be removed without turning off the AC power or removing the field wiring. However, the PLC manufacturer's instruction manual must be consulted before removing I/O module under power conditions.

Figure 1-5 shows some typical configurations for I/O modular housings. The backplane of the housings into which the modules are plugged have a printed circuit card that contains the communications bus to the processor and the DC voltages to operate the digital and analog circuits in the I/O modules.

These I/O housings can be mounted in a control panel or on a subpanel in an enclosure. The housings are designed to provide some protection for the I/O module circuits from dirt, dust, electrical noise, and mechanical

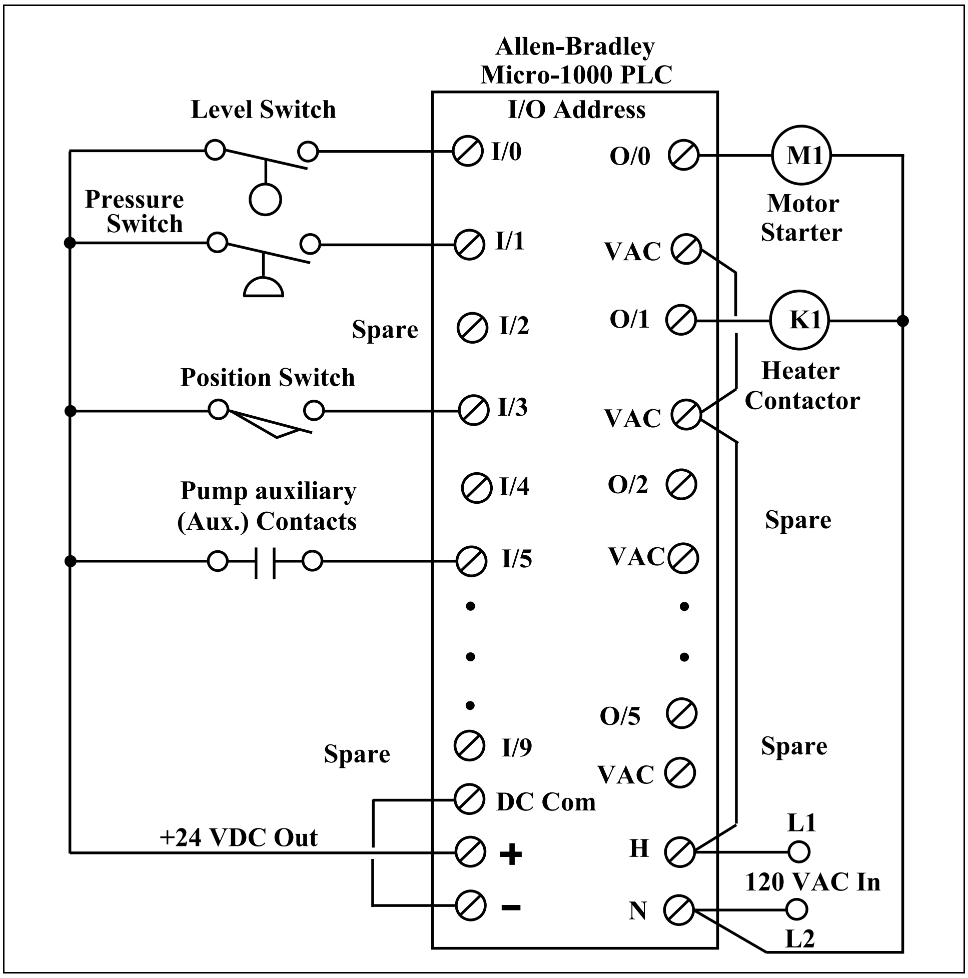


Figure 1-4. Typical Micro-1000 PLC Wiring Diagram

vibration, but the housings are normally mounted in control panels to provide complete protection from the harsh industrial environment encountered in typical application.

The backplane of the I/O chassis has sockets for each module. These sockets provide the power and data communications connection to the processor for each module.

Discrete Inputs/Outputs

Discrete is the most common class of input/output in a programmable controller system. This type of interface module connects field devices that have two discrete states, such as on/off or open/closed, to the processor. Each discrete I/O module is designed to be activated by a field-supplied

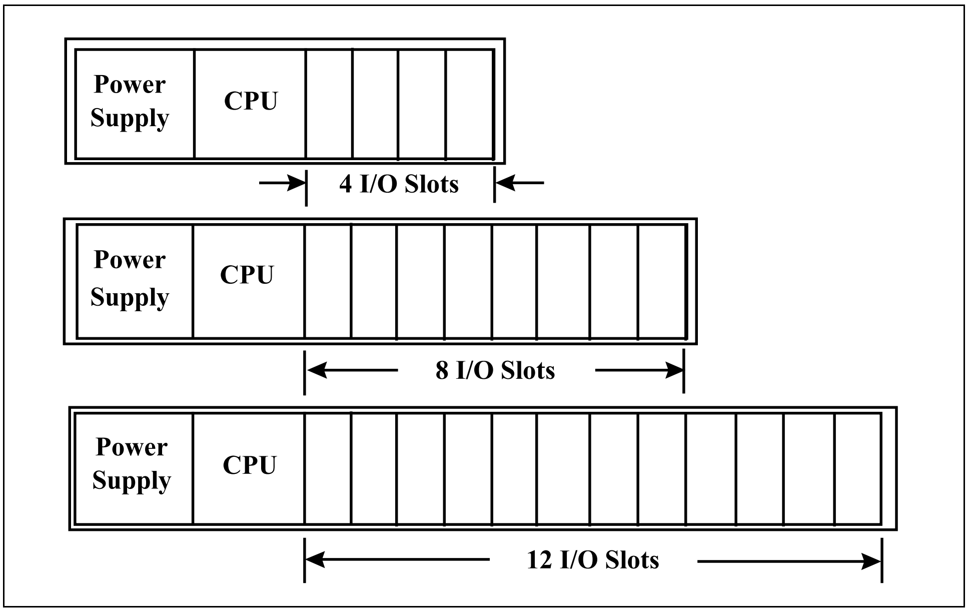


Figure 1-5. Typical I/O Modular Housings

voltage signal, such as +5 VDC, +24 VDC, 120 VAC, or 220 VAC, and so on.

In a discrete input (DI) module, if an input switch is closed, an electronic circuit in the input module senses the supplied voltage and converts it to a logic-level signal acceptable to the processor to indicate the status of that device. A logic 1 indicates ON or CLOSED, and a logic 0 indicates OFF or OPENED for a field input device or switch.

A typical discrete input module is shown in Figure 1-6. Most input modules will have a light-emitting diode (LED) to indicate the status of each input.

In a discrete output (DO) module, the output interface circuit switches the supplied control voltage that will energize or deenergize the field device. When an output is turned on by the control program, the supplied control voltage is switched by the interface circuit to activate the referenced (addressed) output device.

Figure 1-7 shows a typical discrete output module-wiring diagram. It can be thought of as a simple switch through which power can be provided to control the output device. During normal operation, the processor sends the output state determined by the logic program to the output module. The module then switches the power to the field device.

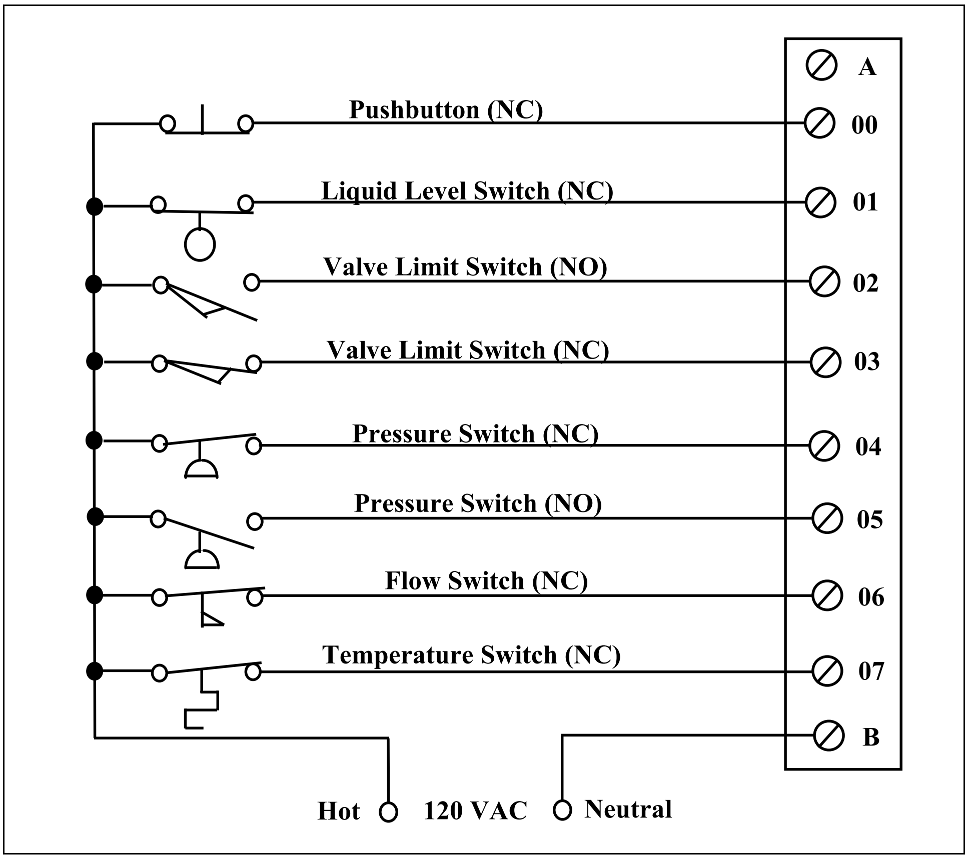


Figure 1-6. Typical Discrete Input Module Wiring Diagram

A fuse is normally provided in the output circuit of the module to prevent excessive current from damaging the wiring to the field device. If the fuse is not provided in the output module, it should be provided in the system design.

Analog I/O Modules

The analog I/O modules allow for monitoring and controlling of analog voltages and currents, which are compatible with many sensors, motor drives, and process instruments. With the use of analog I/O, most process variables can be measured or controlled with appropriate interfacing.

Analog I/O interfaces are generally available for several standard unipolar (single polarity) and bipolar (negative and positive polarity) ratings. In most cases, a single input or output interface can accommodate two or more different ratings and can satisfy either a current or a voltage requirement. The different ratings are either hardware (i.e., switches or jumpers) or software selectable.

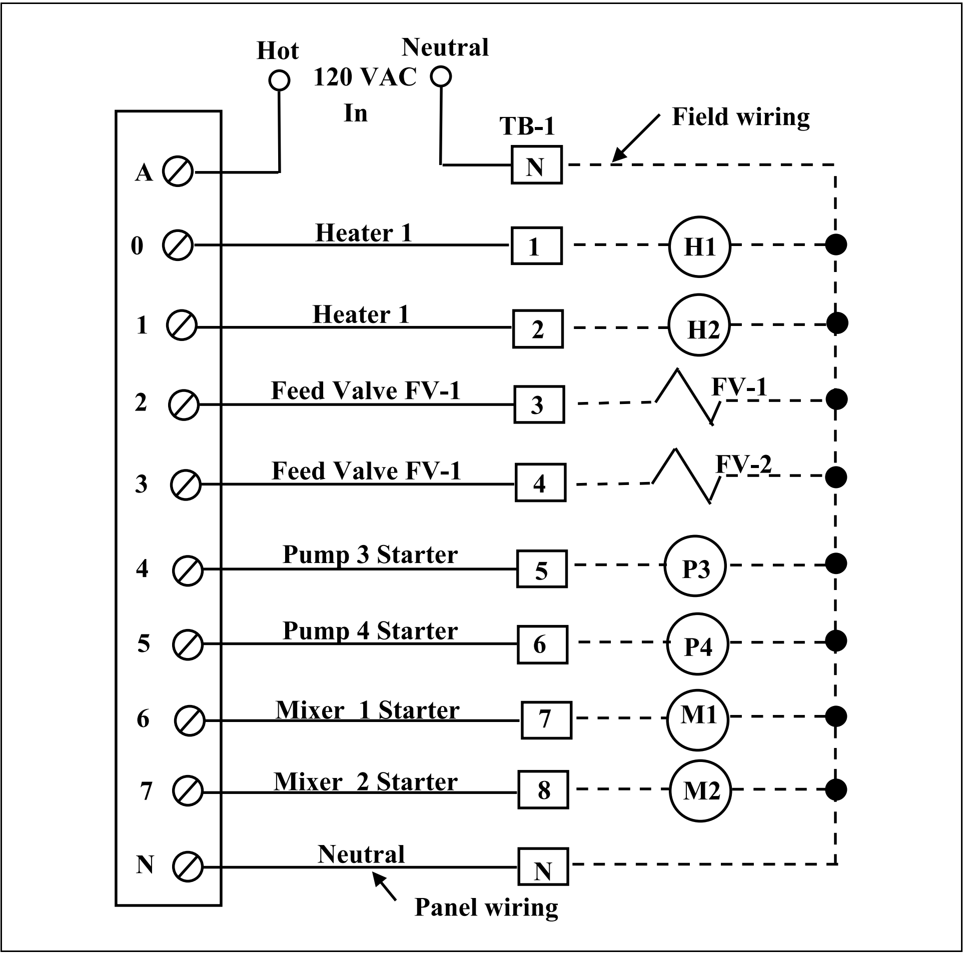


Figure 1-7. Typical Discrete Output Module Wiring Diagram

Digital I/O Modules

Digital I/O modules are similar to discrete I/O modules in that discrete ON/OFF signals are processed. However, the main difference is that discrete I/O interfaces require only a single bit to read an input or control an output. On the other hand, digital I/O modules process a group of discrete bits in parallel or serial form.

Typical devices that interface with digital input modules are binary encoders, bar code readers, and thumbwheel switches. Some instruments driven by digital output modules include LEDs and intelligent display panels.

Special Purpose Modules

The discrete and analog I/O modules will normally cover about 80% of the input and output signals encountered in programmable controller applications. However, to process certain types of signals or data efficiently, the programmable controller system will require special purpose modules. These special interfaces include those that condition input signals, such as thermocouple modules, pulse counters, or other signals that cannot be interfaced using standard I/O modules. Special purpose I/O modules may also use an on-board microprocessor to add intelligence to the interface. These intelligent modules can perform complete processing functions independent of the CPU and the control program scan.

Another important class of special purpose I/O modules is communication modules that communicate with distributed control systems (DCSs), other PLC networks, plant computers, or other intelligent devices.

Memory

Memory is used to store the control program for the PLC system; it is usually located in the same housing as the CPU. The information stored in memory determines how the input and output data will be processed.

Memory stores individual pieces of data called bits. A bit has two states: 1 or 0. Memory units are mounted on circuit boards and are usually specified in thousands or "K" increments where 1K is 1024 words (i.e., $2^{10} = 1024$) of storage space. Programmable controller memory capacity may vary from less than one thousand words to over 64,000 words (64K words) depending on the programmable controller manufacturer. The complexity of the control plan will determine the amount of memory required.

Although there are several different types of computer memory, they can always be classified as *volatile* or *nonvolatile*. Volatile memory will lose its programmed contents if all operating power is lost or removed. Volatile memory is easily altered and quite suitable for most programming applications when supported by battery backup and/or a recorded copy of the program. Nonvolatile memory will retain its data and program even if there is a complete loss of operating power. It does not require a backup system.

The most common form of volatile memory is Random Access Memory, or RAM. RAM is relatively fast and provides an easy means to create and store application programs. If normal power is disrupted, PLCs with RAM use battery or capacitor backups to prevent program loss.

The Electrically Erasable Programmable Read Only Memory (EEPROM) is a nonvolatile memory that is programmed through application software, which runs on a PC or through a micro PLC hand-held programmer.

There are two areas of memory in the PLC system that the user can access: program files and data files. Program files store the control application program, subroutine files, and the error file. Data files store data associated with the control program, such as input/output status bits, counter and timer preset and accumulated values, and other stored constants or variables. Together, these two general memory areas are called user or application memory. The processor also has an executive or system memory that directs and performs operational activities such as executing the control program and coordinating input scans and output updates. This process system memory, which is programmed by the PLC manufacturer, cannot be accessed or changed by the user.

Power Supply

The power supply converts AC line voltages to DC voltages to power the electronic circuits in a programmable controller system. These power supplies rectify, filter, and regulate voltages and currents to supply the correct amounts of voltage and current to the system. The power supply normally converts 120-VAC or 240-VAC line voltage into DC voltages such as +5 VDC, –15 VDC, or +15 VDC.

The power supply for a programmable controller system may be integrated with the processor, memory, and I/O modules into a single housing, or it might be a separate unit connected to the system through a cable. As a system expands to include more I/O modules or special function modules, most programmable controllers require an additional or auxiliary power supply to meet the increased power demand. Programmable controller power supplies are usually designed to eliminate electrical noise present on the AC power and signal lines of industrial plants so that this electrical noise does not introduce errors in the control system. They are also designed to operate properly in the higher temperature, vibration and humidity environments present in most industrial applications.

Communication Devices

The main function communication device or port is to communicate with the programming to enter, modify, and monitor the PLC control plan. In a small stand-alone PLC, there may be a single serial port to connect with the programming device. However, most PLC systems have more than one communication port or device. In a typical PLC system, there is the standard serial RS-232C port for programming and a vendor proprietary communication network that is used to transfer information between the remote I/O equipment racks and the other PLCs in the system. In larger PLC systems, there is normally an Ethernet link to communicate with PCs and other networks connected to the system.

Ethernet network has started to move onto the factory floor due in part to the confluence of the MIS systems and factory floor machine and process control systems. The widespread use of this type of network in general computer, PCs, telecomm systems, and the Internet has caused rapid advances in PLC Control System applications.

One general difference between Ethernet hardware in PCs and in PLCs is the equipment cost. While dropping for factory floor equipment, PC Ethernet hardware is very inexpensive, because most of the computing resource is being supplied by the PC. The PC Ethernet card generally supplies little more than the physical interface to the cable. The typical PLC Ethernet card supplies much of the computing power for the network and therefore is more complex.

Programming Languages

The programming language lets the user communicate with the programmable controller via a programming device. Programmable controller manufacturers use several different programming languages, but they all convey to the system, by means of instructions, a basic control plan.

A control plan or program is defined as a set of instructions that are arranged in a logical sequence to control the actions of a process or machine. For example, the program might direct the programmable controller to turn a motor starter on when a push button is depressed and at the same time direct the programmable controller to turn on a control panel-mounted RUN light when the motor starter auxiliary contacts are closed.

A program is written by combining instructions in a certain order. Rules govern the manner in which instructions are combined and the actual form of the instructions. These rules and instructions combine to form a language.

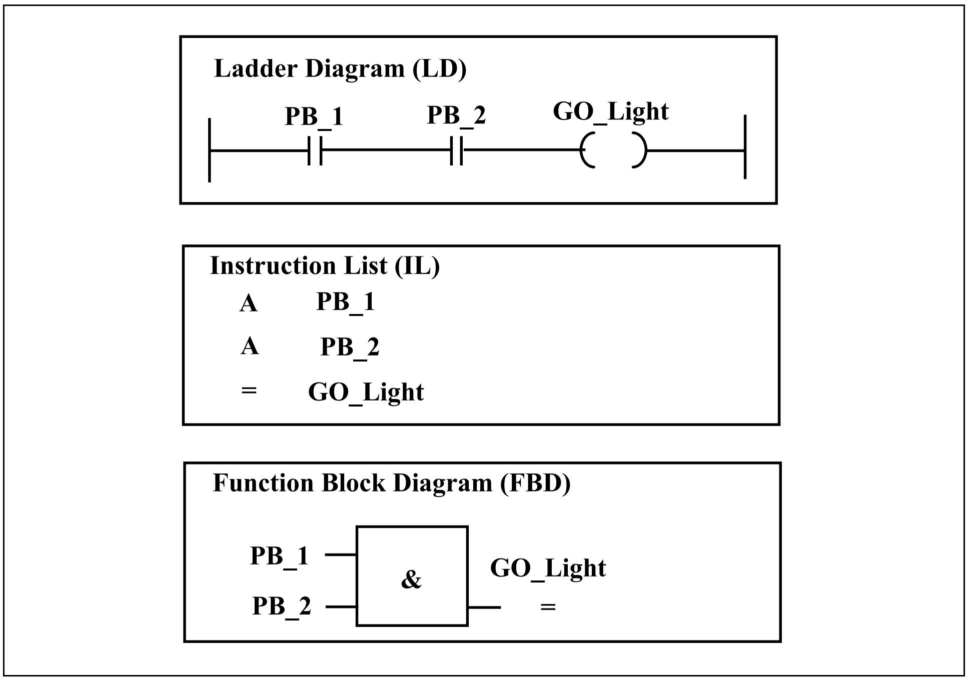
International Standard for PLC Languages

In the early 1970s, different national and international committees have proposed numerous PLC programming standards to develop a common interface for programmable controllers. Then, in 1979, a working group of international PLC experts was appointed by various national committees to write a first draft of a comprehensive PLC standard. The first committee draft was issued in 1982.

After an initial review of the document by the national committees, it was decided that the standard was too complex to handle as a single document. As a result, the working group was split into five task forces, one for each part of the standard. The subject of each part is as follows: Part 1, General Information; Part 2, Equipment and Testing Requirements; Part 3, Programming Languages; Part 4, User Guidelines; and Part 5, Communications. Each task group consisted of several international experts, each backed by a national advisory group. International Electrotechnical Commission (IEC) issued its standard for PLC programming languages in March of 1993 and assigned it the number IEC 61131-3.

The IEC standard has three graphical languages: Ladder Diagram (LD), Function Block Diagram (FBD), and Sequential Function Chart (SFC), and two text-based languages: Instruction List (IL) and Structured Text (ST). The PLC language standard allows different parts of an application to be programmed in different languages that can be combined into a single executable program. The three standard languages most commonly used in PLC applications are Ladder Diagram, Instruction List, and Function Block Diagram.

Figure 1-8 shows a simple logic function implemented using PLC language types. The logic function shown is an "AND" function (i.e., if pushbutton 1 (PB-1) is closed and pushbutton 2 (PB-2) is closed then the Go Light is on).



Instruction List (IL)

```
A     PB_1
A     PB_2
=     GO_Light
```

Figure 1-8. An Example Logic Function Using Standard PLC Languages

Ladder Diagram

Ladder Diagram (LD) is the most common and widely used language in PLC applications. The reason for this is relatively simple. The original programmable controllers were designed to replace electrical relay-based control systems. Technicians and engineers using electrical drawings called ladder diagrams designed these systems. The ladder diagram consists of a series of symbols interconnected by lines to indicate the flow of current through the various devices. The ladder drawing consists of basically two things: first is the power source, which forms the sides of the ladder (rails), and the second is the current that flows through the various logic input devices that form the rungs of the ladder. If there is electrical current flow through the relay contacts in a rung, the output relay coil will be turned on. This is termed "Power Flow" in the ladder rung.

In electrical design, the ladder diagram is intended to show only the circuitry necessary for the basic operation of the control system. Another diagram, called the *wiring diagram*, is used to show the physical connection of control devices. The discrete I/O module diagrams shown earlier are examples of wiring diagrams. A typical electrical ladder diagram is shown in Figure 1-9. In this diagram, a pushbutton (PB1) is used to energize a pump start control relay (CR1) if the level in a liquid storage tank is not high. Each device has a special symbol assigned to it to make reading of the diagram easier and faster.

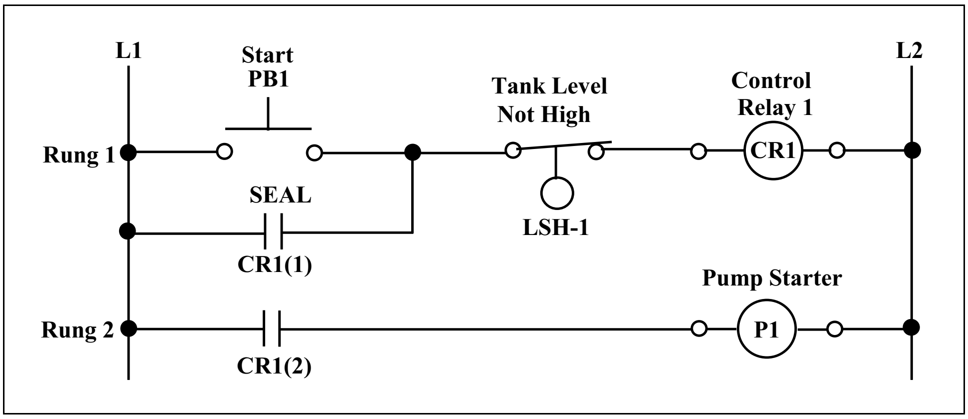


Figure 1-9. Typical Electrical Ladder Diagram

The same control application can be implemented using a PLC LD program as shown in Figure 1-10. The diagrams are read in the same manner from left to right, with the logic input conditions on the left and the logical outputs on the right. In the case of electrical diagrams, there must be electrical continuity to energize the output devices; for programmable controller ladder programs, there must be logic continuity to energize the outputs.

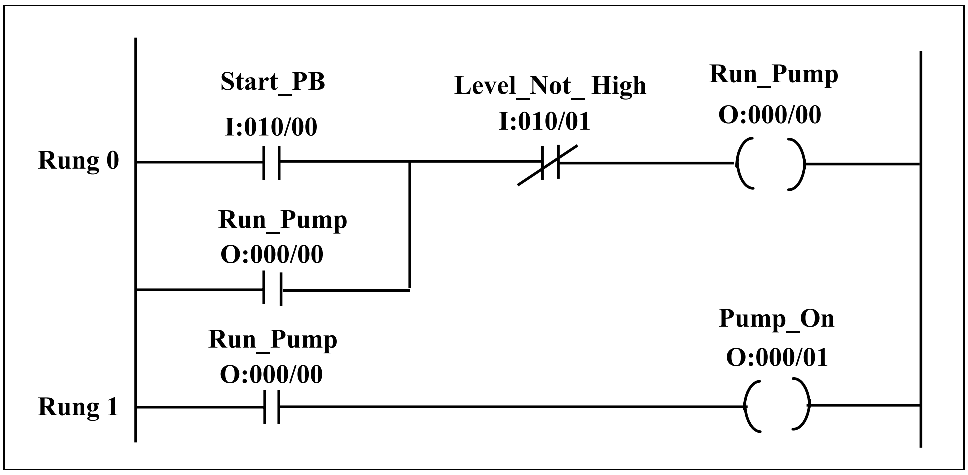


Figure 1-10. Typical Ladder Diagram Program

In ladder programs, three basic instructions are used to form the program. The first symbol is similar to the normally open (NO) relay contacts used in electrical ladder diagrams; this instruction uses the same NO symbol in ladder programs. It instructs the processor to examine its assigned bit location in memory. If the bit is ON (logic 1), the instruction is true and there is logic continuity through the instruction on the ladder rung. If the bit is OFF (logic 0), there is no logic continuity through the instruction on the rung.

The second important instruction is similar to the normally closed (NC) contact from electrical ladder diagrams and it is called the "examine off" instruction. Unlike the "examine on" instruction, it directs the processor to examine the bit for logical zero or the OFF condition. If the bit is OFF, the instruction is true and there is logic continuity through the instruction. If the bit is ON, the normally closed instruction is false and there is no logic continuity.

The third instruction is the output coil instruction. This instruction is similar to relay coil in electrical ladder diagrams and it directs the processor to set a certain location in memory to ON or 1, if there is logic continuity in any logic path preceding it. If there are no complete logic continuity paths in the ladder rung, the processor sets the output coil instruction bit to zero or OFF.

Figure 1-10 shows a typical Allen-Bradley Ladder Program. In this proprietary and non-standard ladder programming language, the letters

O or I followed by a 5-digit number above the instructions are the reference addresses for the logic bits. The letter "I" before the 5-digit number indicates an input bit and the letter "O" before the 5-digit number indicates an output bit. The reference address indicates where in the memory the logic operation will take place.

In the Allen-Bradley LD program shown in Figure 1-10, the "examine on" instruction for the start pushbutton (PB) directs the processor to see if the reference address I:010/00 is ON. In the same manner, the examine on instruction for the "tank level not high" input instructs the processor to see if the reference address I:010/01 is OFF. If there is logic continuity through both instructions, the output coil instruction at address O: 000/00 is turned ON. Logic continuity from the left side to the right end of a rung is called "Logical Power Flow" in PLC programming.

This same output bit is then used to "seal in" the start pushbutton instruction and turns on the energized instruction bit O:000/01 to turn on the pump run light.

In the IEC LD programming language standard, the input address I:010/00 would have the form %IX010.00 and the output address O:000/01 would be listed as %QX000.0. Now not all PLC manufacturers fully comply with the IEC programming standard, so many programming examples used in this book will not comply with the IEC standard. Before using any programmable example from this book, a PLC manufacturer's instruction manual must be consulted for the proper programming language format.

Instruction List

Instruction List (IL) is a textual programming language that can be used to create the code for a PLC control program. Its syntax for statements is similar to microprocessor assembly language and consists of instructions followed by addresses on which the instructions act. The IL language contains a comprehensive range of instructions for creating a complete user program. For example, in the Siemens S7 programming software package, there are over 130 different basic IL instructions and a wide range of addresses available depending on the model PLC used.

IL instruction statements have two basic structures. One, a statement made up of an instruction alone (e.g., NOT) and another where the statement is made up of an instruction and an address. The most common structure is for the statement to have an instruction and an address. The address of an instruction statement indicates a constant or the location where the instruction finds a value on which to perform an operation.

The most basic type of IL instructions is the Boolean Bit Logic Instructions. These instructions perform logic operations on single bits in PLC memory. The basic Bit Logic Instructions are: 1) “AND” (A) and its negated form “And Not” (AN), 2) “Or” (O), and 3) “Exclusive Or “ (Or) and its negated form, “Exclusive Or Not” (XN). These instructions check the signal state of a bit address to establish whether the bit is activated “1” or not activated “0”.

Bit logic instructions are also called relay logic instructions since they can execute commands that can replace a relay logic circuit. Figure 1-11 is an example of “AND” logic operation where the IL program is listed on the left side and the relay logic circuit is shown on the right side for comparison. In this example the IL program uses an AND instruction (A) to program two normally open (NO) contacts in series. Only when the signal state of both the NO contacts is “1”, can the state of output Q4.0 also be “1” and the coil is then energized.

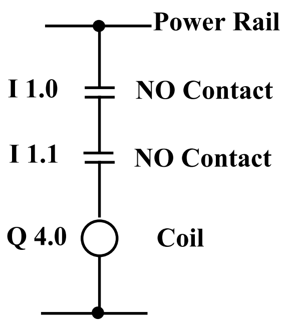
Instruction List Program	Relay Logic Diagram
A I 1.0 A I 1.1 = Q 4.0	

Figure 1-11. Comparison of IL Program and Relay Logic Circuit

Function Block Diagram

The function block diagram (FBD) is a graphical programming language. It allows the programmer to build complex control procedures by taking existing functions from the FBD library and wiring them in a graphic diagram area. An FBD describes a relationship or function between input and output variables. A function is described as a set of elementary function blocks as shown in Figure 1-12. Input and output variables are connected to blocks by connection lines.

An entire function operated by a FBD program is built with standard elementary function blocks from the FBD library. Each elementary function block has a fixed number of input connection points and a fixed number of output connection points. For example, the Boolean AND function block shown in Figure 1-12 has two inputs and only one output. The inputs are connected on its left border. The outputs are connected on its right border. An elementary function block performs a single function between its inputs and its outputs. For example, the elementary function block shown in Figure 1-12 performs the Boolean AND operation on its two inputs and produces a result at the output. The name of the function to be performed by the block is written in its symbol, in the case of the "AND" function the symbol is "&".

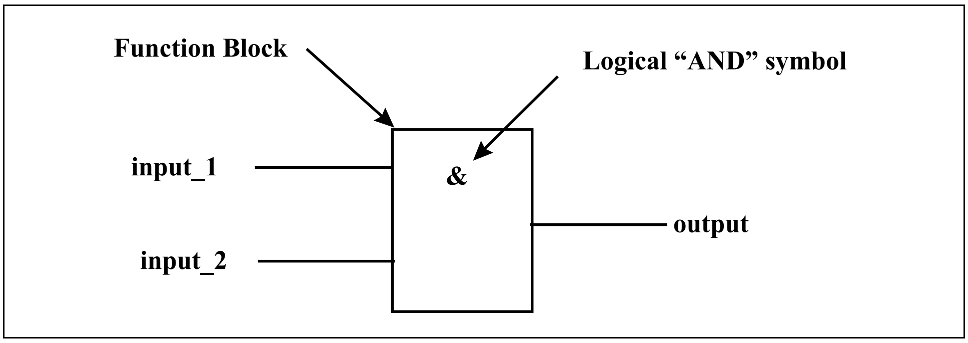


Figure 1-12. Typical Elementary Function Block

Programming Devices

The programming device can be a vendor-designed portable unit or a PC with programming software installed. The PC-based systems normally have the following basic components: keyboard, visual display or CRT, PC, printer, and communications interface card and cable as shown in Figure 1-13.

The programming devices are normally connected to the programmable controller system only during programming, start-up, or troubleshooting of the control system. Otherwise, the programming device is disconnected from the system.

The vendor-designed portable units are generally only used to program small programmable controllers. Most of these units resemble portable calculators, but with larger displays and a somewhat different keyboard. The displays are generally light-emitting diode (LED) or dot-matrix liquid crystal display (LCD), and the keyboard consists of alphanumeric keys, programming instruction keys, and special function keys. Even though they are mainly used for writing and editing the control program, the portable programmers are also used for testing, changing, and monitoring the program.

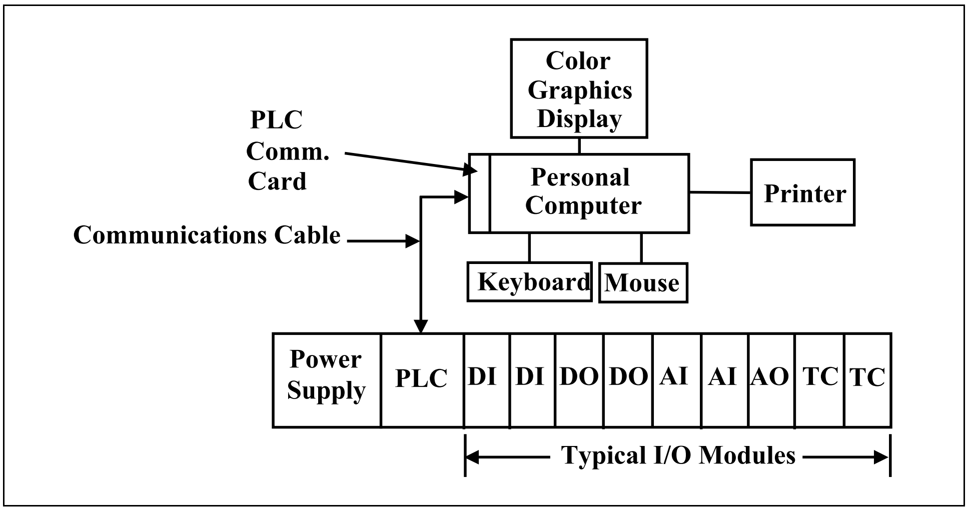


Figure 1-13. Typical Programmable Controller System

The standard programming terminal is a PC with the programming software loaded on the hard drive. These units can perform program editing and storage. They also have added features such as program printouts and connection to local area networks (LANs). LANs give the programmer or engineer access to any programmable controller in the communications network, so that any device in the network can be monitored and controlled. Normally, laptop PCs are used because they are light and portable and can be easily used in the field during testing, start-up, and modification of the control program.

Process Visualization

There are two methods commonly used to provide operators with real-time visual interface to programmable controller–based control systems. The first method is to wire directly from the programmable controller I/O modules to hard-wired lights and digital indicators on a display or control panel. This method is cost effective in small systems that will not be changed, but it is generally not recommended or used in larger control systems that might be expanded in the future.

The more common method is to use PCs with HMI software that can generate a visual representation of the process or machine being controlled. The process or machine display screens are usually based on the process and instrument drawings or mechanical drawings for the system being controlled and provide the operators with an overall view of the process or machine under control.

The software-based graphical display method has advantages in that the process display screens can be easily modified for process changes, and the software can perform other functions, such as control, alarms, messaging, report generation, historical trending, integrated real-time statistical process control (SPC), and batch process recipe control.

The process overview pictures are generally interesting to visitors but historical trending and alarm message display are more widely viewed and used by the operators in daily operations.

Alarms must be quickly analyzed and responded to by the operators to prevent product or process damage or injury to personnel. Most HMI software packages allow the operator to configure or tailor the messages to the process or plant being controlled. The messages can be derived from individual I/O bits or analog signals that are out of range or process limits. Message systems minimize process or machine downtime by alerting the operators to out-of-tolerance conditions or events.

The historical trending of key process or machine parameters allows the operator to detect and then prevent potential production or machine problems. The SPC software continuously records process or machine sequences and events that are related to product quality. This makes it possible to verify product quality on a continuous basis.

HMI software provides standard screen objects, such as pushbuttons, selector switches, and panel indicators to create user interfaces that are appropriate to the process being controlled. HMI systems not only acquire process data, messages, and events, the software may also store the information in archives and then the information can be made available on a filtered or sorted basis. The configuration engineer can design a message or alarm so that an operator must acknowledge the condition.

In a typical system, a Microsoft SQL Server is used to archive messages. The system archives information when a message or an alarm occurs or when there is a change in status. Message and alarm sequence logs are used to document events on a chronological basis. The system can generally print out all changes in status (new, departed or acknowledged) of the alarms or messages. Process values are generally archived cyclically on an event or limits violation basis. The analog process values can also be archived on a condensed basis using signal averaging.

The Microsoft SQL Server uses effective loss-free data compression functions so that memory resources are conserved. However, a large process can generate an archive of considerable size so the user must set a maximum archiving period, such as one week or month, to limit the data volume. Each data archive can be segmented (e.g., per week, per month) and then the completed individual archives can be exported to a long-term archive server.

HMI software packages have reporting systems that allow printing of process data in various formats, such as message sequence logs, system message logs, or user production reports. Before the reports are printed out, they can be reviewed on a monitor and saved to a report file. Most HMI packages make it possible to start a report on a time or event driven basis or on command from the operator. The user can generally assign separate printers to each print job.

The HMI graphics design software is generally simple and easy to use. Graphic objects can be easily linked to PLC internal tags for animation or control purposes. When a new object has been placed on a graphics screen, an easy-to-edit dialog box appears on the screen. Graphics designer software allows the operator to specify and animate virtually all object properties. Using OPC, the system operator can connect the HMI software to any communication device for which an OPC-compliant server or driver is available. HMI systems can function as both a native OPC client and OPC server, and most HMI software packages support browsing for OPC server addresses.

The HMI software packages provide Wizards (software assistants) to aid the configuration engineer in the system development. For example, a message wizard might offer default settings that the developers can confirm, modify, or reject. Preview windows are used to display the effects of any chosen parameters. If the configuration designer confirms the setting, the solution is implemented by the system.

HMI systems provide a library of configured display and control modules like valves, process tanks, pumps, pipes, instruments, and switches. The developer can also create custom objects and store them in the library for future use. Objects are generally stored by topics in a library. When needed, objects are simply brought into the process screen using the standard computer drag and drop technique.

A completed control and graphics configuration can be tested without having to connect to a PLC by using simulation software that is part of the HMI software system. To simulate PLC inputs or outputs, each point can be assigned a characteristic value (analog or discrete). When a picture appears on the PC screen during a test, it becomes clear, via a color or value change whether or not the control, display, or alarm objects are functioning properly.

Personal Computer–Based PLCs

The use of PCs to directly replace PLCs in control applications is becoming more widespread. One disadvantage of programmable controllers is that they use proprietary hardware and software, which require additional engineering to adapt to plant or technological changes.

After decades of exponential growth in capabilities, PCs now come in rugged packages, with ever-faster processors, larger memory, and a multitude of Windows-based software products to support process control. All these improvements in the performance of PCs have come with lower cost due to higher competition and high-volume manufacturing.

There are two types of PC-based controllers used to replace proprietary PLCs in control applications. The first is a system that consists of an industrial grade PC running under Windows operating software and connected to standard PLC input and output modules. A typical example is the Allen-Bradley SoftLogix 5 Controller. This is a software package that runs on an industrial grade PC using Microsoft Windows operating engine.

Using a SoftLogix 5 controller, the user can integrate programming, control, and HMI functions on a single PC. The Windows NT/Windows 2000 soft control engine provides the open architecture and power of PCs, combined with the functionality of the Allen-Bradley PLC-5 processor. The system can monitor and control across ControlNet, DeviceNet, Universal Remote I/O, and some third-party networks. The SoftLogix 5 system is programmed using Allen-Bradley software running on a PC.

The second type is a software system called SoftPLC® that converts an industrial PC (x86 based) into a full-function PLC-like process controller. The SoftPLC system combines the standard PID; discrete and analog control functions typical of PLCs with the data handling, computing and network capabilities of PCs.

The SoftPLC system provides a complete instruction set and almost unlimited user program and data table memory space. SoftPLC is normally configured as an embedded system and runs on PC hardware much like a PLC but with no hard drive, monitor, keyboard, or mouse attached to the system. It runs as an embedded 32-bit real-time multitasking kernel on 386, 486, Pentium, and other x86-compatible processor platforms. Since a SoftPLC controller does not have a Windows operating system installed, there are no concerns about the system becoming non-deterministic or failing suddenly.

SoftPLC's ladder logic instruction set is a superset of the Allen-Bradley PLC-5 ladder logic set. The standard ladder instructions include: contacts, coils and branching; timers and counters; data comparisons and moves; math and logical operation; shift registers and sequencers; jumps and subroutines, and special functions (such as PID control, messaging and diagnostics).

The ladder logic program and data table application can be very large. The program size is only limited by the amount of RAM in the processor. The ladder logic programs can contain up to 998 program files that are organized as a main program with ladder subroutine areas. The data table can be configured with up to 1000 files of 1000 elements each.

The main advantage of the SoftPLC system is that it is an open architecture, non-proprietary system, so the user can easily interface to other vendor control devices and can easily connect to PC networks and other industrial networks.

EXERCISES

1.1 Explain the operation and purpose of the processor in a typical programmable controller system.

1.2 What is the main purpose of the input and output system in a PLC system?

1.3 List some typical discrete input devices found in process industries.

1.4 List some typical discrete output devices encountered in industrial applications.

1.5 List several typical analog signal values found in process applications.

1.6 Explain the difference between volatile and nonvolatile memory.

1.7 List some common applications for PCs in programmable controller systems.

1.8 What is the most common device used to program PLCs?

1.9 Discuss the three basic instructions used in ladder logic programs.

1.10 What is power flow in a relay ladder diagram?

1.11 Explain the concept of logical power flow in a ladder diagram program.

1.12 What are the two basic structures used in IL instruction statements?

1.13 What are the most common methods used to produce a visual representation of a process or machine under control?

1.14 What are the two types of PC-based control systems used to replace proprietary PLCs?

1.15 What is the main advantage of a SoftPLC system?

BIBLIOGRAPHY

1. Webb, J. W., and Reis, R. A., *Programmable Logic Controllers—Principles and Applications*, Third Edition, Prentice-Hall, Inc., 1995.

2. *Micro Mentor-Understanding and Applying Micro Programmable Controllers*, Allen-Bradley Company, Inc., 1995.

3. Wisnosky, D. E., *SoftLogic: Overcoming Funnel Vision*, Wizdom Controls, Inc., 1996.

4. Poque, D., *Windows Me: The Missing Manual,* Poque Press, LLC. 2000.

2 Binary Logic Fundamentals

Introduction

The design and maintenance of programmable controllers in process control applications requires an understanding of binary logic fundamentals. In this chapter, the basic concepts of binary signals, numbering systems, binary logic functions and relay ladder logic design will be discussed. Since data in programmable logic controllers (PLCs) is in binary form (0 or 1), we will first discuss binary signals and codes.

Binary Signals and Codes

Using PLCs in process control requires that the measurement and control signals be encoded into binary form. Binary signals are simply two-state signals (on/off, start/stop, high voltage/low voltage, etc.).

The simplest approach to encoding analog data into a binary-based digital word is provided by the American Standard Code for Information Interchange (ASCII). This method uses a pattern of 7 bits (1s and 0s) to represent letters and numbers. Sometimes an extra bit (parity bit) is used to check that the correct pattern has been transmitted. For example, the number 1 is represented by 011 0001 and the number 2 is represented by 011 0010. On the other hand, the capital letter A is represented by 100 0001 and the capital letter B by 100 0010.

Many other methods are used for encoding digital numbers in programmable controllers and digital computers. The most common method is the simple binary code that we will be discussed later, but first we will briefly cover numbering systems.

Numbering Systems

The most commonly used numbering system in programmable controllers is the binary system, but the octal and hexadecimal numbering systems are also encountered in these applications. We will start with a brief review of the decimal numbering system, followed by coverage of the other three systems.

Decimal Numbering System

The decimal numbering system is in common use probably because early on people started to count with their fingers. However, the decimal system is not easy to implement electronically. A 10-state electronic device would be quite costly and complex. It is much easier and more efficient to use the binary (2-state) numbering system when manipulating numbers using logic circuits found in computers.

A decimal number, N_{10} can be written mathematically as:

$$N_{10} = d_nR^n + \ldots\ldots + d_2R^2 + d_1R^1 + d_0R^0 \tag{2-1}$$

where R is equal to the number of digit symbols used in the system. R is called the radix and is equal to 10 in the decimal system. The subscript 10 on the number N in Equation 2-1 indicates that it is a decimal number. However, it is common practice to omit this subscript in writing out decimal numbers. The decimal digits, $d_n \ldots d_2, d_1, d_0$, can assume the values of 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9 in the decimal numbering system.

For example, the decimal number 1735 can be written as:

$$1735 = 1x10^3 + 7x10^2 + 3x10^1 + 5x10^0$$

$$1735 = 1000 + 700 + 30 + 5.$$

When written as 1735, the powers of 10 are implied by positional notation. The value of the decimal number is computed by multiplying each digit by the weight of its position and summing the result. As we will see, this is true for all numbering systems; the decimal equivalent of any number can be calculated by multiplying the digit times its base raised to the power of the digit's position. The general equation for numbering systems is:

$$N_b = Z_nR^n + \ldots\ldots + Z_2R^2 + Z_1R^1 + Z_0R^0 \tag{2-2}$$

Where Z is the value of the digit, R is the radix or base of the numbering system.

Binary Numbering System

The binary numbering system has a base of 2, and the only allowable digits are 0 or 1. This is the basic numbering system for computers and programmable controllers, which are basically electronic devices that manipulate 0s and 1s to perform math and control functions.

It was easier and more convenient to design digital computers that operate on two entities or numbers rather than the 10 numbers used in the decimal world. Furthermore, most physical elements in the process environment have only two states, such as a pump on or off, a valve open or closed, a switch on or off, and so on.

A binary number follows the same format as a decimal one, that is the value of a digit is determined by its position in relation to the other digits in a number. In the decimal system, a 1 by itself is worth 1; placing it to the left of a zero makes the 1 worth 10, and putting it to the left of two zeros makes it worth 100. This simple rule is the foundation of the numbering systems. For example, numbers to be added or subtracted are first arranged so that their place columns line up.

In the decimal system, each position to the left of the decimal point indicates an increasing power of 10. In the binary system, each place to the left signifies an increased power of two (i.e., 2^0 is one, 2^1 is two, 2^2 is four, 2^3 is eight, and so on). So, finding the decimal equivalent of a binary number is simply a matter of noting which place columns the binary 1s occupy and adding up their values. A binary number also uses standard positional notation. The decimal equivalent of a binary number can be found using the equation:

$$N_2 = Z_n 2^n + \dots\dots\dots + Z_2 2^2 + Z_1 2^1 + Z_0 2^0 \qquad (2\text{-}3)$$

where the radix or base equals 2 in the binary system, and each binary digit (bit) can have the value of 0 or 1 only.

The decimal equivalent of the binary number 10101_2 can be found as follows:

$$10101_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

or

$$(1 \times 16)+(0 \times 8)+(1 \times 4)+(0 \times 2)+(1 \times 1)= 21 \text{ (decimal)}$$

EXAMPLE 2-1

Problem: Convert the binary number 101110_2 to its decimal equivalent.

Solution: Using Equation 2-3

$$N_2 = Z_n2^n + \ldots\ldots\ldots + Z_22^2 + Z_12^1 + Z_02^0$$

or

$$\begin{aligned} N_2 &= 1 \times 2^5 + O \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= (1 \times 32) + (O \times 16) + (1 \times 8) + (1 \times 4) + (1 \times 2) + (O \times 1) \\ &= 32 + 0 + 8 + 4 + 2 + 0 \\ &= 46 \end{aligned}$$

To this point, we have discussed only positive binary numbers. Several common methods are used to represent negative binary numbers in programmable controller systems. The first is signed-magnitude binary. This method places an extra bit (sign bit) in the left-most position and lets this bit determine whether the number is positive or negative. The number is positive if the sign bit is 0 and negative if the sign bit is 1. For example, in a 16-bit machine, if we have a 12-bit binary number, $000000010101_2 = 21_{10}$, to express the positive and negative values, we would manipulate the left most or most significant bit. So, using the signed magnitude method: 0000000000010101 = + 21 and 1000000000010101 = –21.

Another common method used to express negative binary numbers is called two's complement binary. To complement a number means to change it to a negative number. For example, the binary number 10101 is equal to decimal 21. To get the negative using the two's complement method, you complement each bit and then add 1 to the least significant bit.

In the case of the binary number 010101 = 21, its two's complement would be: 101011 = – 21.

Octal Numbering System

The binary numbering system requires substantially more digits to express a number than the decimal system. For example, $130_{10} = 10000010_2$, so that it takes 8 or more binary digits to express a decimal number over 127. It is also difficult for people to read and manipulate large numbers without making errors. To reduce errors in binary number manipulations, some computer manufacturers started using the octal numbering system. This system uses the number 8 as a base or radix with the 8 digits 0, 1, 2, 3, 4, 5, 6, 7. Like all other number systems, each digit in an octal number has a weighted value according to its position.

For example,

$$1301_8 = 1 \times 8^3 + 3 \times 8^2 + 0 \times 8^1 + 1 \times 8^0$$

$$= 1 \times 512 + 3 \times 64 + 0 \times 8 + 1 \times 1$$

$$= 512 + 192 + 0 + 1$$

$$= 705_{10}$$

The octal system is used as a convenient means of writing or manipulating binary numbers in PLC systems. A binary number with a large number of 1s and 0s can be represented with an equivalent octal number with fewer digits. As shown in Table 2-1, one octal digit can be used to express three binary digits so that the number is reduced by a factor of three.

Binary	Octal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Table 2-1. Binary and Octal Equivalent Numbers

For example, the binary number 11010101010_2 can be converted to an octal number by grouping binary bits in groups of three starting with the least significant bit, as follows:

$$11\ 010\ 101\ 010 = 3252_8$$

EXAMPLE 2-2

Problem: Represent the binary number $\mathbf{101011001101111_2}$ in octal.

Solution: To convert from a binary to an octal number, we simply divide the binary number into groups of three bits, starting with the least significant bit (LSB). Then we use Table 2-1 to convert the 3-bit groups to their octal equivalent.

To solve, we place binary number into groups of three: $101\ 011\ 001\ 101\ 111_2$. Since $101_2 = 5_8$, $011_2 = 3_8$, $001_2 = 1_8$, $101_2 = 5_8$, and $111_2 = 7_8$,

we obtain $\mathbf{101011001101111_2 = 53157_8}$.

We can convert a decimal number to an octal number by successively dividing the decimal number by the octal base number 8. This is best illustrated in the following example.

EXAMPLE 2-3

Problem: Convert the decimal number 370_{10} to an octal number.

Solution: Decimal to octal conversion is obtained by successive division by the octal base number 8 as follows:

So that $370_{10} = 562_8$.

Hexadecimal Numbering System

The hexadecimal numbering system provides an even shorter notation than the octal system and is a commonly used numbering system in PLC applications. The hexadecimal system has a base of 16, and four binary bits are used to represent a single symbol. The sixteen symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The letters A through F are used to represent the binary strings 1010, 1011, 1100, 1101, 1110, and 1111, which correspond to the decimal numbers 10 through 15. The hexadecimal digits and their binary equivalents are given in Table 2-2.

To convert a binary number to a hexadecimal number, we use Table 2-2.

For example, the binary number 0110 1111 1000 is 6F8. Again, the hexadecimal numbers follow the standard positional convention H_n . . . H_2, H_1, H_0 where the positional weights for hexadecimal numbers are powers of sixteen with 1, 16, 256, and 4096 being the first four decimal values.

Hexadecimal	Binary	Hexadecimal	Binary
0	0000	9	1001
1	0001	A	1010
2	0010	B	1011
3	0011	C	1100
4	0100	D	1101
5	0101	E	1110
6	0110	F	1111
7	0111	10	10000
8	1000	11	10001

Table 2-2. Hexadecimal and Binary Equivalent Numbers

To convert from hexadecimal to decimal numbers, we use the following equation:

$$N_{16} = H_n16^n + ... + H_216^2 + H_116^1 + H_016^0 \qquad (2\text{-}4)$$

where the radix equals 16 in the hexadecimal system, and each digit can take on the value of 0 through 9 and the letters A, B, C, D, E, and F.

EXAMPLE 2-4

Problem: Convert the hex number 1FA to its decimal equivalent.

Solution: Using equation 2-4:

$$N_{16} = H_n16^n + ... + H_216^2 + H_116^1 + H_016^0$$

and since $H_2 = 1 = 1_{10}$, $H_l = F = 15_{10}$, and $H_O = A = 10_{10}$,

we obtain:

$$N_{16} = 1 \times 16^2 + 15 \times 16^1 + 10 \times 16^0$$

$$= 256 + 240 + 10 = 506_{10}$$

To convert a decimal number to a hexadecimal number, we use the following procedure:

1. Divide the decimal number by 16 and record the quotient and the remainder.

2. Divide the quotient from the division in Step 1 by 16 and record the quotient and the remainder.
3. Repeat Step 2 until the quotient is zero.
4. The hexadecimal equivalents of the remainders generated by the divisions are the digits of the hexadecimal number, where the first remainder is the least significant digit (LSD) and the last remainder is the most significant digit (MSD).

An example will illustrate the conversion from a decimal to a hexadecimal number.

EXAMPLE 2-5

Problem: Convert the decimal number 610_{10} to a hexadecimal number.

Solution:

Division	Quotient	Remainder
610/16	38	2 (LSD)
38/16	2	6
2/16	0	2 (MSD)

Therefore, $610_{10} = 262_{16}$.

It is important to note that the octal and hexadecimal systems are used for human convenience only, and the computer system actually converts the octal and hex numbers into binary strings and operates on the binary digits.

Binary Data Codes

Data codes translate information (alpha, numeric, or control characters) to a form that can be transferred electronically and then converted back to its original form. A code's efficiency is a measure of its ability to utilize the maximum capacity of the bits and to recover from error. In the evolution of the various codes, there has been a steady increase in their efficiency to transfer data. A brief discussion of the commonly used codes follows.

Binary Code

It is possible to represent 2^n different symbols in a purely binary code of n bits. The binary code is a direct conversion of the decimal number to the binary. This is illustrated in Table 2-3.

Decimal	Binary	Decimal	Binary
0	00000	11	01011
1	00001	12	01100
2	00010	13	01101
3	00011	14	01110
4	00100	15	01111
5	00101	16	10000
6	00110	17	10001
7	00111	18	10010
8	01000	19	10011
9	01001	20	10100
10	01010	21	10101

Table 2-3. Binary to Decimal Code

It is the most commonly used code in computers because it is a systematic arrangement of the digits.

It is also a weighted code, where each column has a magnitude of 2^n associated with it, and it is easy to translate. In Table 2-3, note that the least significant bit (LSB) alternates every time, whereas the second least significant bit repeats every two times, the third least significant bit repeats every four times, and so on.

Baudot Code

The Baudot code was the first successful data communications code. It is also known as the International Telegraphic Alphabet #2 (ITA#2). The code was meant primarily for text transmission. It has only uppercase letters and is used with punched paper tape units on teletypewriters. It uses five consecutive bits and an additional start/stop bit to represent a data character as shown in Figure 2-1. It transfers asynchronous data at a very slow rate (10 characters per second) using a teletypewriter.

Most early teletypewriters used basically the same circuit as the telegraph with the mechanics of a typewriter. As with the telegraph, the teletypewriter had to have at one end a method of knowing when the other end wanted to transmit, so a mark signal (current) would be sent as a "line idle" signal. Since a mark is the idle condition, the first element or bit of any code would have to be a space (no current). This bit is known as the "start space" as shown in Figure 2-1. Also, the "current on" (mark) condition needs to exist after the character code pulses have been sent, so the receiver device will know when the character is complete and separate this transmission character from the next character to be transmitted. This period of current is known as the "stop mark" and is either 1, 1.42, or 2

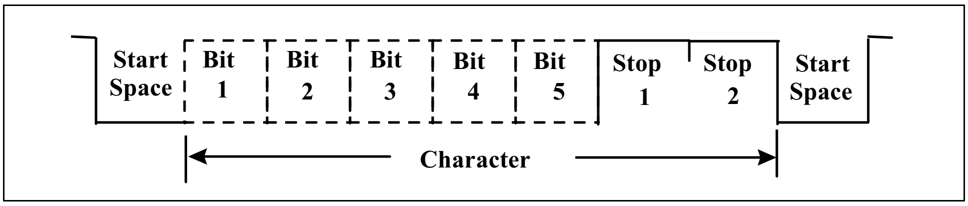


Figure 2-1. Baudot Character Communication Format

elements in duration. The bit time or duration is determined by the teletype's motor speed.

The Baudot or teletypewriter code is given in Table 2-4. There are 26 uppercase letters, 10 numerals, and various items of punctuation and teletype control. This code uses five bits (2 to a 5th power) or 32 patterns. However, the code developers used the mechanical shift of the teletypewriter to produce 26 patterns out of a possible 32 for letters and 26 patterns shifted for numbers and punctuation. Only 26 were available in either shift because six patterns were the same for both as shown in Table 2-4. The six common patterns are carriage return, line feed, shift up (figures), shift down (letters), space, and blank (no current).

This binary code is still the most efficient code for narrative text in terms of transmission overhead because it requires very little machine operation or error detection. While this code is no longer widely used, at one time it was the most extensively used binary transmission code.

A disadvantage of the code is that it can represent only the 58 characters shown in Table 2-4. Other limitations of Baudot code are the sequential nature of the code, the high overhead, and the lack of error detection.

BCD Code

As computer and data communications technology improved, more efficient codes were developed. The BCD, binary coded decimal code, was first used to perform internal numeric calculations within data processing devices. The BCD code is commonly used in programmable controllers to

Character Case		Bit Pattern	Character Case		Bit Pattern
Lower	Upper	54321	Lower	Upper	54321
A		00011	Q	1	10111
B	?	11001	R	4	01010
C	:	01110	S	‘	00101
D	$	01001	T	5	10000
E	3	00001	U	7	00111
F	!	01101	V	;	11110
G	&	11010	W	2	10011
H	#	10100	X	/	11101
I	8	00110	Y	6	10101
J	Bell	01011	Z	“	10001
K	(	01111	Letters Shift Down		11111
L	)	10010	Figures Shift Up		11011
M	.	11100	Space		00100
N	,	01100	Carriage Return		01000
O	9	11000	Line Feed		00010
P	0	01101	Blank or null		00000

Table 2-4. Baudot Code

code data to numeric LEDs and from panel-mounted digital thumbwheel units. Its main disadvantages are that it has no alpha characters and no error-checking capability. A listing of the BCD code for decimal numbers from 0 to 19 is given in Table 2-5.

Note that four-bit groups are used to represent the decimal numbers 0 through 9. To represent higher numbers, such as 10 through 19 another four-bit group is used and place to the left of the first four-bit group.

Decimal	BCD Code	Decimal	BCD Code
0	0000	10	0001 0000
1	0001	11	0001 0001
2	0010	12	0001 0010
3	0011	13	0001 0011
4	0100	14	0001 0100
5	0101	15	0001 0101
6	0110	16	0001 0110
7	0111	17	0001 0111
8	1000	18	0001 1000
9	1001	19	0001 1001

Table 2-5. BCD Code

EXAMPLE 2-5

Problem: Convert the following decimal numbers to BCD code:

(a) 276,

(b) 567,

(c) 719, and

(d) 4500.

Solution: Using Table 2-5, the decimal numbers can be expressed in BCD code as follows:

(a) 276 = 0010 0111 0110,

(b) 567 = 0101 0110 0111,

(c) 719 = 0111 0001 1001, and

(d) 4500 = 0100 0101 0000 0000

ASCII Code

The most widely used code is ASCII, the American Standard Code for Information Interchange, which was developed in 1963. This code has 7 bits for data (allowing 128 characters) as shown in Table 2-6.

The ASCII code can operate synchronously or asynchronously with 1 or 2 stop bits. ASCII format has 32 control characters. The legend for these control characters are listed in Table 2-7. These control codes are used to indicate, modify, or stop a control function in the transmitter or receiver. Seven of the ASCII control codes are called *format effectors.* They pertain to the control of a printing device. Using format effectors increases code efficiency and speed by replacing frequently used character combinations with a single code. The following is a list of the format effectors used in the ASCII code: BS (backspace), HT (horizontal tab), LF (line feed), VT (vertical tab), FF (form feed), CR (carriage return), and SP (space).

Bits	7	0	0	0	0	1	1	1	1
	6	0	0	1	1	0	0	1	1
	5	0	1	0	1	0	1	0	1
4321	HEX	0	1	2	3	4	5	6	7
0000	0	NUL	DLE	SP	0	@	P	‘	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	“	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	’	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	’	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	-	o	DEL

Table 2-6. ASCII Code

EXAMPLE 2-6

Problem: Express the words PUMP 100 ON using ASCII code. Use Hex notation for brevity.

Solution: Using Table 2-6, we obtain:

MESSAGE:	PUMP 100 0N
ASCII (Hex) String:	50 55 4D 50 20 31 30 30 20 4F 4E

Mnemonic	Meaning	Mnemonic	Meaning
NUL	Null	DLE	Data Link Escape
SOH	Start of heading	DC1	Device Control 1
STX	State of Text	DC2	Device Control 2
ETX	End of Text	DC3	Device Control 3
EOT	End of Transmission	DC4	Device Control 4
ENQ	Enquiry	NAK	Negative Acknowledge
ACK	Acknowledge	SYN	Synchronous Idle
BEL	Bell	ETB	End of Transmission Block
BS	Backspace	CAN	Cancel
HT	Horizontal Tabulation	EM	End of Medium
LF	Line Feed	SUB	Substitute
VT	Vertical Tabulation	ESC	Escape
FF	Form Feed	FS	File Separator
CR	Carriage Return	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator
		DEL	Delete

Table 2-7. Legend for ASCII Control Characters

Binary Logic Functions

In control system applications, binary numbers 1 and 0 are represented by voltage levels, relays contact status, switch position, etc. For example, in transistor-transistor logic (TTL) gates, a binary 1 is represented by a voltage signal in the range of 2.4 to 5.0 V, and a binary 0 is represented by a voltage level between 0 and 0.8 V. Solid-state electronic circuits are available that can be used to manipulate digital signals to perform a variety of logical functions, such as NOT, AND, OR, NAND, and NOR. In hardwired electrical logic systems, electrical relays are used to implement the logic functions.

NOT Function

The most basic binary logic function is the NOT or inversion function. The NOT, or logic inverter, produces an output opposite to the input. An inversion bar is drawn over a logic variable to indicate the NOT function. For example, if a *NOT* operation is performed on a logic variable *A*, it is designated by $Z = \overline{A}$. The binary logic truth table for the NOT function in Table 2-8 lists the results of the NOT function on the Input A. In relay-based logic circuits, a normally closed (NC) set of contacts is used to perform the NOT function as shown in Figure 2-2. If the electric relay A is not energized, there is electrical current flow or logic continuity through the normally closed contacts so that output relay coil is energized or On. In other words, if input A is logic 0 or not On, then the output Z is logic 1 or On. If input A is logic 1 or On, the normally closed contacts are opened and there is no current flow or logic flow in the circuit and relay Z is off and output Z is 0.

Input	Output
A	Z
0	1
1	0

Table 2-8. NOT Function Binary Logic Truth Table

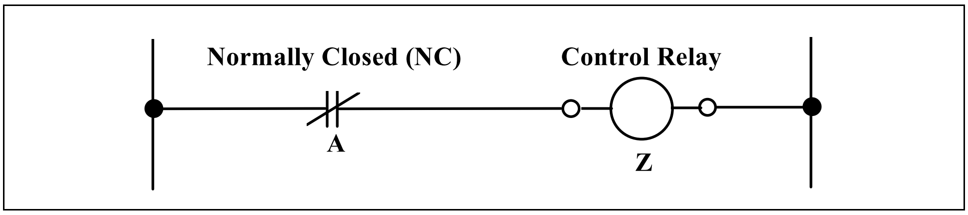


Figure 2-2. NOT Function Implemented with Relay

OR Function

A logical OR function, with two or more inputs and a single output, operates in accordance with the following definition: *The output of an OR function assumes the 1 state if one or more inputs assume the 1 state*.

The inputs to a logic function OR gate can be designated by $A, B, \ldots, N$ and the output by Z. It is assumed that the inputs and outputs can take only one of two possible values, either 0 or 1. The logic expression for this function is $Z = A + B + \ldots + N$. A two input truth table for an OR function is given in Table 2-9 for a two-input OR function.

Input		Output
A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

Table 2-9. Two Input OR Function Truth Table

An example of OR logic in process control would be as follows: If the water level in a hot water heater is low, or the temperature in the tank is too high, a logic system can be designed to turn off the heater in the system using logic circuits or relays. Figure 2-3 shows this application using relays to perform the logic function.

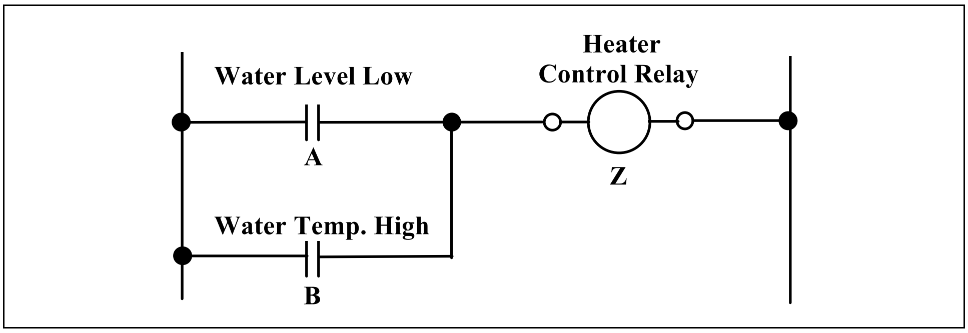


Figure 2-3. Relay-based OR Logic Control Application

The following logic identities for OR functions can be easily verified using the two-input truth table for the OR function given in Table 2-9.

$$A + B + C = (A + B) + C = A + (B + C) \tag{2-5}$$

$$A + B = B + A \tag{2-6}$$

$$A + A = A \tag{2-7}$$

$$A + 1 = 1 \tag{2-8}$$

$$A + 0 = A \tag{2-9}$$

Remember that A, B, and C can take on only the value of 0 or 1.

AND Function

An AND function has two or more inputs and a single output, and it operates in accordance with the following rule: *The output of an AND gate assumes the 1 state if and only if all the inputs assume the 1 state*. The general equation for the AND function is given by $ABC \ldots N = Z$. A two-input AND function truth table is given by Table 2-10.

Inputs		Output
A	*B*	*Z*
0	0	0
0	1	0
1	0	0
1	1	1

Table 2-10. Two-Input AND Function Truth Table

An example of the use of AND logic in process control would be as follows: if the liquid level in a process tank is high and the inlet feed pump to the tank is running, design a logic circuit to open the tank outlet valve using electric relays. Figure 2-4 shows the relay ladder diagram to perform the required AND function in the application.

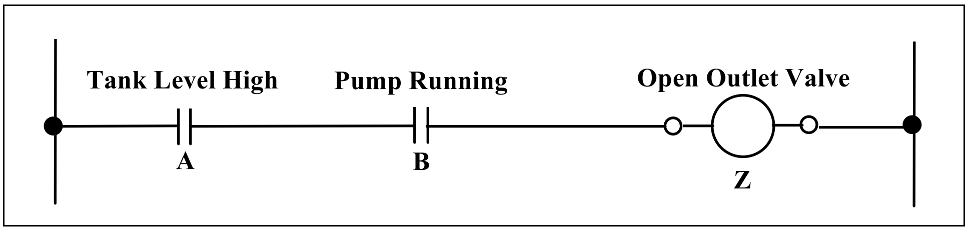


Figure 2-4. Relay-Based AND Logic Control Application

The logic expressions for the AND function are as follows:

$$A \bullet B \bullet C = (A \bullet B) \bullet C = A \bullet (B \bullet C) \qquad (2\text{-}10)$$

$$A \bullet B = B \bullet A \qquad (2\text{-}11)$$

$$A \bullet A = A \qquad (2\text{-}12)$$

$$A \bullet 1 = A \qquad (2\text{-}13)$$

$$A \bullet 0 = 0 \qquad (2\text{-}14)$$

These identities can be verified by reference to the definitions of the AND gate and by using a truth table for the AND gate.

For example, Equation 2-13 ($A \bullet 1 = A$) can be verified as follows: let $B = 1$ and tabulate $A \bullet B = Z$ or $A \bullet 1 = Z$ in a truth table as shown below.

Inputs		Output
A	*B*	*Z*
0	1	0
1	1	1

Note that $A = Z$ in the truth table above, so that $A \bullet 1 = A$ for both values (0 or 1) of A.

Some important auxiliary identities used in logic design are as follows:

$$A + A \bullet B = A \tag{2-15}$$

$$A + \overline{A} \bullet B = A + B \tag{2-16}$$

$$(A + B) \bullet (A + C) = A + B \bullet C \tag{2-17}$$

These identities are important because they help reduce the number of logic elements or gates required to implement a logic function.

NOR Function

Another common logic gate is the NOR gate, and Table 2-10 shows its operation for two inputs. It produces a logic 1 result if and only if all inputs are logic 0. Notice that the output of the NOR function is opposite from the output of an OR gate.

Inputs		Output
A	*B*	*Z*
0	0	1
0	1	0
1	0	0
1	1	0

Table 2-11. Two Input NOR Function Truth Table

NAND Function

Another logic operation of interest is the NAND gate; its operations are summarized in Table 2-11 for a two-input NAND function. Note that the output of the NAND function is the exact opposite of the AND function output. When both inputs to the NAND are 1, the output is 0. In all other configurations, the NAND function output is 1.

Inputs	Output
A B	*Z*
0 0	1
0 1	1
1 0	1
1 1	0

Table 2-12. Two Input NAND Function Truth Table

Logic Function Symbols

There are two common sets of symbols used in process control applications to represent logic function: graphic and ladder. The graphic symbols are generally used on engineering drawings to convey the overall logic plan for a discrete or batch control system and they are based on ISA-5.2-1976 - (R1992) Binary Logic Diagrams For Process Operations. The ladder logic symbols are used to describe a logic control plan if relays are being used to implement the control system or if programmable controller ladder logic is used to implement the control plan. Figure 2-5 compares these two sets of symbols for the most common logic functions encountered in logic control design.

Electrical Ladder Diagrams

Electrical ladder diagrams are a traditional method used to describe electrical controls. These ladder diagrams are encountered in the machine and process industries. They are called ladder diagrams because they look like ladders with vertical side rails and horizontal rungs. Each rung of a ladder is numbered so that we can easily cross-reference between sections on the drawing that describe the control.

We can appreciate the utility of ladder logic diagrams by investigating a simple control example. Figure 2-6 shows a typical process level control application. In this application, let us assume that the flow into the tank is random and we need to control the level in the tank by opening or closing the outlet valve (LV-1) based on the level sensed in the tank by a level switch (LSH-1). We will also provide the operator with a three-position hand, off and automatic (HOA) switch to manually turn the valve on or

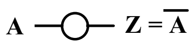
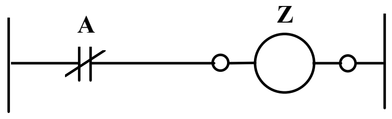
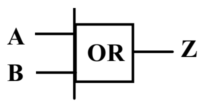
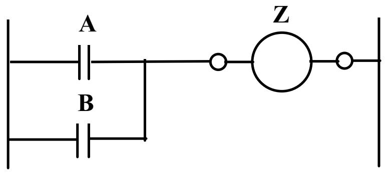
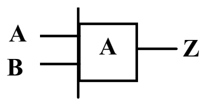
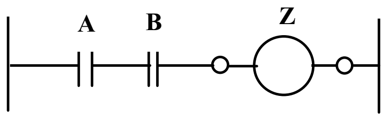
Logic Function	Logic Symbol	Ladder Symbol
NOT, $Z = \overline{A}$		
OR, $Z = A + B$		
AND, $Z = A \bullet B$		

Figure 2-5. Comparison of Logic Symbols

off or the option to select automatic control using the level switch to maintain the proper level in the tank.

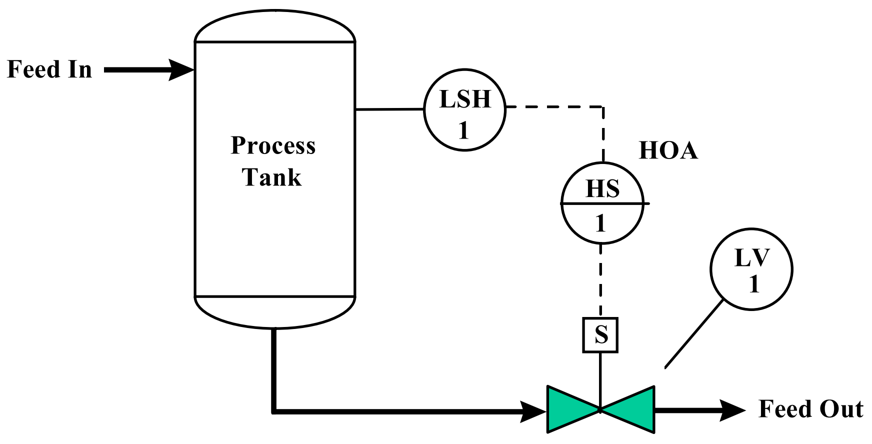


Figure 2-6. On-Off Control of Process Tank Level

The ladder diagram design for this application is shown in Figure 2-7. If the panel-mounted HOA switch is in the automatic position and the level switch is closed, the solenoid will be energized. This is a simple example of a logical AND function in process control. If the HOA switch is in the hand or manual position, the valve will also be turned on.

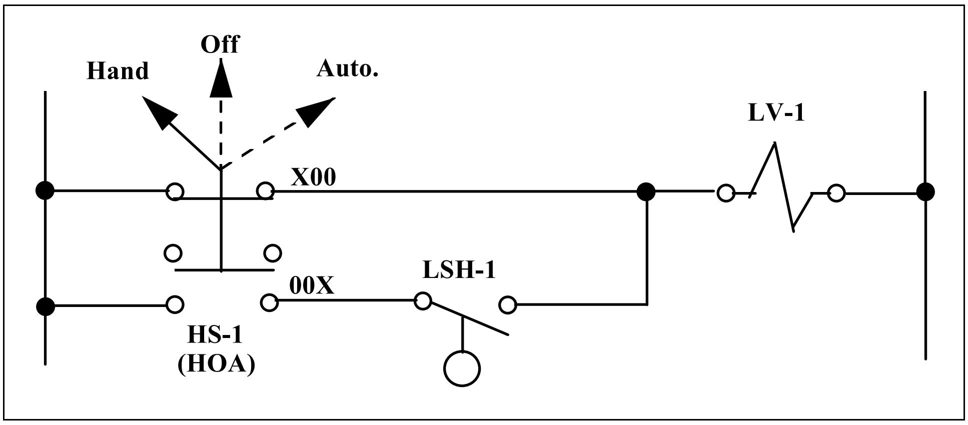


Figure 2-7. Ladder Diagram for Process Tank Level Control

If we designate the logic variables for the ladder diagram of Figure 2-7 as follows: *A* for hand position of HOA switch, *B* for automatic position of the HOA switch, C for the status of level switch LSH-1, and Z for the output to turn on solenoid valve LV-1, then the logic equation for the control circuit is Z = *A* + *BC.* We can implement this control circuit with logic gates as shown in Figure 2-8.

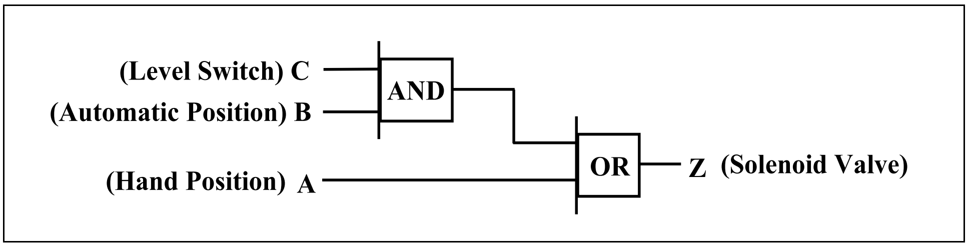


Figure 2-8. Logic Gate Implementation of Tank Level Control

Process Control Ladder Logic Application

A more complex process controls application might be to regulate the liquid level between two level switches, a level switch high (LSH) and a level switch low (LSL) mounted on a process tank. In this application, a pump supplying liquid to a tank is turned on and off to maintain the liquid level in the tank between the two level switches. The process is shown in Figure 2-9 and it uses steam at a regulated flow to boil down a liquid to produce a more concentrated solution, which is drained off periodically by opening a manual valve on the bottom of the tank. Note that a small diamond symbol is used to indicate that the pump is interlocked with the level switches on the tank.

Figure 2-9 is a typical example of a Process and Instrument Diagram (P&ID). In the measurement and control field, a standard set of symbols is used to prepare drawings of control systems for processes and mechanical systems. The symbols used in these drawings are based on the standard ISA-5.1-1984 - (R1992) Instrumentation Symbols and Identification.

In standard P&IDs, the process flow lines, such as process fluid and steam, are indicated with heavier solid lines than the lines that are used to represent the instrument. The instrument signal lines use special markings to indicate whether the signal is pneumatic, electric, hydraulic, and so on. Standard ISA-5.1-1984 - (R1992) lists the instrument line symbols that are used on P&IDs and Mechanical Flow Diagrams (MFDs). In Figure 2-9, a dashed line is used for the electrical control lines between various instruments and the process equipment.

A balloon symbol with an enclosed letter and number code is used to represent the instrumentation associated with the process or mechanical control loop. This letter and number combination is called an instrument identification or instrument *tag number*.

The first letter of the tag number is normally chosen so that it indicates the measured variable of the control loop. In the P&ID shown in Figure 2-9, *L* is the first letter in the tag number that is used for the instruments in the level switch on the process tank (i.e., Condenser). The succeeding letters are used to represent a readout or passive function or an output function, or the letter can be used as a modifier. For example, the balloon in Figure 2-9 marked *LSH-1* represents Level Switch High 1 and the balloon marked *LSL-1* is level switch low 1. The line across the center of the hand switches (HS-1 and HS-2) balloon symbol indicates that the controller is mounted on the front of a main control panel. No line indicates a field-mounted instrument, and two lines mean that the instrument is mounted in a local or field-mounted panel. Dashed lines indicate that the instrument is mounted inside the panel.

Normally, sequences of three- or four-digit numbers are used to identify each loop. In our simple process application (Figure 2-9), we used only loop numbers 1 and 2. Smaller processes use three-digit loop numbers; larger processes or complex manufacturing plants may require four or more digits to identify all the control loops.

Special marks or graphics are used to represent process equipment and instruments. For example, in our P&ID for the condenser liquid level control in Figure 2-9, the feed pump, the condenser and the manual drain valve uses special symbols that can be found in standard ISA-5.1-1984 - (R1992).

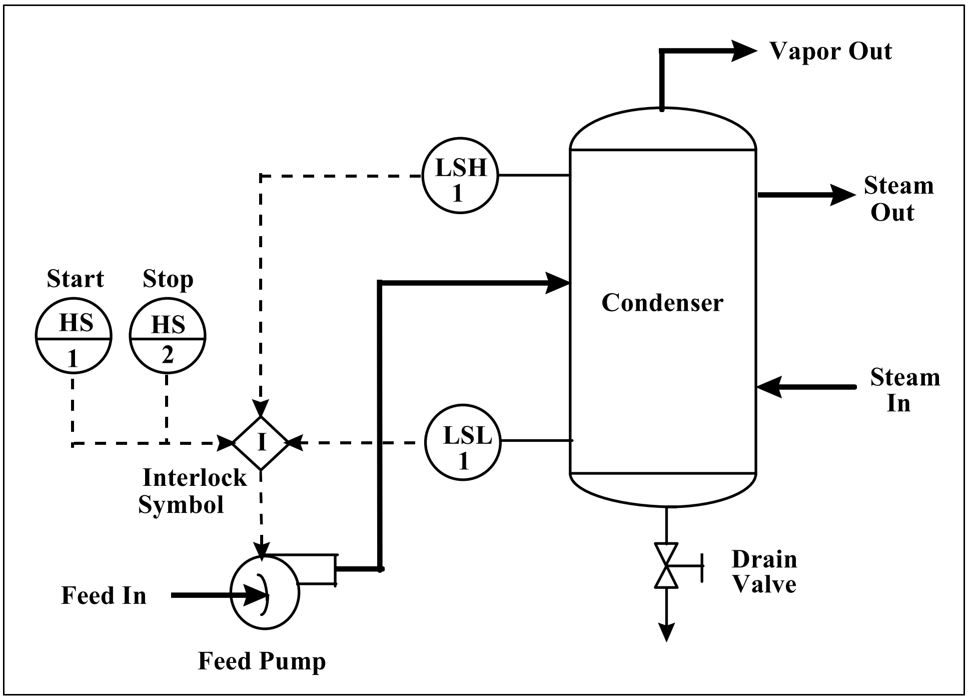


Figure 2-9. Pump Control of Condenser Liquid Level

The electrical ladder diagram used to control the feed pump and, hence, the liquid level in the process tank is shown in Figure 2-10. To explain the logic of the control system, we will assume the tank is empty, the low level switch is closed, and the high level switch is closed. When a level switch is activated, the normally open contacts are closed and the normally closed contacts are opened. To start the control system, the operator depresses the start pushbutton (HS1). This energizes control relay (CR1), which seals in the start pushbutton with the first set of contacts, denoted as CR1(1) on the ladder diagram. At the same time, the second control relay (CR2) is energized in rung 3 through contacts on LSH-1, LSL-1, and CR1. This turns on the pump motor starter MS1. The first set of contacts on CR2 (1) is used to seal in the low-level switch contacts so that when the level in the tank rises above the position of the low-level switch on the tank, the pump will stay on until the liquid level reaches the high-level switch. After the high-level switch is activated, the pump will be turned off. The system will now cycle on and off between the high and low levels until the operator depresses the stop pushbutton (HS2).

Mechanical System Ladder Diagram Control Application

Another typical ladder diagram application consists of controlling the mechanical conveyor system shown in Figure 2-11. In this application, conveyor 2 receives production parts from conveyor 1 so that conveyor 2

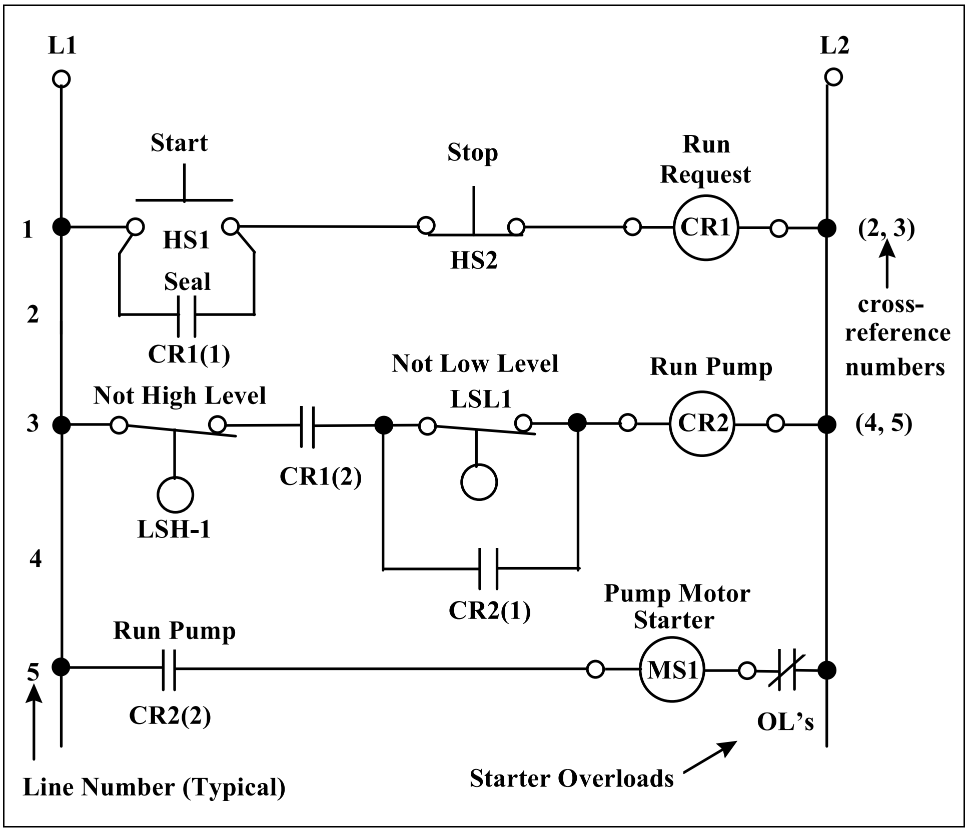


Figure 2-10. Ladder Diagram for Pump Control

must be running before conveyor 1 is started. Figure 2-11 is called a Mechanical Flow Diagram (MFD), which shows the mechanical equipment and controls for the conveyor system.

The MFD shows five pushbuttons (PB1 through 5) used to start and stop the two conveyors. Emergency stop (e-stop) buttons are required in all mechanical control systems to prevent injury to personnel or damage to the equipment. These pushbuttons are placed at strategic locations near the mechanical equipment and within easy access of the operator. Only one pushbutton is shown in Figure 2-11 for simplicity, but there is generally more than one button. The normally closed contacts of these e-stops are wired in series so that pushing any e-stop button removes electrical power from the system. The normally closed contacts are used to guarantee that electrical power will be removed if a wire is broken or removed from the e-stop circuit.

A key design objective of the control system is to prevent material from the first conveyor from piling up on the second conveyor, if the second

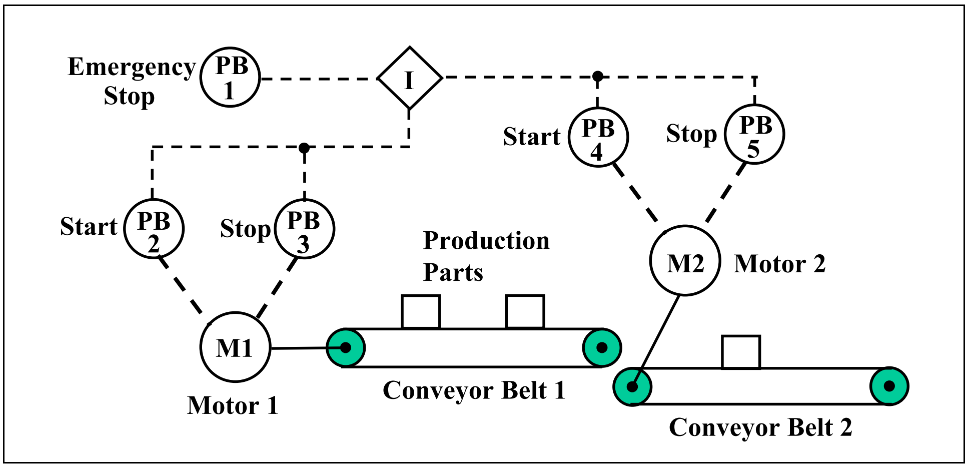


Figure 2-11. MFD for Conveyor Control System

conveyor is stopped for any reason. The ladder diagram for the conveyor system is shown in Figure 2-12.

This logic system will not allow the first conveyor to operate unless the second conveyor has been started or it is running. The system operates as follows; if the operator depresses start button 4 (PB4), electric current will flow to the magnetic motor starter (MS2) to energize the coil and start the constant speed motor for conveyor 2. When motor starter (MS2) energizes, its auxiliary contacts (M2) will close and seal in its start button (PB4). When start pushbutton PB2 is depressed, electric current will flow to motor starter (MS1) through e-stop (PB1), M2, PB3, and PB2 to energize the starter coil and start the constant speed motor for conveyor 1 so that conveyor 1 cannot be started unless conveyor 2 is started and running. If the overloads on starter coil M2 are activated, M2 will stop and the auxiliary contacts of M2 will stop conveyor 1. Also depressing any one of the stop buttons (PB1, 2, or 3) will turn off only conveyor 1, but depressing the e-stop (PB1) or stop 2 buttons (PB3) will turn both conveyors off.

EXERCISES

2.1 Convert the binary number 1010 to a decimal number.

2.2 Convert the binary number 10011 to a decimal number.

2.3 Find the octal equivalent of the binary number 10101011.

2.4 Find the octal equivalent of the binary number 101111101000.

2.5 Convert the decimal number 33_{10} to its octal equivalent.

2.6 Convert the decimal number 451_{10} to its octal equivalent.

2.7 Convert the hexadecimal number 47_{16} to its decimal equivalents.

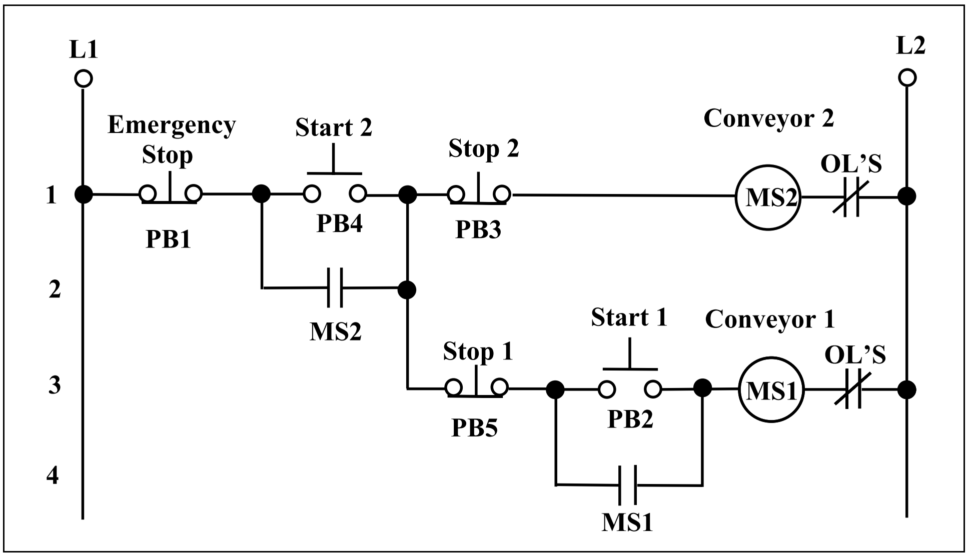


Figure 2-12. Ladder Diagram for Conveyor System

2.8 Convert the hexadecimal number 157_{16} to its decimal equivalents.

2.9 Convert the decimal number 56_{10} to its hexadecimal equivalent.

2.10 Convert the decimal number 37_{10} to its equivalent BCD code.

2.11 Convert the decimal number 270_{10} to its equivalent BCD code.

2.12 Express the words Level Low using ASCII code. Use Hex notation.

2.13 Verify the logic identity $A + 1 = 1$ using a two input OR truth table.

2.14 Write the logic equation for the start/stop control logic in lines 1 and 2 of the ladder diagram for the pump control application shown in Figure 2-10.

2.15 Write the logic equation for the control logic in lines 3 and 4 of the ladder diagram for the pump control shown in Figure 2-10.

2.16 Draw a Logic Gate circuit for the standard Start/Stop circuit used in the pump control application shown in Figure 2-10.

2.17 Describe the control actions that will stop conveyor one (shown in Figure 2-11).

3 Electrical and Electronic Fundamentals

Introduction

The design of programmable control systems requires a thorough knowledge of the basic principles of electricity and electronics. This chapter will discuss the fundamentals of electricity and magnetism and then examine the electrical and electronic circuits that are commonly encountered in process control. We will also discuss the operation and purpose of electro-magnetic and electronic devices, such as transformers, power supplies, electromagnetic relays, solenoid valves, contactors, control switches, and indicating lights.

Fundamentals of Electricity

Electricity is a fundamental force in nature that can produce heat, motion, light, and many other physical effects. The force is an attraction or repulsion between electric charges called *electrons* and *protons*. An electron is a small atomic particle having a negative electric charge. A proton is a basic atomic particle with a positive charge. It is the arrangement of electrons and protons that determines the electrical characteristics of all substances. As an example, this page of paper contains electrons and protons, but you cannot detect any evidence of electricity because the number of electrons equals the number of protons. In this case, the opposite electrical charges cancel, making the paper electrically neutral.

If we want to use the electrical forces that are associated with positive and negative charges, work must be performed that separates the electrons and protons. For example, an electric battery can do such work because its chemical energy separates electrical charges to produce an excess of electrons at its negative terminal and an excess of protons at its positive terminal.

The basic terms encountered in electricity are electric *charge* (*Q*), *current* (*I*), *voltage* (*V*), and *resistance* (*R*). For most common applications of electricity a charge of a very large number of electrons and protons is required. It is therefore convenient to define a practical unit called the *coulomb* (*C*) that is equal to the charge of 6.25×10^{18} electrons or protons. The symbol for electric charge is *Q*, which stands for *quantity of charge*, and a charge of 6.25×10^{18} electrons or protons is stated as $Q = 1$ *C*. Voltage is the potential difference or force produced by the differences in opposite electrical charges. Fundamentally, the volt is a measure of the work required to move an electric charge. When 1 *joule* of work is required to move 1 C between two points, the potential difference is 1 V.

When the potential difference between two different charges forces a third charge to move, the charge in motion is called *electric current*. If the charges move past a given point at the rate of 1 C per second, the amount of current is defined as 1 *ampere* (*A*). In equation form, $I = dQ/dt$, where *I* is the instantaneous current in amperes and *dQ* is the differential amount charge in coulombs passing a given point during the time period (*dt*) in seconds. If the current flow is constant, it is simply given by the following equation:

$$I = \frac{Q}{t} \tag{3-1}$$

where *Q* is the amount of current in coulombs flowing past a given point in *t* seconds.

Example 3-1 shows how to calculate current flow.

EXAMPLE 3-1

Problem: A steady flow of 12 C of charge passes a given point in an electrical conductor every 3 seconds. What is the current flow?

Solution: Since constant current flow is defined as the amount of charge (Q) that passes a given point per period of time (t) in seconds, we have

$$I = \frac{Q}{t} = \frac{12 \text{ coulombs}}{3 \text{ seconds}} = 4 \text{ amps}$$

The fact that a conductor carrying electric current can become hot is evidence that the work done by the applied voltage in producing current must be meeting some form of opposition. This opposition, which limits current flow, is called *resistance*.

Conductivity, Resistivity, and Ohm's Law

An important physical property of some material is *conductivity*, that is, the ability to pass electric current. Suppose we have an electric wire (conductor) of length L and cross-sectional area A, and we apply a voltage V between the ends of the wire. If V is in volts and L is in meters, we can define the voltage gradient (E), as follows:

$$E = \frac{V}{L} \tag{3-2}$$

Now, if a current I in amperes flows through a wire of area A in meters squared (m^2), we can define the current density J as follows:

$$J = \frac{I}{A} \tag{3-3}$$

The conductivity C is defined as the current density per unit voltage gradient E or, in equation form, we have

$$C = \frac{J}{E} \tag{3-4}$$

Using the definitions for current density J and voltage gradient E, we obtain conductivity in a different form:

$$C = \frac{I/A}{V/L} \tag{3-5}$$

Resistivity (r) is defined as the inverse of conductivity, or

$$r = \frac{1}{C} \tag{3-6}$$

The fact that resistivity is a natural property of certain materials leads to the basic principle of electricity called *Ohm's law*.

Consider a wire of length *L* and cross-sectional area *A*. If the wire has resistivity, *r*, then its resistance, *R*, is

$$R = r\frac{L}{A} \tag{3-7}$$

The unit of resistance is the ohm, which is denoted by the Greek letter omega, Ω. Since resistivity, *r*, is the reciprocal of conductivity, we obtain the following:

$$r = \frac{V/L}{I/A} \tag{3-8}$$

When Equation 3-8 is substituted into Equation 3-7, we obtain the following:

$$R = \frac{V}{I} \tag{3-9}$$

This relationship in its three forms—$R = V/I$ or I = V/R or $V = IR$—is called Ohm's law. It assumes that the resistance of the material that is used to carry the current flow will have a linear relationship between the voltage applied and the current flow. That is, if the voltage across the resistance is doubled, the current through it also doubles. The resistance of materials such as carbon, aluminum, copper, silver, gold, and iron is linear and follows Ohm's law. Carbon is the material most commonly used to manufacture a device that has fixed resistance. This device is called a *resistor*.

Wire Resistance

To make it possible to compare the resistance and size of one conductor with another, the U.S. established a standard or unit size for conductors. The standard unit of measurement for the diameter of a circular wire or conductor is the *mil* (0.001 inch), and the standard unit of wire length is the *foot*. The standard unit of wire size in the U.S. is the *mil-foot*. That is, a wire is said to have a unit size if it has a diameter of 1 mil and a length of 1 foot.

The *circular mil* is the standard unit of cross-sectional area for wire used in U.S. wire-sizing tables. Because the diameter of circular wire is normally only a small fraction of an inch, it is convenient to express these diameters in mils, to avoid the use of decimals. For example, the diameter of a 0.010-inch conductor is expressed as 10 mils instead of 0.010 inch. The circular mil area is defined as the square of the mil diameter of a conductor, or area (cmil) = d^2 (mil^2). Example 3-2 illustrates how to calculate conductor area in circular mils.

EXAMPLE 3-2

Problem: Calculate the area in circular mils of a conductor with a diameter of 0.002 inch.

Solution: First, we must convert the diameter in inches to a diameter in mils. Since we defined 0.001 in. = 1 mil, this means that 0.002 in. = 2 mils, so the circular mil area is (2 mil)^2 or 4 cmil.

A *circular-mil per foot* is a unit conductor 1 foot in length that has a cross-sectional area of 1 circular mil. The unit cmil per foot is useful for comparing the resistivity of various conductors. Table 3-1 gives the specific resistance (r) in cmil-ohms per foot of some common solid metals at 20°C.

Conductors that are used to carry electric current are normally manufactured out of copper because of its low resistance and relatively low cost. We can use the specific resistance (r) given in Table 3-1 to find the resistance of a conductor of length (L) in feet and cross-sectional area (A) in cmil by using Equation 3-6, $R=rL/A$.

Material	*r*, resistivity, cmil-Ω/ft
Silver	9.8
Copper (drawn)	10.37
Gold	14.70
Aluminum	17.02
Tungsten	33.20
Brass	42.10
Steel	95.80

Table 3-1. Specific Resistivity

Example 3-3 illustrates how to find the resistance of copper wire with a fixed conductor area in circular mils.

The equation for the resistance of a conductor, $R = rL/A$, can be used in many applications. For example, if you know R, r, and A, you can determine the length by a simple mathematical transformation of the equation for the resistance of a conductor. A typical application is locating a problem ground point in a telephone line. To find ground faults, the

EXAMPLE 3-3

Problem: Find the resistance of 1,000 feet of copper (drawn) wire that has a cross-sectional area of 10,370 cmil and a wire temperature of 20°C.

Solution: From Table 3-1, the specific resistance of copper wire is 10.37 cmil-ohms/ft. By inserting the known values into Equation 3-6, the resistance is determined as follows:

$$R = r\frac{L}{A}$$

$$R = (10.37\ \text{cmil - }\Omega/\text{ft})\left(\frac{1000\ \text{ft}}{10{,}400\ \text{cmil}}\right)$$

$$R = 1\Omega$$

telephone company uses specially designed test equipment. This equipment operates on the principle that the resistance of a conductor varies in direct relation to distance, so the distance between a test point and a fault can be measured accurately with properly designed equipment.

Example 3-4 illustrates how to find the distance to a short to ground in a telephone line.

Wire Gauge Sizes

Electrical conductors are manufactured in sizes that are numbered according to a system known as the American Wire Gauge (AWG). As Table 3-2 shows, the wire diameters become smaller as the gauge numbers increase, and the resistance per one thousand feet increases as the wire diameter decreases. The largest wire size listed in the table is "08," and the smallest is "24." This is the normal range of wire sizes typically encountered in process control applications. The complete AWG table goes from 0000 to 40; the larger and smaller sizes not listed in Table 3-2 are manufactured but are not commonly encountered in process control.

EXAMPLE 3-4

Problem: The resistance to ground on a faulty underground telephone line is 5Ω. Calculate the distance to the point where the wire is shorted to ground if the line is a copper conductor that has a cross-sectional area of 1,020 cmil and the ambient temperature of the conductor is 20°C.

Solution: To calculate the distance to the point where the wire is shorted to ground, we transform Equation 3-6 as follows:

$$L = \frac{RA}{r}$$

Since R = 5Ω, A = 1,020 cmils, and r = 10.37 cmil-Ω/ft, we have

$$L = \frac{(5\Omega)(1020 cmil)}{10.37 cmil - \Omega / ft} = 492 ft$$

AWG Number	Diameter, mil	Area cmil	Ω/1000 ft at 25°C	Ω/1000 ft at 65°C
08	128.0	16500	0.641	0.739
10	102.0	10400	1.020	1.180
12	81.0	6530	1.620	1.870
14	64.0	4110	2.58	2.97
16	51.0	2580	4.09	4.73
18	40.0	1620	6.51	7.51
20	32.0	1020	10.4	11.9
22	25.3	642	16.5	19.0
24	20.1	404	26.2	30.2

Table 3-2. American Wire Gauge for Copper Wire

Example 3-5 shows how to calculate the resistance of a typical conductor.

Direct and Alternating Current

Basically, two types of voltage signals are encountered in process control and measurement: direct current (DC) and alternating current (AC). In direct current, the flow of charges is in just one direction. A battery is one example of a DC power source. Figure 3-1a shows a graph of a DC voltage over time, and Figure 3-1b shows the symbol for a battery.

EXAMPLE 3-5

Problem: Determine the resistance of 2,500 feet of 14 AWG copper wire. Assume the wire temperature is 25°C.

Solution: Using Table 3-2, we see that 14 AWG wire has a resistance of 2.58Ω per 1,000 ft at 25°C. So the resistance of 2,500 ft is calculated as follows:

$$R = (2.58\Omega/1000 \text{ ft})(2500 \text{ ft}) = 6.5\Omega$$

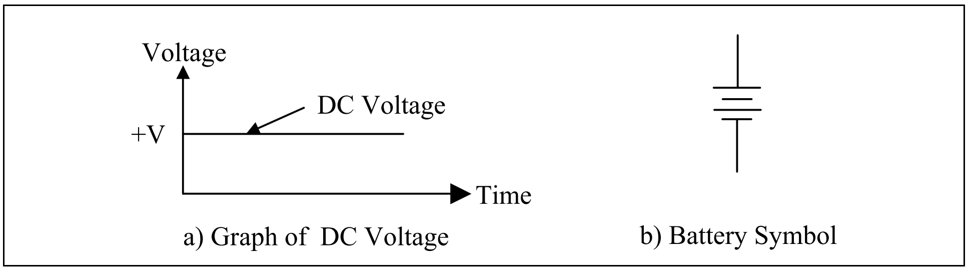


Figure 3-1. DC Voltage Graph and Symbol

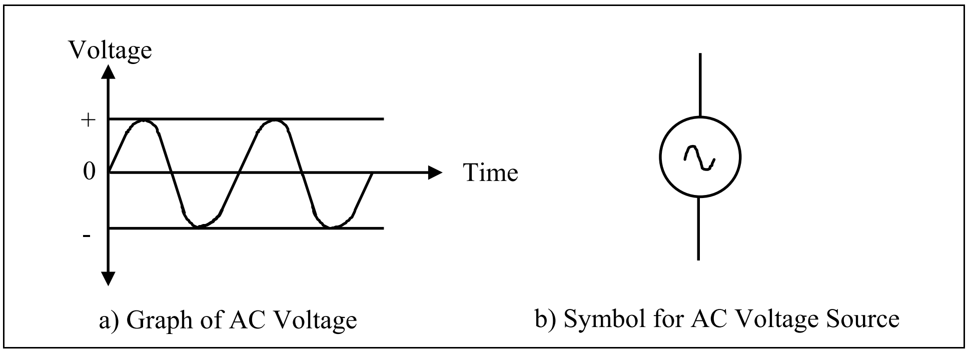


Figure 3-2. AC Voltage Graph and Symbol

An alternating voltage source periodically reverses its polarity. Therefore, the resulting current flow in a closed circuit will reverse direction periodically. Figure 3-2a shows a sine-wave example of an AC voltage signal over time. Figure 3-2b shows the schematic symbol for an AC power source.

The 60-cycle AC power that is used in homes and industry in the U.S. is a common example of AC power. The term *60 cycle* means that the voltage polarity and current direction go through 60 reversals or changes per second. The signal is said to have a frequency of 60 cycles per second. The unit for 1 cycle per second (cps) is called 1 hertz (Hz). Therefore, a 60-cycle-per-second signal has a frequency of 60 Hz.

Series Resistance Circuits

The components in a circuit form a series circuit when they are connected in successive order with the end of a component that is joined to the end of the next element. Figure 3-3 shows an example of a series circuit.

In this circuit, the current (I) flows from the negative terminal of the battery through the two resistors (R_1 and R_2) and back to the positive terminal. According to Ohm's law, the amount of current (I) flowing between two points in a circuit equals the potential difference (V) divided by the resistance (R) between these points. If V_1 is the voltage drop across R_1, V_2 is defined as the voltage drop across R_2, and the current (I) flows through both R_1 and R_2, then from Ohm's law we obtain the following:

$$V_1 = IR_1 \text{ and } V_2 = IR_2$$

so that

$$V_t = IR_1 + IR_2$$

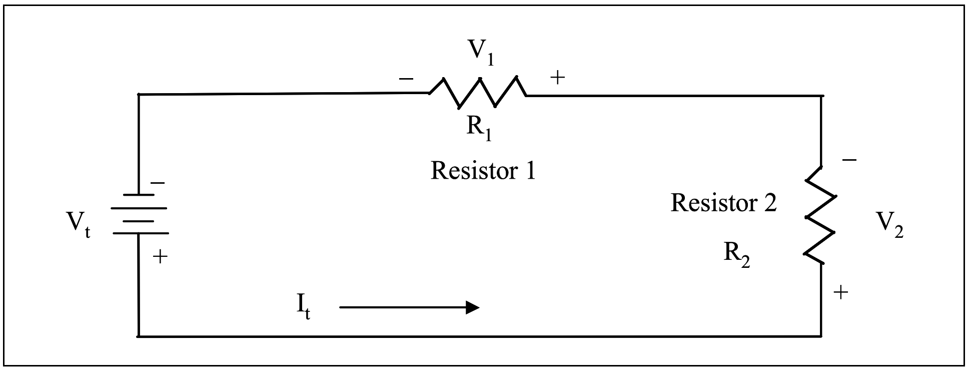


Figure 3-3. Series Resistance Circuit

If we divided both sides of this equation by the current, I, in the series circuit, we obtain the following:

$$V_t/I = R_1 + R_2$$

Ohm's law defines the total resistance of the series circuit as the total voltage applied divided by the current in the circuit or $R_t = V_t/I$. As a result, the total resistance of the circuit is given by the following equation:

$$R_t = R_1 + R_2$$

We can derive a more general equation for any number of resistors in series by using the classical law of conservation of energy. According to this law, the energy or power supplied to a series circuit must equal the power dissipated in the resistors in the circuit. Thus,

$$P_t = P_1 + P_2 + P_3 \ldots\ldots P_n$$

since power in a resistive circuit is given by $P = I^2R$, we obtain the following:

$$I^2R_t = I^2R_1 + I^2R_2 + I^2R_3 + \ldots\ldots + I^2R_n$$

If we divide this equation by I^2, we obtain the series resistance formula for a circuit with n resistors:

$$R_t = R_1 + R_2 + R_3 + \ldots\ldots + R_n \qquad (3\text{-}10)$$

Example 3-6 illustrates calculations for a typical series DC circuit.

EXAMPLE 3-6

Problem: Assume that the battery voltage in the series circuit shown in Figure 3-3 is 6 VDC and that resistor R_1 = 1kΩ and resistor R_2 = 2kΩ, where k is 1,000 or 10^3. Find the total current flow (I_t) in the circuit and the voltage across R_1 and R_2.

Solution: To calculate the circuit current I_t, first find the total circuit resistance R_t using Equation 3-10:

$$R_t = R_1 + R_2$$

$$R_t = 1\text{k}\Omega + 2\text{k}\Omega$$

$$R_t = 3\text{k}\Omega$$

Now, according to Ohm's law, we have:

$$I_t = V_t / R_t$$

$$I_t = 6\text{V}/3\text{k}\Omega = 2\text{ mA}$$

The voltage across R_t (i.e., V_1) is given by

$$V_1 = R_1 I_t$$

$$V_1 = (1\text{k}\Omega)(2\text{ mA}) = 2\text{ V}$$

Thus, the voltage across R_2 (i.e., V_2) is obtained as follows:

$$V_2 = R_2 I_t$$

$$V_2 = (2\text{k}\Omega)\ (2\text{ mA}) = 4\text{ V}$$

Parallel Resistance Circuits

When two or more components are connected across a power source, they form a parallel circuit. Each parallel path is called a *branch,* and each branch has its own current. In other words, parallel circuits have one common voltage across all the branches but individual currents in each branch. These characteristics contrast with series circuits, which have one common current but individual voltage drops.

Figure 3-4 shows a parallel resistance circuit with two resistors across a battery. You can find the total resistance (R_t) across the power supply by dividing the voltage across the parallel resistance by the total current into the two branches. In the circuit of Figure 3-4, total current is given by $I_t = I_1 + I_2$, where the current in branch 1 is given by $I_1 = V_t/R_1$ and the current in branch 2 is given by $I_2 = V_t/R_2$. Thus, $I_t = V_t/R_1 + V_t/R_2$. Since according to Ohm's law total current is given by $I_t = V_t/R_t$, we obtain Equation 3-11 for total resistance, R_t, of two resistor in parallel:

$$\frac{1}{R_t} = \frac{1}{R_1} + \frac{1}{R_2} \tag{3-11}$$

Transforming this equation into a more useful form, we obtain the following:

$$R_t = \frac{R_1 \, x \, R_2}{R_1 + R_2} \tag{3-12}$$

We can derive the general reciprocal resistance formula for any number of resistors in parallel from the fact that the total current I_t is the sum of all the branch currents, or

$$I_t = I_1 + I_2 + I_3 + \ldots + I_n$$

Since current flow is defined *as* $I = dQ/dt$, this is simply the law of electrical charge conservation:

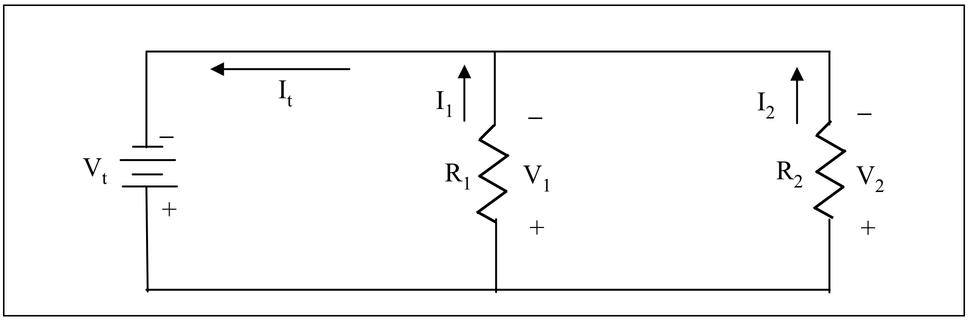


Figure 3-4. Parallel Resistance Circuit

$$\frac{dQ_t}{dt} = \frac{dQ_1}{dt} + \frac{dQ_2}{dt} + \frac{dQ_3}{dt} + \ldots\ldots + \frac{dQ_n}{dt}$$

This law implies that, since no charge accumulates at any point in the circuit, the differential charge, dQ, from the power source in the differential time period, dt, must appear as charges dQ_1, dQ_2, $dQ_3 \ldots dQ_n$ through the resistors R_1, R_2, $R_3 \ldots R_n$ at the same time. Since the voltage across each branch is the applied voltage V_t and $I = V/R$, we obtain the following:

$$\frac{V_t}{R_t} = \frac{V_t}{R_1} + \frac{V_t}{R_2} + \frac{V_t}{R_3} + \ldots\ldots + \frac{V_t}{R_n}$$

If we divide both sides of this equation by V_t, we obtain the total resistance of n resistors in parallel:

$$\frac{1}{R_t} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \ldots\ldots + \frac{1}{R_n} \tag{3-13}$$

To illustrate the basic concepts of parallel resistive circuits, let's look at Example 3-7.

EXAMPLE 3-7

Problem: Assume that the voltage for the parallel circuit shown in Figure 3-4 is 12 V and we need to find the currents I_t, I_1, and I_2 and the parallel resistance R_t, given that R_1 = 30 kΩ and R_2 = 30 kΩ.

(continued)

EXAMPLE 3-7 continued

Solution: We can find the parallel circuit resistance R_t by using Equation 3-12:

$$R_t = \frac{R_1 \ x \ R_2}{R_1 + R_2}$$

$$R_t = \frac{(30\text{k})(30\text{k})}{30\text{k} + 30\text{k}} \Omega = 15\text{k}\Omega$$

The total current flow is given by the following equation:

$$I_t = \frac{V_t}{R_t} = \frac{12\text{V}}{15\text{k}\Omega} = 0.8 \text{ mA}$$

The current branch 1 (I_1) is obtained as follows:

$$I_1 = \frac{V_t}{R_1} = \frac{12\text{V}}{30\text{k}\Omega} = 0.4 \text{ mA}$$

Since $I_t = I_1 + I_2$, the current flow in branch 2 is given by

$$I_2 = I_t - I_1 = 0.8 \text{ mA} - 0.4 \text{ mA} = 0.4 \text{ mA}$$

Wheatstone Bridge Circuit

The Wheatstone bridge circuit, named for English physicist and inventor Sir Charles Wheatstone (1802-1875), was one of the first electrical instruments invented to accurately measure resistance. A typical Wheatstone bridge circuit is shown in Figure 3-5. The circuit has two parallel resistance branches with two series resistors in each branch and a galvanometer (G) connected across the branches. The galvanometer is a very sensitive electric-current-measuring instrument that is connected across the bridge between points "a" and "b" when switch 2 is closed. If these points are at the same potential, the meter will not deflect. The purpose of the circuit is to balance the voltage drops across the two parallel branches to obtain 0 V across the meter and zero current through the meter. In the Wheatstone bridge, the unknown resistance R_x is balanced against a standard accurate resistor R_s to make possible precise measurement of resistance.

In the circuit shown in Figure 3-5, the switch S_1 is closed to apply the voltage V_t to the four resistors in the bridge, and switch S_2 is closed to connect the galvanometer (G) across the bridge. To balance the circuit, the

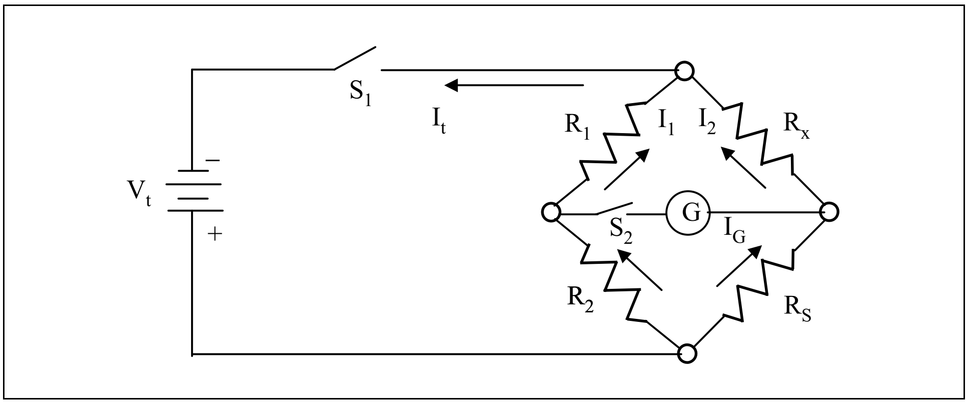


Figure 3-5. Wheatstone Bridge Circuit

value of R_x is varied until a zero reading is obtained on the meter with switch S_2 closed.

A typical application of the Wheatstone bridge circuit is to replace the unknown resistor R_x with a resistance temperature device (RTD). An RTD is a device that varies its resistance in response to a change in temperature so the balancing resistor dial can be calibrated to read out a process temperature. In a typical application, an electronic sensor that is designed to convert the ambient temperature into resistance is connected to a Wheatstone bridge circuit to display the temperature.

When the bridge circuit is balanced (i.e., there is no current through the galvanometer G), the bridge circuit can be analyzed as two series resistance strings in parallel. The equal voltage ratios in the two branches of the Wheatstone bridge can be stated as follows:

$$\frac{I_2 R_x}{I_2 R_s} = \frac{I_1 R_1}{I_1 R_2} \tag{3-14}$$

Note that I_1 and I_2 can be canceled out in the previous equation, so we can invert R_s to the right side of the equation to find R_x as follows:

$$R_x = R_s \frac{R_1}{R_2} \tag{3-15}$$

To illustrate the use of a Wheatstone bridge, let's consider Example 3-8.

EXAMPLE 3-8

Problem: Assume a Wheatstone bridge with R_1 = 1 kΩ and R_2 = 10kΩ. If R_s is adjusted to read 50Ω when the bridge is balanced (i.e., the galvanometer reading is at zero), calculate the value of R_x.

Solution: We can use Equation 3-15 to determine R_x:

$$R_x = R_s \frac{R_1}{R_2}$$

$$R_x = 50\Omega \frac{1\text{K}\Omega}{10\text{K}\Omega} = 5\Omega$$

Instrumentation Current Loop

The brief discussion on resistive circuits earlier in this chapter served as a starting-off point for this section on instrumentation current loops. The DC current loop shown in Figure 3-6 is used extensively in the instrumentation field to transmit process variables to indicators and controllers. It is also used to send control signals to field devices to manipulate process variables such as temperature, level, and flow. The standard current range used in these loops is 4 to 20 mA, and this value is normally converted to 1 to 5 VDC by a 250 Ω resistor at the input to controllers and indicators. These instruments are normally high-input-impedance (Zin > 10 MΩ) electronic devices that draw virtually no current from the instrument loop.

There are two main advantages to using the 4- to 20-mA current loop. First, only two wires are required for each remotely mounted field transmitter, so a cost savings is realized on both labor and wire when installing field devices. The second advantage is that the current loop is not affected by electrical noise or changes in lead wire resistance caused by temperature changes.

Selecting Wire Size

You must consider several factors when selecting the wire size for a programmable controller application. One factor is the permissible power loss ($P = I^2R$) in the electrical line. This power loss is electrical energy being converted into heat. If the heat produced is excessive, the conductors or system components may be damaged. Using large-diameter (low-wire-gauge) conductors will reduce the circuit resistance and, therefore, the power loss. However, larger-diameter conductors are more

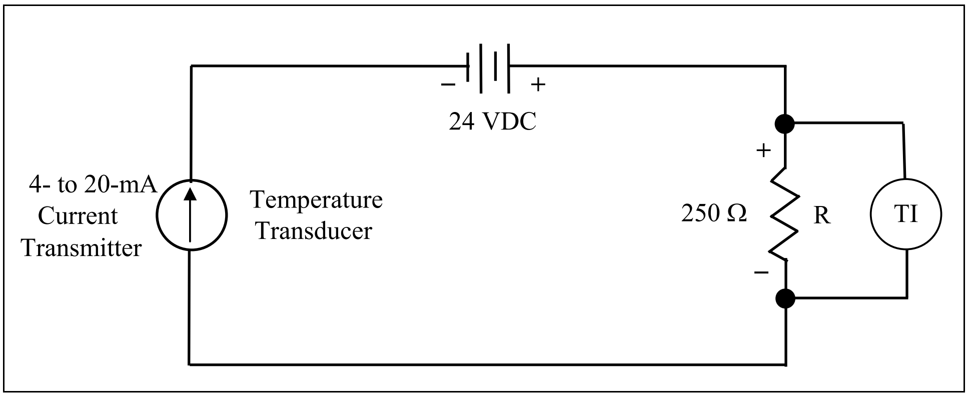


Figure 3-6. Typical 4- to 20-mA Current Loop

expensive than smaller ones and they are more costly to install, so you will need to make design calculations to select the proper wire size.

A second factor when selecting wire is the resistance of the wires in the circuit. For example, let's assume that we are sending a full-range 20-mA DC signal from a field instrument to a programmable controller input module that is located 2,000 feet away, as shown in Figure 3-7.

Assuming that we are using 20 AWG wire, we can easily calculate the wire resistance (R_w). Using data from Table 3-2, we see that 20 AWG wire has a resistance of 10.4Ω per 1,000 ft at 25°C. So the resistance for the 4,000 feet of wire is calculated as follows:

$$R_W = (10.4\Omega/1000\text{ ft})(4000\text{ ft}) = 41.6\Omega$$

This resistance increases the load on the 24 VDC power supply that is used to drive the current loop. You can calculate the voltage drop V_w in the wire when the field instrument is sending 20 mA of current to the programmable controller analog input module by using Ohm's law:

$$V_W = IR_W = (20\text{ mA})(41.6\Omega) = 0.832\text{ V}$$

A third factor when selecting wire is the current-carrying capacity of the conductor. When current flows in a wire, heat is generated. The temperature of the wire will rise until the heat that is radiated away, or otherwise dissipated, is equal to the heat generated in the conductor. If the conductor is insulated, the heat produced is not dissipated so quickly as it would if the conductor were not insulated. So, to protect the insulation from excessive heat, you must keep the current flowing in the wire below a certain value. Rubber insulation will start to deteriorate at relatively low temperatures. Teflon™ and certain plastic insulations retain their

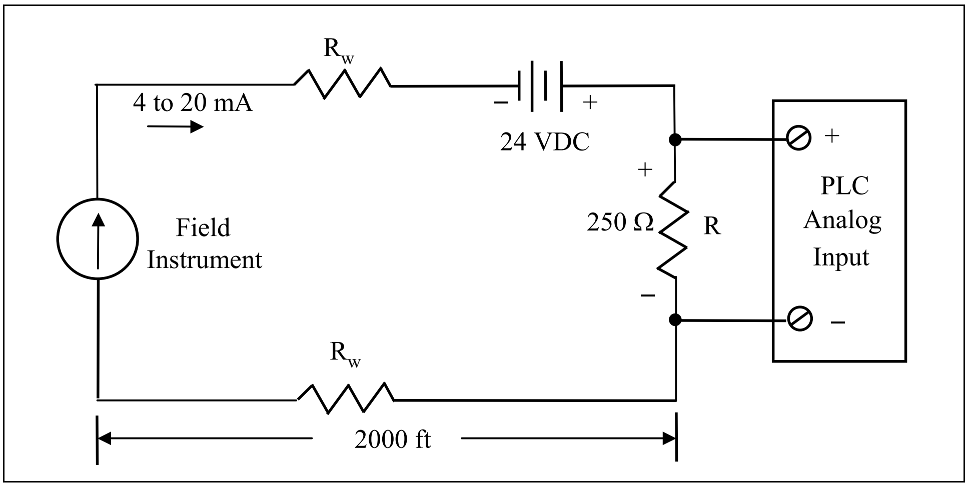


Figure 3-7. Voltage Drop in Instrument Loop

insulating properties at higher temperatures, and insulations such as asbestos are effective at still higher temperatures.

You could install electrical cables in locations where the ambient temperature is relatively high. In these cases, the heat produced by external sources adds to the total heating of the electrical conductor. When designing a programmable controller system, you must therefore make allowances for the ambient heat sources encountered in industrial environments. The maximum allowable operating temperature for insulated wires and cables is specified in manufacturers' electrical design tables.

Table 3-3 gives an example of the maximum allowable current-carrying capacities of copper conductor with three different types of insulation. The current ratings listed in the table are those permitted by the National Electrical Code. This table can be used to determine the safe and proper wire size for any electrical wiring application. Example 3-9 will help to illustrate a typical wire-sizing calculation.

Using Table 3-3, we can see that the main power conductors (hot, neutral, and ground) must have a minimum size of 14 AWG.

Fundamentals of Magnetism

Magnetism, like electricity is a fundamental force in nature. As we discussed earlier, electrical effects exist in two forms, voltage and current. Separated electric charges or voltage have the potential to do mechanical work in attracting or repelling charges. Similarly, a changing electric current has an associated magnetic field that can do work of attraction or

EXAMPLE 3-9

Problem: Calculate the proper wire size for the 120-VAC feed to the programmable controller system shown in Figure 3-8, assuming a maximum ambient temperature of 60°C.

Solution: The total AC feed current is the sum of all the branch currents, or

$I_t = I_1 + I_2 + I_3 + I_4$

$I_t = 2\text{ A} + 8\text{ A} + 5\text{ A} + 5\text{ A}$

$I_t = 20\text{ A}$

Conductor Size (AWG)	60°C (140°F) Types: TW, UF	75°C (167°F) Types: FEPW, RH, RHW, THHW, THW, THWN, XHHW, USE, ZW	85°C (185°F) Type: V18
14	20	20	25
12	25	25	30
10	30	35	40
8	40	50	55
6	55	65	70

Table 3-3. Ampacities of Insulated Copper Conductor National Electrical Code

repulsion. Some materials made of iron, nickel, and cobalt can concentrate their magnetic effects on opposite ends, where the magnetic material meets a nonmagnetic medium such as air.

The term magnetism is derived from the iron oxide mineral, *magnetite*, a naturally occurring magnet first used in navigational compasses. Ferromagnetism refers specifically to the magnetic properties of iron. There are two types of magnets: permanent and temporary. The natural magnet, *magnetite*, and the man-made magnets, such as horseshoe, compass, and bar magnets, are examples of permanent magnets. An example of a man-made temporary magnet is shown in Figure 3-9. It consists of a soft iron core wound with an electrical wire that is connected to a DC power supply through an on/off switch.

If the switch is closed, current will flow through the coil and a magnetic field is produced as shown by the magnetic lines of force in Figure 3-9.

Danish physicist Hans C. Oersted discovered this relationship between electricity and magnetism in 1819. He found that when current flows

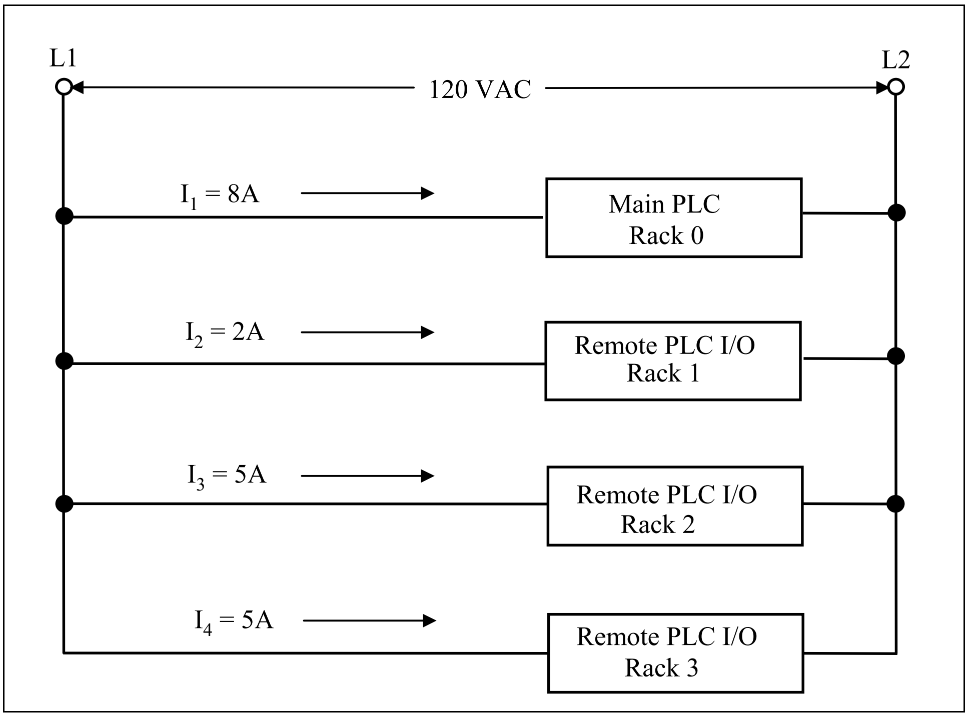


Figure 3-8. PLC Wire-Sizing Application

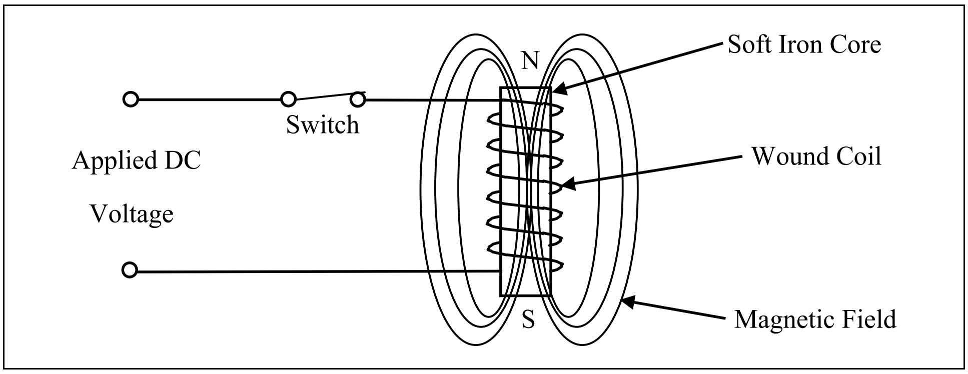


Figure 3-9. Temporary Magnet Using a Solenoid Coil

through an electrical conductor, a magnetic field is created around that conductor. When the conductor is tightly wound in a cylindrical coil shape as shown in Figure 3-8, the magnetic field is intensified. This was the first time that magnetism other than that produced by the rock *magnetite* was observed. Oersted found that there are three ways to increase the magnetic effect of a coil of wire. First, you can increase the amount of electrical current by increasing the applied voltage. Second, you can increase the number of turns of wire in the coil. Third, you can insert an iron core through the center of the coil and it will concentrate the magnetic lines of force. The softer the iron core the more intense the magnetic field.

The result of Oersted's discovery and experiments is the development of electromagnetic industries that rely on magnetic coils in many devices such as motors, electrical generators, transformers, electromagnetic relays, and solenoids.

Solenoids

A solenoid is simply an electromagnet consisting of a coil of wire and an applied voltage as shown in Figure 3-9. However, in practical applications solenoids are combined with a moving armature that transmits the force created by the solenoid into a useful function. Since, the solenoid is used in a variety of electromagnetic devices and control applications, the term solenoid can be confusing unless the full device name or application for the solenoid is also stated. In other words, a complete or proper description of a solenoid might include terms such as, solenoid valve, solenoid operated valve, solenoid clutch, solenoid switch, solenoid operated relay, or solenoid operated contactors. Most of theses devices will be discussed later.

Transformers

Another common electromagnetic device encountered is the transformer. When the current in a solenoid coil changes, the varying magnetic flux cuts across any other coil nearby and produces induced voltage in the other coil. In Figure 3-10, the primary coil L_P is connected to an AC power source that produces varying current in the turns of the primary coil. The secondary coil, L_S is not directly connected to L_P, but the turns are linked by the magnetic field produced by the varying AC current in the primary. This arrangement is called a *transformer*. A soft iron core is generally used to concentrate the magnetic field to improve the efficiency of the transformer. One purpose of the transformer is to transfer power from the primary, where the AC power source is connected, to the secondary, where the induced secondary voltage can produce current in the load resistance that is connected across the secondary coil.

Another purpose of a transformer is to isolate the primary circuit from the secondary load, since the primary and secondary are not electrically connected to each other but are coupled to each other by a magnetic field. A final function of a transformer is to step up or step down the input AC voltage to provide a different value of AC voltage to the output load.

A common application for transformers is in there use in power supplies that are used in the control industry.

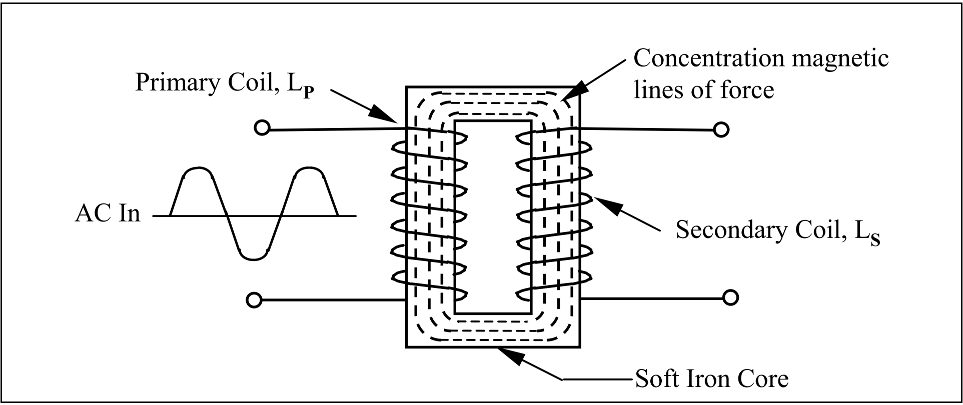


Figure 3-10. Soft Iron Core Transformer

Power Supplies

It is important for process control users to have a basic understanding of the design and operation of DC power supplies because of their wide use in control systems applications. DC power supplies use either half-wave or full-wave rectification, depending on the power supply application in question. Figure 3-11 shows the output waveforms produced by half-wave and full-wave rectification. A schematic diagram of a half-wave rectifier is shown in Figure 3-12. In the circuit in Figure 3-12, the positive and negative cycles of the AC voltage across the secondary winding are in phase with the signal at the primary of the transformer. This is indicated by the dots on each side of the transformer symbol.

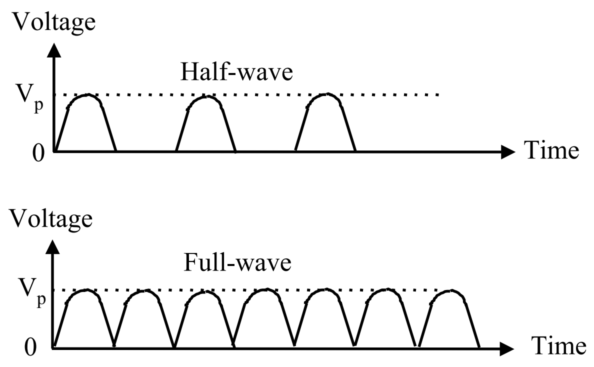


Figure 3-11. Comparison of Half- and Full-Wave Rectification

If we assume that the top of the transformer secondary is positive and the bottom is negative during a positive cycle of the input, then the diode is forward biased and current flow is permitted through the load resistor, R_L, as indicated.

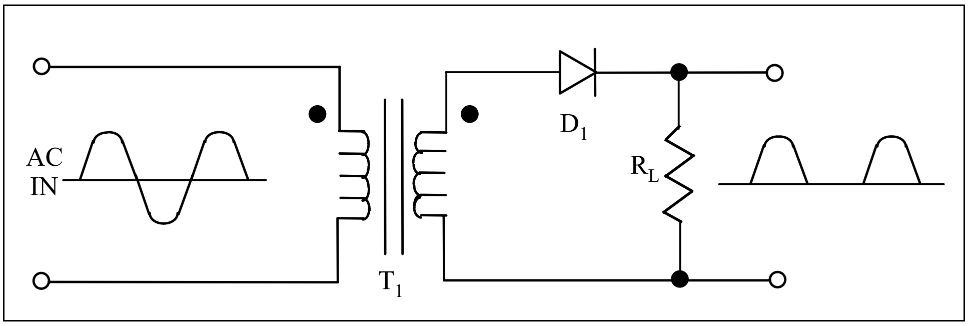


Figure 3-12. Half-wave Rectifier Circuit

During the negative input cycle, the top of the transformer will be negative and the bottom will be positive. This voltage polarity reverse biases the diode. A reverse-biased diode represents an extremely high resistance and serves as an open circuit. Therefore, with no current flowing through R_L, the output voltage will be zero for this cycle.

The basic full-wave rectifier using two diodes is shown in Figure 3-13. The cathodes of both diodes are connected to obtain a positive output. The anodes of each diode are connected to opposite ends of the transformer secondary winding. The load resistor (R_L) and the transformer center are connected to ground to complete the circuit.

In a full-wave rectifier, the current through R_L goes in the same direction for each alteration of the input, so we obtain DC output for both halves of the sine-wave input, or full-wave rectification.

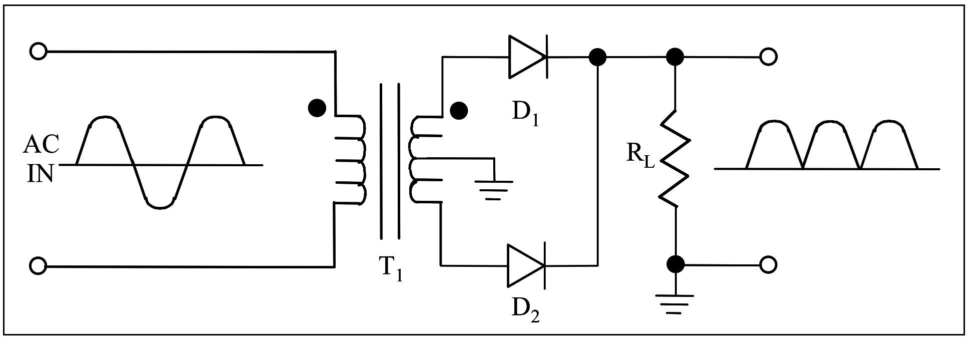


Figure 3-13. Full-wave Rectifier Circuit Using Two Diodes

The ripple frequency of the full-wave rectifier is also different from that of a half-wave circuit. Since each cycle of the input produces an output across R_L, the ripple frequency will be twice the input frequency. The higher ripple frequency and characteristic output of the full-wave rectifier is easier to filter than is the corresponding output of a half-wave rectifier.

To reduce the amount of ripple, a filter capacitor is placed across the output. A capacitor consists of two metal plates separated by a dielectric material.

A bridge structure of four diodes is used in most power supplies to achieve full-wave rectification. In the bridge rectifier shown in Figure 3-14, two diodes will conduct during the positive alteration and two will conduct during the negative alteration. A bridge rectifier does not require a center-tapped transformer as is used in a two-diode, full-wave rectifier.

The DC output that appears across R_L of the bridge circuit has less ripple amplitude because a filter capacitor has been placed across the output. A capacitor opposes any changes in the voltage signal. If the filter capacitor is properly sized and placed across the output, it will prevent the output signal from returning to zero at the end of each half cycle, as shown in Figure 3-14.

Bridge rectifiers are commonly used in the electronic power supplies of instruments because of their simple operation and desirable output. The diode bridge is generally housed in a single enclosure that has two input and two output connections.

The DC output voltage signal from a bridge rectifier is filtered to produce the smooth DC signal that most instrumentation circuits and devices need.

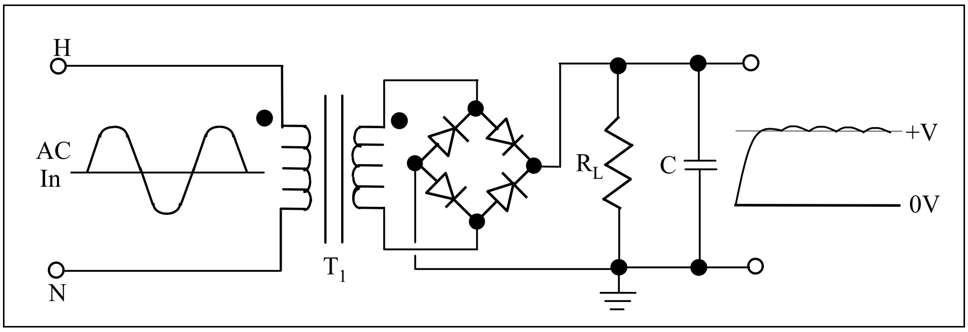


Figure 3-14. Full-wave Bridge Rectifier Circuit

We have now presented the most common types of electronic devices and circuits encountered in programmable controller applications. We now need to discuss some of the common electrical control devices.

Electrical Control Devices

A wide variety of electrically operated devices can be found in control applications. In this section, we will discuss the most common devices (as well as the electrical symbol used in drawings to represent each device).

Electrical Relays

The most common control device encountered in control applications is the *electromechanical relay*. It is called electromechanical because it consists both of electrically operated parts and mechanical components. One electrical part is a long thin wire that is wound into a close-packed helix around an iron core. When a changing electric current is applied, a strong magnetic field is produced. The resulting magnetic force moves the iron core, which is connected to a set of contacts. This transfers the common contact from the normally closed contact to the normally open contact.

A simplified representative diagram of an electrical relay is shown in Figure 3-15, with two sets of relay contacts. Each set consists of a common (COM) contact, a normally open (NO) contact, and a normally closed (NC) contact. *Normally open* and *normally closed* refer to the status of contacts when no electrical power is applied to the relay or when the relay is in the shelf position. For example, if there is no electrical continuity between a common contact and another contact in the set when the relay is on a storage shelf, then this set of contacts is termed the "normally open" set of contacts. These contacts in the relay are used to make or break electrical connections in control circuits.

The electrical schematic symbol for a NO set of contacts is given in Figure 3-16a, and the symbol for the NC set is shown in Figure 3-16b. The standard symbol for the relay solenoid is a circle with two connection dots as shown in Figure 3-16c. This symbol will normally also include a combination of letter(s) and number(s) to identify each control relay on a control drawing. Typical designations for the control relays on a schematic are CR1, CR2, . . . CRn.

Electric Solenoid Valves

Another electrically operated device that is commonly encountered in process control is the *solenoid valve*. The solenoid valve is a combination of two basic functional units: an electromagnetic solenoid with its core and a valve body that contains one or more orifices. The flow through an orifice can be allowed or prevented by the action of the core when the solenoid is energized or deenergized, respectively.

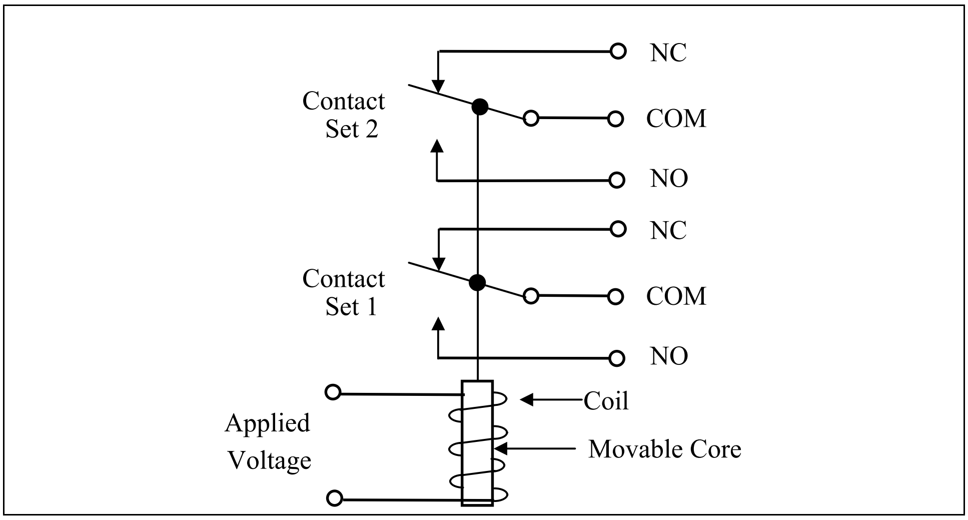
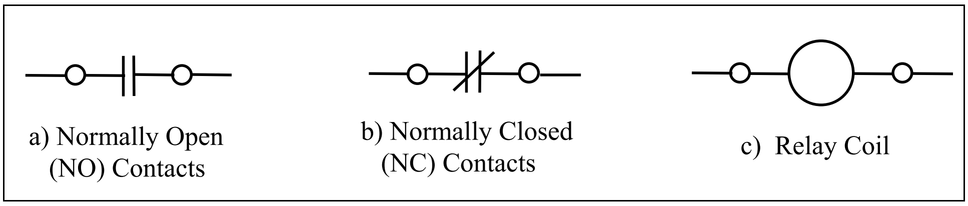


Figure 3-15. Pictorial Representation of Electric Relay


a) Normally Open (NO) Contacts
b) Normally Closed (NC) Contacts
c) Relay Coil
Figure 3-16. Electrical Relay Symbols

Solenoid valves normally have a solenoid coil that is mounted directly on the valve body. The coil is enclosed and free to move in a sealed tube called a *core tube*, which thus gives it a compact assembly. The two different electrical schematic symbols that are used to represent solenoid valve coils on electrical and control drawings are given in Figure 3-17.

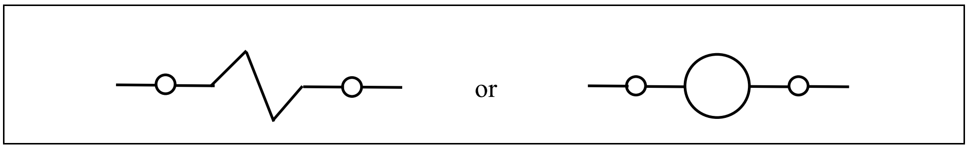


Figure 3-17. Schematic Symbols for Solenoid Valve Coils

The three most common types of solenoid valves are direct acting, internal pilot operated, and manual reset. In *direct-acting valves*, the solenoid core directly opens or closes the orifice, depending on whether the solenoid is energized or deenergized, respectively. The valve will operate from 0 psi inlet pressure to its maximum rated inlet pressure. The force that is required to open a solenoid valve is proportional to the orifice size and the pressure drop across the valve. As the orifice size increases, the force needed to operate the valve increases. To keep the solenoid coil small and open large orifices at the same time, some plants use internal pilot-operated solenoid valves.

Internal pilot-operated solenoid valves have a pilot and bleed orifice and use the line pressure to operate. When the solenoid is energized, the core opens the outlet side of the valve. The imbalance of pressure causes the line pressure to lift the piston or diaphragm off the main valve orifice, thus opening the valve. When the solenoid is deenergized, the pilot orifice is closed, and to close the valve the full line pressure is applied to the top of the piston or diaphragm through the bleed orifice. In some applications, a small manual valve replaces the bleed orifice so the plant personnel can control the speed that is required to open and close the valve.

The *manual reset* type of solenoid valve must be manually positioned (latched). It will return to its original position when the solenoid is energized or deenergized, depending on the valve design.

Three configurations of solenoid valves are normally available for process control applications: two, three, and four way. Figure 3-18 shows the process diagram symbols that are used to represent two-, three-, and four-way solenoid valves. Two-way valves have one inlet and one outlet process connection. They are available in either NO or NC configurations. Three-way solenoid valves have three process piping connections and two orifices (one orifice is always open and the other is always closed). These valves are normally used in process control to alternately apply air pressure to and exhaust air from diaphragm-operated control valves or single-acting cylinders. Four-way solenoid valves have four pipe connections: one for supplying air pressure, two for process control, and one for air exhaust. These valves are normally used to control double-acting actuators on control valves.

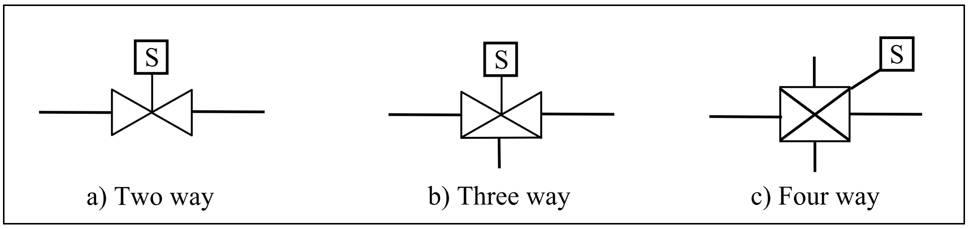


Figure 3-18. Process Diagram Symbols for Solenoid Valves

Electrical Contactors

Electrical contactors are relays that can switch high-current loads that are normally greater than 10 amps. They are used to repeatedly make and break electrical power circuits. An electrically operated solenoid is the most common operating mechanism for contactors. The solenoid is powered by low-ampere AC voltage signals from control circuits. Instead of opening or closing a valve, the linear action of the solenoid coil is used to open or close sets of contacts with high-power ratings. The higher-rated sets of electrical contacts are used in turn to control loads, such as motors, pumps, and heaters. The schematic symbol for a typical electrical contactor is shown in Figure 3-19.

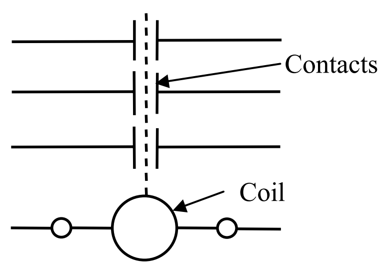


Figure 3-19. Schematic Symbol for an Electrical Contactor

Other Control Devices

Hand switches, instrument switches, control switches, pushbuttons, and indicating lights are some of the other devices commonly encountered in process control. Switches and pushbuttons are used to control other devices, such as solenoid valves, electric motors, motor starters, heaters, and other process equipment.

Hand switches come in a variety of designs and styles. They can be mounted on control panels or in the field near the process equipment. Instrumentation switches are generally field-mounted and are used to sense pressure, temperature, flow, liquid level, and position.

To make it easier to read control drawings, a set of standardized symbols is defined by IEEE 315-1975, "Graphic Symbols for Electrical and Electronics Diagrams (Including Reference Designation Letters)". The most common symbols used in control diagrams are shown in Figure 3-20.

EXERCISES

3.1 A charge of 15 coulombs moves past a given point every second. What is the current flow?

3.2 Calculate the current in a conductor, if 30×10^{18} electrons pass a given point in the wire every second.

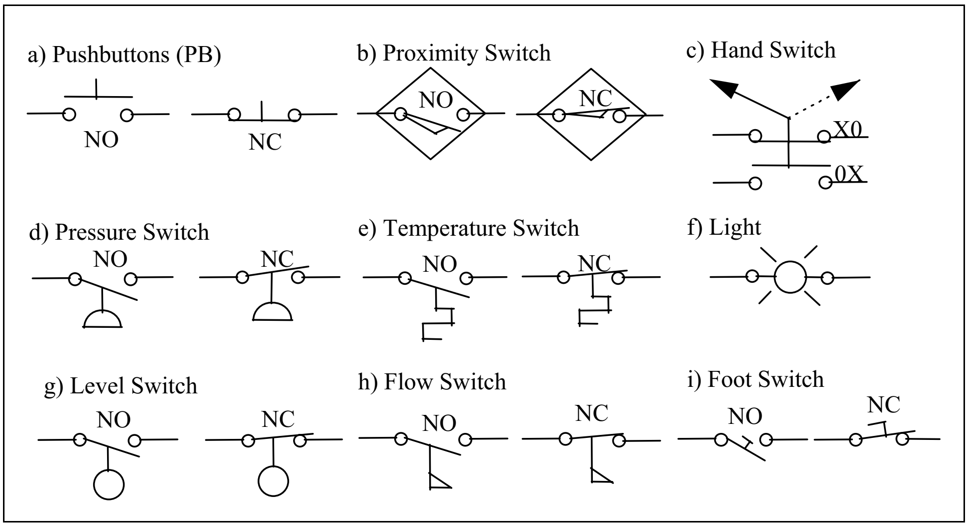


Figure 3-20. Common Electrical and Instrument Symbols

3.3 Calculate the area in circular mils (cmil) of a conductor that has a diameter of 0.003 inch.

3.4 Calculate the resistance of 500 feet of copper wire that has a cross-sectional area of 4110 cmil if the ambient temperature is 20°C.

3.5 Calculate the resistance of 500 feet of silver wire with a diameter of 0.001 inch if the ambient temperature is 20°C.

3.6 The resistance to ground on a faulty underground telephone line is 20 Ω. Calculate the distance to the point where the wire is shorted to ground if the line is a 22 AWG copper conductor and the ambient temperature is 25°C.

3.7 Find the total current (I_t) in a circuit that has two 250 Ω resistors in series with a voltage source of 24 VDC. Also find the voltage drop across each resistor, where V_1 is the voltage across the first resistor and V_2 is voltage across the other resistor.

3.8 In the parallel resistance circuit shown in Figure 3-4, assume that $R_1 = 100\ \Omega$, $R_2 = 200\Omega$, and $V_t = 100$ VDC. Find I_1, I_2, and I_t.

3.9 Assume that the Wheatstone bridge circuit shown in Figure 3-5 is balanced (i.e., $I_g = 0$) and that $R_1 = 2\text{k}\ \Omega$, $R_2 = 10\text{k}\ \Omega$, $R_s = 2\text{k}\ \Omega$, and $V_t = 12$ VDC. Calculate the value of I_1, I_2, and R_x. Also, what are the voltage drops across R_1 and R_2?

3.10 What are the three methods used to increase the magnetic effect of a coil of wire?

3.11 List some of the functions performed by a transformer.

3.12 What is the advantage of a full-wave bridge rectifier circuit over a half-wave bridge circuit?

3.13 Explain the purpose of electromechanical relays and how they operate.

3.14 List the three common types of electrically operated solenoid valves.

BIBLIOGRAPHY

1. Rockis, G. and Mazur, G., *Electrical Motor Controls Automated Industrial Systems*, 2nd edition. Homewood, IL: American Technical Publishers, Inc., 1987.

2. *The National Electrical Code Handbook*, 6th edition., National Fire Protection Association, 1996.

4 Input/Output Systems

Introduction

The input/output (I/O) system provides the physical connection between the process equipment and the central processing unit (CPU) or simply the "processor." It uses various interface circuits to convert the sensed or measured physical quantities of the process, such as motion, level, temperature, pressure, and position, to a logic signal to be used by the PLC processor. Based on the status of the sensed or measured values, the control program in the PLC processor uses various output circuits or modules to activate devices, such as valves, motors, pumps, and alarms, to exercise control over a machine or process.

I/O Circuit Mounting Configurations

There are three basic mounting configurations used for the I/O circuits. First, I/O circuits are packaged in groups of 4, 8, 16 and 32 in points in industrial grade modular units. These modules are then mounted in rugged equipment housings. The backplane of the housings into which the modules are plugged has a printed circuit card that contains the parallel communications bus to the processor. It also has the connections for the DC voltages that supply DC power to the digital and analog circuits in each module. These I/O housings can be mounted in a main control panel or in a field-mounted enclosure protected from the weather and adverse conditions.

In the second I/O mounting configuration is a standard din-mounting rail, which is used to hold point type I/O blocks. These point blocks use printed circuit board (PCB) technology but have connector pins on the side of each module. These side connectors have serial communication lines (normally 4 lines) that are connected to the PLC processor module. The side connectors also have several pins for the DC voltages that are required to power the logic and interface circuits in the I/O point blocks. In the point block type I/O system, the module assembly is mounted horizontally or vertically on a standard terminal strip metal din rail. The point blocks are installed by sliding them together on the din rail. They can also be pulled apart easily for maintenance or troubleshooting.

For both the module housings or din rail configuration, any I/O module or point module can be inserted into any I/O position, and the housings or mounting bases are designed so that the I/O modules can be removed without turning off the power or removing the field wiring. Both configurations have removable terminal blocks for wiring of the field devices to the modules. These terminal strips can be removed with power on for maintenance.

In the final I/O mounting configuration, where there is a limited number of I/O point such as in the case of small PLCs with normally 32 or fewer I/O points, the I/O circuits are an integral part of the PLC.

Discrete Inputs

Discrete inputs are the most common class of field signals in a PLC system. A discrete input field device provides an input signal that is separate and distinct in nature. The discrete input signals have only two states, open/closed or on/off. Table 4-1 lists the most common discrete input devices that are encountered in process control applications. Except for the photoelectric sensor, they are all normally some form of switch contacts or relay contacts that are open or closed. The photoelectric sensor may have a relay contact output or an on/off voltage signal such as 0 or 5 VDC.

When in operation, if a discrete or digital input device is closed, the input circuit in the PLC senses the supplied voltage and converts it to a logic-level signal acceptable to the CPU to indicate the status of the device. A logical 1 indicates ON or Closed, and a logical 0 indicates OFF or Opened.

Thumbwheel switches	Position switches
Temperature switches	Pressure switches
Flow switches	Hand switches
Level switches	Proximity switches
Valve position switches	Relay contacts
Starter auxiliary contacts	Limit switches
Selector switches	Motor starter contacts
Pushbuttons	Photoelectric sensors

Table 4-1. Discrete Input Devices

Discrete Outputs

Discrete output control is limited to devices that require switching only one of two states, such as on/off, open/closed, or extended/retracted. Table 4-2 lists the most common discrete output devices encountered in process and machine control applications.

Annunciators	Electric valves
Alarm lights	Alarm horns
Electric control relays	Solenoid valves
Electric fans	Motor starters
Indicating lights	Heater starters

Table 4-2. Discrete Output Devices

In operation, the digital output interface circuit switches the control voltage that is connected to a field or remote device. If an output is turned ON through the control program, the control voltage is switched by the interface circuit to activate the referenced (addressed) output device.

I/O Signal Types

Each discrete input and output signal is powered by either a field or a panel supplied voltage source (e.g., +5 VDC, 120 VAC, 24 VDC, and so on). I/O interface circuits are available for various AC and DC voltage ratings, as listed in Table 4-3.

The most common voltage levels used in industrial applications is 120 VAC and 230 VAC because these voltages are readily available in industrial plants. However, low-voltage DC sources, such as +5 VDC, +12 VDC, +24 VDC, and +48 VDC, are also widely used for safety reasons because there is less risk of injury with low-voltage DC sources than with 120 or 230 VAC.

+ 5 VDC	120 VAC or VDC
+ 12 VDC	230 VAC or VDC
+ 24 VAC or VDC	100 VDC
+ 48 VAC or VDC	Dry contacts

Table 4-3. Typical Discrete I/O Signal Values

Sinking and Sourcing Operations

Sink and source operations refer to the electrical configuration of an electronic circuit in a device whether it is an input module or a field input device. If the device provides current during its ON state, the device is said to be sourcing current. Conversely, if the device receives current in the ON or true state, it is said to be sinking current. Therefore, we can have sinking and sourcing field devices as well as sinking and sourcing input modules. However, the most common configuration using PLC applications is the sourcing field input device and the sinking input module.

Potential interface problems can arise if the user does not design the I/O system to properly match sinking and sourcing operations for the devices in a system. The user must use a sinking input module if the field devices connected to the module are sourcing devices. Conversely, for proper operation, sourcing input circuits must be used if they are connected to a sinking field device. A problem would arise, for instance, if an input module is designed for sink operation and all input devices except one are operating in a source configuration. The sinking input device may be ON, but the module would not detect the ON signal even though a voltage could be measured across the module's input terminals. There is also the potential for damage to the mismatched field device and input circuit in the I/O module.

Discrete AC Voltage Input Electronic Circuits

A block diagram of a typical AC voltage discrete input electronic circuit is shown in Figure 4-1. Discrete AC input circuits vary widely among PLC manufacturers, but, in general, discrete AC input circuits operate in a manner similar to that shown in Figure 4-1.

The input voltage circuit is composed of two primary parts: the power section and the logic section. The power and logic sections of the circuit are normally coupled with a circuit that electrically isolates the input

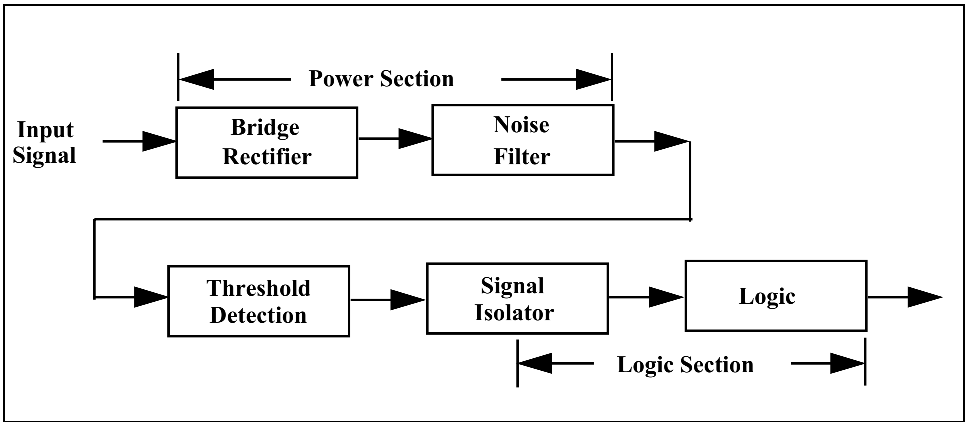


Figure 4-1. Block Diagram of a Typical Discrete AC Input Circuit

power section from the logic circuits. This electrical isolation is very important in a normally noisy industrial environment. The main problem with the early application of computers in process control was that the input and output circuits were not designed for the harsh industrial environment.

Figure 4-2 shows a typical discrete AC input circuit. The power section of the circuit performs the function of converting the incoming voltage (115 VAC, 230 VAC, etc.) from a field input device, such as described in Table 4-3, to a logic-level signal to be used by the PLC processor. The bridge rectifier circuit converts the AC incoming signal to a DC level that is sent to a filter circuit consisting of capacitor, C , and resistors, R_2 and R_3, which minimize the ripple from the full-wave bridge circuit. This RC filter circuit causes a signal delay that is typically 10–25 ms. The threshold circuit uses a Zener diode (Z_d) to detect whether the incoming signal has reached the proper voltage level for the specified input rating. If the input signal exceeds and remains above the threshold voltage for a duration of at least the filter delay, the signal will be accepted as a valid input.

When a valid signal has been detected, it is passed through the isolation circuit, which completes the electrically isolated transition from AC or DC voltage to a logic level voltage. The logic level signal from the isolator is used by the logic circuit and made available to the processor via the data bus on the backplane of the PLC rack. Electrical isolation up to 1500 VAC is generally provided so that there is no electrical connection between the field device (power) and the controller (logic). This electrical separation helps prevent large voltage spikes from damaging the logic side of the interface (or the controller). An optical coupler as shown in Figure 4-2 normally provides the coupling between the power and logic sections. Electrical isolation is one of the reasons the programmable controller has gained wide acceptance in the process industries.

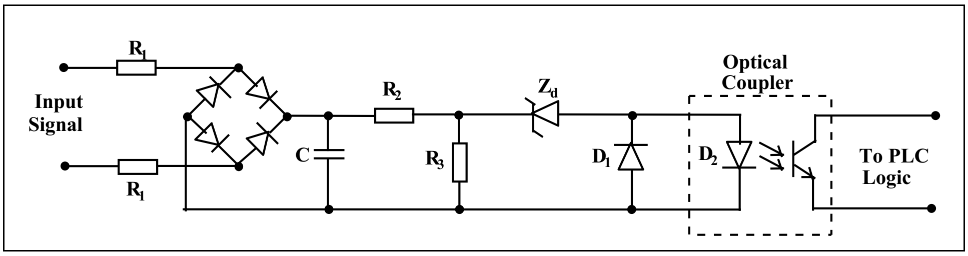


Figure 4-2. Typical AC Voltage Discrete Input Circuit

In medium, and large PLC systems, these discrete input circuits are mounted together on a single circuit board and installed in an input module or point block. Input modules are available with 4, 8, 16, or 32 input circuits in one unit.

Discrete AC Input Modules

Most discrete AC input modules will have a signal indicator to signify that the proper input voltage level is present (a switch is closed). A LED is normally used to indicate the status of the input. These indicating lights are an important aid during system start-up and troubleshooting. A discrete AC input connection diagram is shown in Figure 4-3 where 120 VAC hot (L1) is connected to the field devices and 120 VAC neutral (L2) is applied to the neutral (N) terminal of the input module.

The letters "ACI-120" shown above the module in Figure 4-3 are a typical model number that might be used by a PLC manufacturer. We will use similar model numbers for other I/O modules in this chapter.

Direct Current (DC) Input Modules

The DC voltage input modules convert discrete on/off direct current inputs to logic level signals compatible with the programmable controller. They are generally available in three voltage ranges: 12, 24, and 48 VDC. Typical instruments that are compatible with the module include limit switches, valve position switches, pushbuttons, DC proximity switches, float switches, and photoelectric sensors.

The wiring diagram for a DC input module would be the same as the wiring of the AC input module shown in Figure 4-3 except the supply voltage would be a direct current voltage instead of an AC voltage. The AC hot voltage signal (L1) sent to the field devices would be replaced by a positive DC voltage and the neutral terminal on the module would be replaced by a DC common terminal.

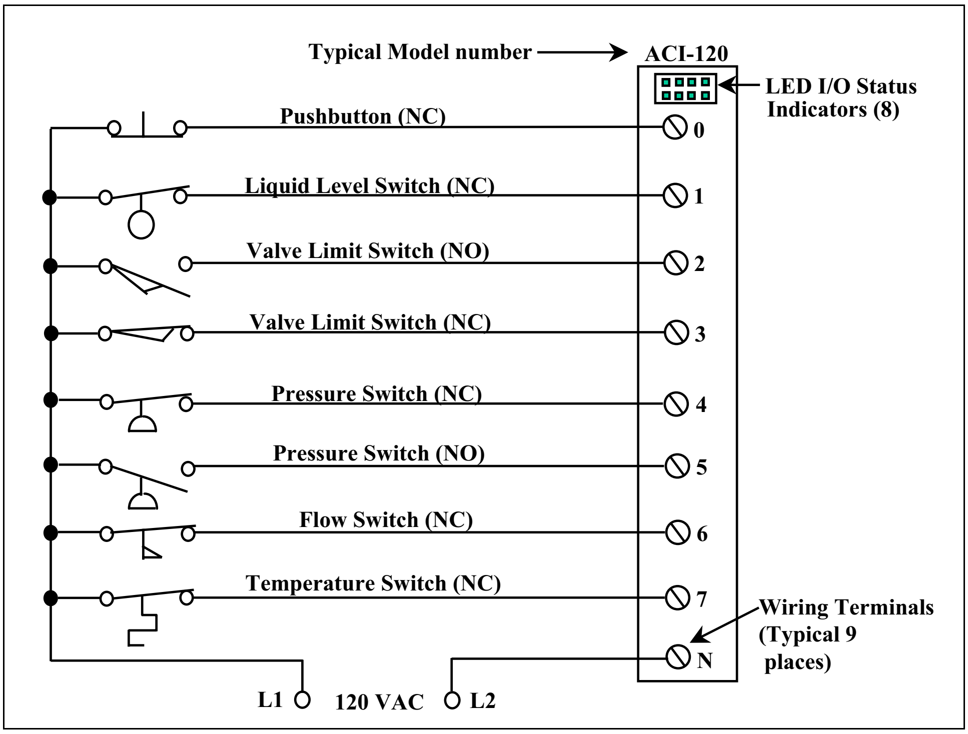


Figure 4-3. Typical Discrete AC Input Module Wiring Diagram

Transistor-Transistor Logic (TTL) Input Modules

The TTL input modules let the controller accept signals from TTL-compatible devices, including solid-state controls and sensing instruments. TTL inputs are also used for interfacing with 5-VDC level control devices and several types of photoelectric sensors. The TTL interface has a configuration similar to the DC input modules; however, the input delay time caused by filtering is generally much shorter. TTL input modules normally require an external +5 VDC power supply.

Isolated Discrete Input Module

I/O modules usually have a common return line connection for each group of inputs or outputs on a single module. However, sometimes it may be required to connect an input device of different ground levels to the controller. In this case, isolated input modules with separate return lines for each input circuit are available (AC or DC) to accept these signals. The operation of the isolated interface is the same as the standard discrete I/O, except the common of each input is separated from the other commons in the module. The result is that the isolated input module requires twice as many input connection terminals so it can accommodate only half the inputs in the same physical space as shown in Figure 4-4. The module number for the isolated 120 VAC input module is "IACI-120" and the isolated 220 VAC input module model number is "IACI-220. "

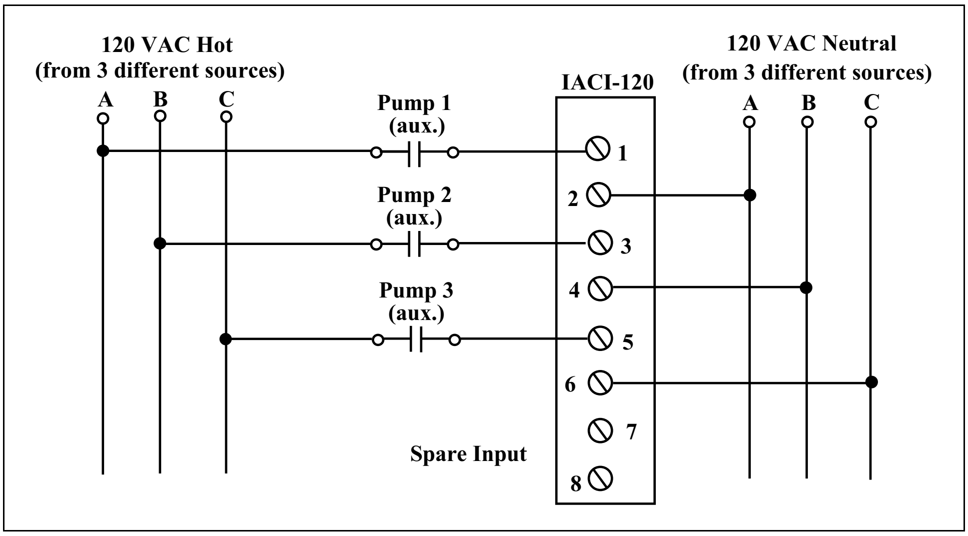


Figure 4-4. Discrete Isolated AC Input Module Wiring Diagram

Discrete AC Output Electronic Circuit

Figure 4-5 shows a block diagram of a typical discrete AC output electronic circuit. AC output circuits vary widely among PLC manufacturers. The block diagram is representative of the basic operations performed in AC output circuits. The circuit consists primarily of the logic and power sections, coupled by an isolation circuit. The output interface can be thought of as a simple switch through which power can be provided to control the output device.

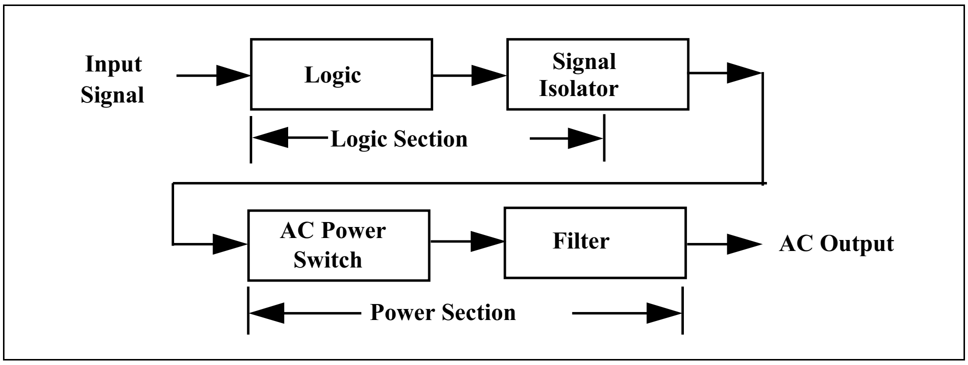


Figure 4-5. Block Diagram of a Discrete AC Output Circuit

First, the processor sends an output signal of 0 or 1 to the logic circuit section. The signal from the logic section is then passed through an isolation circuit. The logic signal from the isolation circuit is then fed to an AC power switching circuit and filter. Finally, this discrete AC output signal controls any AC operated field device connected to the module output point.

The AC power switching section generally uses a Triac or a silicon-controlled rectifier (SCR) to switch the AC power between on or off. The AC switch is normally protected by an RC circuit or a metal-oxide varistor (MOV), which is used to limit the peak voltage to some value below the maximum voltage rating. These protection circuits or devices also prevent electrical noise from affecting the module operation. A fuse may be provided in the output circuit to prevent excessive current from damaging the AC switch. If the fuse is not provided on each circuit in the module, it should be added to the exterior of each output circuit.

Discrete AC Output Module

In small, medium, and large PLC systems, discrete output AC circuits are mounted together on a single circuit board and installed in an output module or a point block. Output modules or point blocks are available with 4, 8, 16, or 32 output points on the circuit card.

As with input modules, the output modules also have LED indicators to show the state of the operating logic. An AC output module connection diagram is illustrated in Figure 4-6 with the LED indicators on the top of the module. The first two outputs on the module are wired to two starters for Heater 1 and Heater 2. The starter overload (OL) contacts are wired in series to turn off the starter in case of a high current in the starter circuits. The next two outputs at terminals 2 and 3 are connected to 120 VAC solenoid valves LV-1 and LV-2. The last four output points are connected by four starters with OL devices in series with the starter coils. Note that the switching voltage is field-supplied to the module so the module is a sourcing device.

Direct Current (DC) Output Module

The DC output module is used to switch DC loads. Functional operation of the DC output is similar to the AC output; however, the power circuit generally employs a power transistor to switch the load. Like Triacs, transistors are also susceptible to excessive applied voltages and large surge currents, which could result in overheating and a short circuit. To prevent this condition from occurring, the power transistor will normally be protected with a fuse.

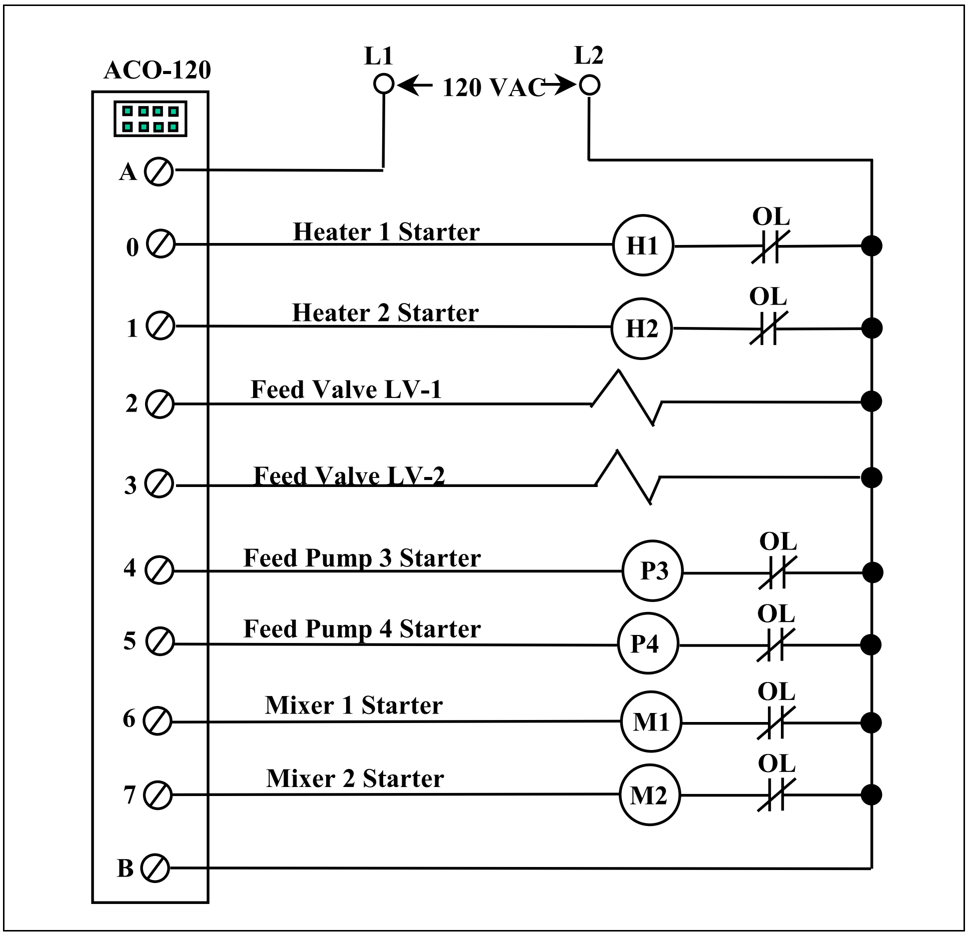


Figure 4-6. Typical Discrete Output Module Wiring Diagram

The wiring diagram for a DC output module would be the same as the wiring of the AC output module shown in Figure 4-6 except the supply voltage would be a DC voltage instead of an AC voltage. A positive DC voltage terminal would replace the terminal of the AC hot line and the AC neutral terminal would be replaced by a ground or negative DC voltage terminal.

Dry Contact Output Module

The dry contact output module allows output devices to be turned ON or OFF by a normally open (NO) or normally closed (NC) set of relay contacts. The advantage of relay or dry contact outputs is that they provide electrical isolation between the PLC and the field device. The solid-state electrical switching circuit in the standard AC output module has a small leakage current even when the switching circuit is turned off, and this small current can cause false signals in some cases. In these applications, the dry contact output module should be used.

Dry contact outputs can be used to switch either AC or DC loads but are normally used in AC applications to provide electrical isolation between PLCs and other complex electrical equipment, such as variable speed drives (VSDs). Figure 4-7 shows a typical dry contact output module with four normally open contacts controlling the starting and stopping of two variable speed motor drive units. In this application, there is complete electrical isolation between the PLC and VSDs.

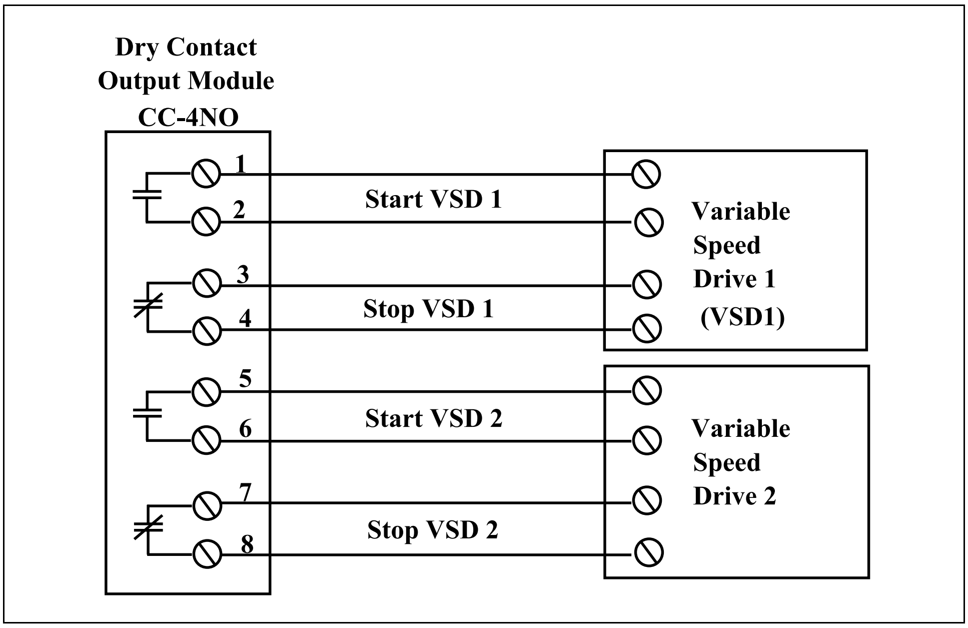


Figure 4-7. Typical Dry Contact Output Module Wiring Diagram

TTL Output Module

The TTL output module allows the controller to drive output devices that are TTL compatible, such as seven-segment LEDs, integrated circuits, and various +5 VDC logic-based devices. These modules generally require an external +5 VDC power supply with specific current requirements to handle the power requirements of the field device.

Isolated AC Output Module

An isolated AC output interface is shown in Figure 4-8. Note that the isolated AC output module is driving three different loads (starters for pumps 1, 2 and 3) that are connected to three different sources of AC power. The advantage of this module is that we do not have to be concerned with the different AC voltage sources in our process plant. The disadvantages are that we increase the amount of wiring required and decrease the number of available inputs per module by a factor of two (2). In the application shown in Figure 4-8, three different sources of 120 VAC power are used to turn on three separate motor starters for Pumps 1, 2, and 3. This is a typical application for isolated AC output modules.

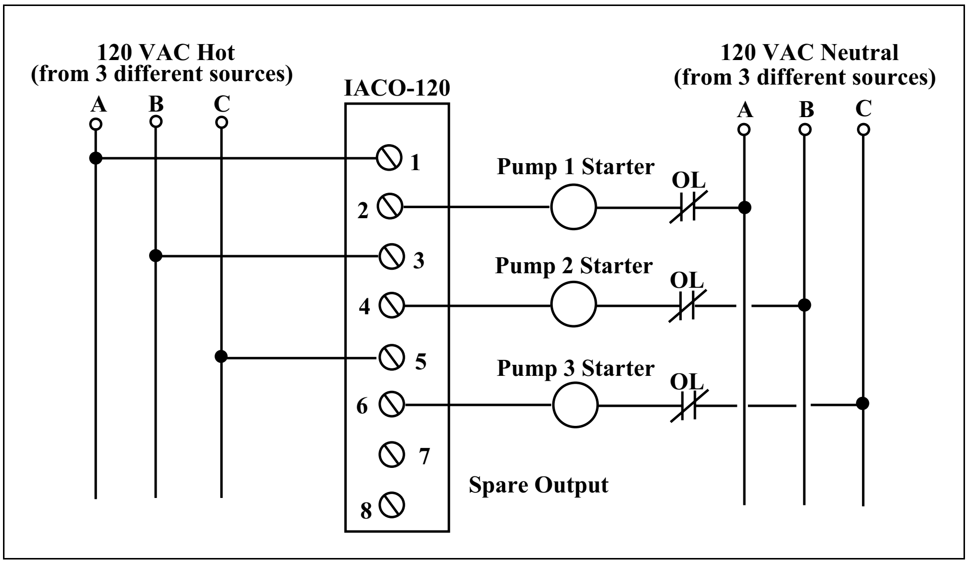


Figure 4-8. Typical Isolated AC Output Module Wiring Diagram

Analog I/O Modules

The availability of low-cost integrated circuits and industrial grade electronic circuits greatly increases the capabilities for analog circuits in PLCs. This expanded capability led to the introduction of sophisticated analog I/O modules.

Analog input modules allow measured quantities to be received from process instruments and other devices that provide analog data, while analog output modules allow control of devices that require a continuous analog signal. The analog I/O will allow monitoring and control of analog voltages and currents, which are compatible with many sensors, motor drives, and process instruments. With the use of analog and special-purpose I/O, most process variables can be measured or controlled with appropriate interfacing. Table 4-4 lists typical I/O devices that are interfaced to PLCs using analog modules.

Analog Input Devices	Analog Output Devices
Flow transmitters	Electric motor drives
Pressure transmitters	Analog meters
Temperature transmitters	Chart recorders
Analytical transmitters	Process controllers
Position transmitters	Current-to-pneumatic transducers
Potentiometers	Electrical-operated valve
Level transmitters	Variable speed drives
Speed instruments	

Table 4-4. Typical Analog I/O Field Devices

Analog Input Modules

The analog input interface contains the circuitry necessary to accept analog voltage or current signals from field devices. The voltage or current inputs are converted from an analog to a digital value by an analog-to-digital converter (ADC). The conversion value, which is proportional to the analog signal, is passed through to the controller's data bus and stored in a memory location for later use.

Typically, analog input interfaces have very high input impedance, which allows them to interface field devices without signal loading. The input line from the analog device generally uses shielded, twisted-pair conductors. The shielded cable greatly reduces the electrical interferences from outside sources. The input stage of the interface provides filtering and isolation circuits to protect the module from additional field noise. A typical analog input connection is illustrated in Figure 4-9. In the example shown, the analog input module is providing the DC voltage required by the field current transmitters.

Most analog modules are designed to sense up to 16 single-ended or 8 differential analog input signals representing flow, pressure, level, etc. It then converts them to a proportional 10- to 15-bit binary word in memory. Inputs to a particular module must, in general, be all single-ended or all differential, and the type of signal is either hardware or software selectable. The converted signals are stored in memory in the module and are sent to the processor memory in groups or blocks of data.

The control program uses configuration data to set up the analog module. Typical configuration information includes range selection (i.e., +1 to +5 VDC, 4 to 20 mA, and so on) and signal scaling.

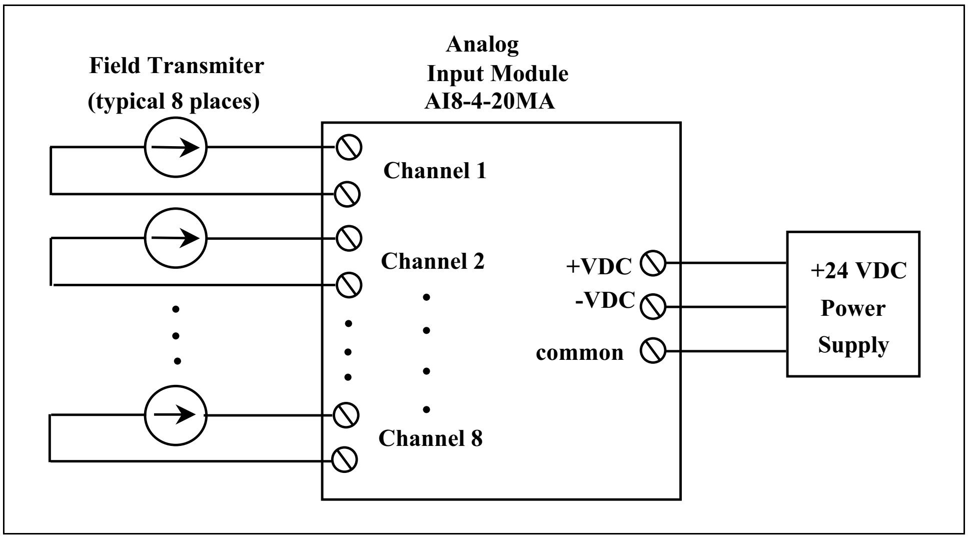


Figure 4-9. Typical 8-Channel Analog Input Module Wiring Diagram

Analog Output Modules

The analog output modules receive digital data from the CPU of the PLC and converts the signal to an analog form (i.e., a continuous voltage or current) that is proportional to the digital signal to control an analog field device. The digital data or signal in the form of "1s" and "0s" is passed through a digital-to-analog converter (DAC) circuit and sent out in analog form. Isolation between the output circuit and the logic circuit is generally provided through optical couplers. These analog output modules or circuits normally require an external power supply with certain current and voltage requirements but in some cases can be powered by the equipment rack power supply.

Special-Purpose Modules

There are a wide variety of special-purpose I/O modules used in PLC systems. One PLC manufacturer has over 120 different types of I/O modules with many special function modules. We will discuss three of the more common special-purpose modules: the Encoder/Counter Input Module, Synchronous Serial Interface Absolute Encoder, and the High Speed Pulse Input Module.

Encoder/Counter Input Module

The encoder/counter input module provides a high-speed counter, external to the processor, which responds to input pulses sensed at the interface. This counter's operation is normally independent of either program scan or I/O scan. The reason for this is fairly simple: if the counter were dependent on the PLC program, high-speed pulses would be missed during a program scan. Typical applications of the encoder/counter interface are operations that require direct encoder input to a counter, which is capable of providing direct comparison outputs.

The encoder/counter input module accepts input pulses from an incremental encoder, which provides pulses that signify position when the encoder rotates. Six bytes of data are transferred from the module to the processor. One byte of data is normally transferred from the processor to the module each scan of the processor.

Absolute encoders are generally used with interfaces that receive BCD or Gray code data, which represents the angular position of the mechanical shaft being measured.

During normal operations, the modules receive input pulses, which are counted and compared with a preset value selected by the operator. The counter input module normally has an output signal available that is energized when the input and preset counts are equal; however, this is not needed in most PLCs. Since the data is available in the CPU, the programmer can use a comparison function to drive an output in the control program.

The data communication between the encoder/counter interface and the processor is bi-directional. The module accepts the preset count value and other control data from the CPU and transmits data and status to the PLC memory. The output controls are enabled from the control program, which instructs the module to operate the outputs according to the count values received. The CPU, using the control program, enables and resets the counter operation.

Synchronous Serial Interface Absolute Encoder Module

The Synchronous Serial Interface (SSI) Absolute Encoder module collects serial data from industrial absolute position encoding sensors, including linear, rotary, and optical distance measuring devices. The module converts a serial data stream from a standard SSI sensor into absolute position data readable as a 32-bit hexadecimal process value. The modules are normally Gray or Binary code capable with Gray code to Binary code conversion. The SSI module normally supports serial data words of 2 through 31 bits in length. User can normally select the SSI word delay with a 16 microsecond (μs) through 64 ms time intervals.

Pulse Counter Input Modules

The pulse counter input modules are used to interface with field instruments that generate pulses, such as positive displacement (PD) flowmeters and turbine type flowmeters. In a typical application, the flowmeter will generate a signal pulse with amplitude of +5 V based on the volume of fluid passing. Each pulse represents a fixed volume, for example 1 pulse might equal 1 liter of fluid. In a typical application, the PLC counts the number of pulses received by the pulse input module and then calculates the total volume of fluid that passes for a fixed time period.

Intelligent I/O Modules

In the previous sections, we discussed discrete, analog, and special-purpose I/O modules that will normally cover 90% of the I/O applications encountered in PLC systems. However, to process certain types of signals or data efficiently, microprocessor-based intelligent modules are required. These intelligent interfaces include those that condition input signals, such as thermocouple modules, or other signals that cannot be interfaced using standard I/O modules. These intelligent modules can perform complete processing functions, independent of the CPU and the control program scan. In this section, we will discuss three of the most commonly available intelligent modules: the thermocouple input module, resistance temperature detector (RTD) input module, and stepping motor output modules.

Thermocouple Input Modules

When two wires composed of dissimilar metals are joined at one end and that end is heated, a voltage can be measured across the open ends of the two wires. All dissimilar metals exhibit this effect, and this configuration of two dissimilar metals joined together is called a *thermocouple*, which is abbreviated T/C. If the temperature around the T/C conjunction is varied the voltage produced will vary.

A thermocouple (input module is designed to accept inputs directly from a T/C as shown in Figure 4-10. The T/C input module provides cold junction temperature compensation to correct for changes in ambient temperature around the T/C modules. The operation of this type of module is similar to the standard analog input with the exception that low-level millivolt signals are accepted from the T/C (approximately 43 mV at maximum temperature for a J-type T/C). These signals are filtered, amplified, and digitized through an A/D converter and then sent to a built-in microprocessor to perform linearization of the millivolt input signal to convert the signal to a temperature value. Finally, the temperature value is sent to the CPU on command from a program instruction. The temperature data can be used by the PLC control program to perform temperature control and/or indication.

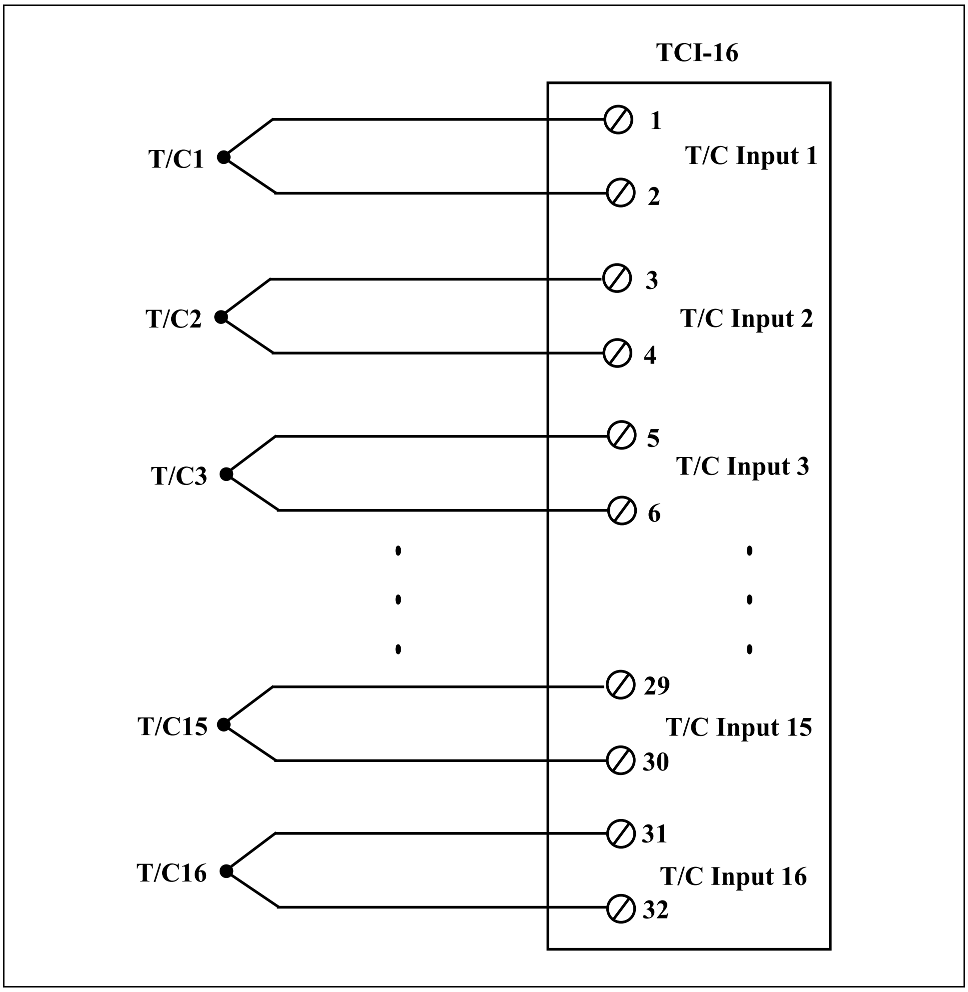


Figure 4-10. Thermocouple Input Module

Resistance Temperature Detector Input Module

In principle, any material could be used to measure temperature if its electrical resistance changes in a significant and repeatable manner when the surrounding temperature changes. In practice, however, only certain metals and semiconductors are used in process control for temperature measurement. This general type of instrument is called a *resistance temperature detector* or RTD. RTDs are the second most widely used temperature measurement device because of their inherent simplicity, accuracy, and stability. The RTD can have two, three, or four electrical lead wires.

A block diagram of four-wire RTD connected to a typical RTD input module is shown in Figure 4-11. When a small current from the RTD module is applied to the RTD, it creates a voltage that varies with temperature. This voltage is processed and converted by the RTD module into a temperature value that can be used and displayed by a process operator.

A typical RTD input module will normally have 4 or 8 input channels and each channel will accept different types of RTDs, such as platinum, nickel, copper and nickel-iron. Most RTD modules also accept other resistance devices like potentiometer. The module converts RTD signals to temperature in °C or °F and convert resistance device inputs to ohms. The RTD module will normally have a removable terminal block that provides for any mix of RTD sensors or resistance input devices.

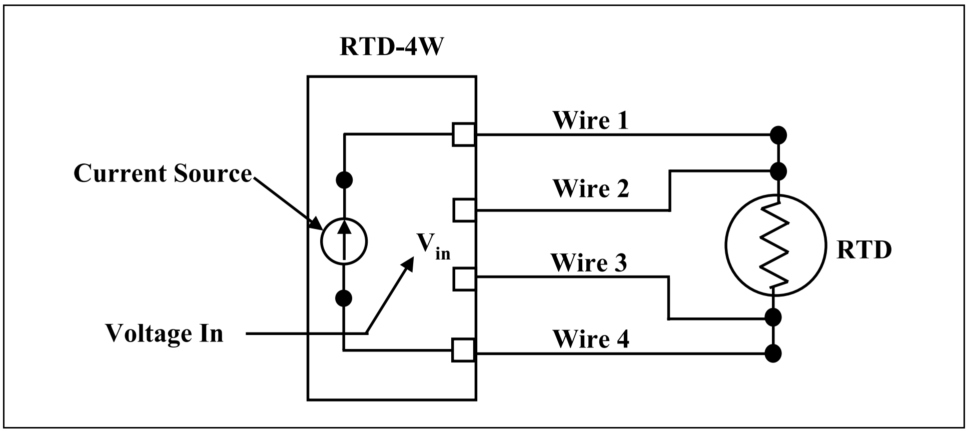


Figure 4-11. Block Diagram of RTD Input Module

Stepping Motor Module

The stepping motor module generates a pulse train that is compatible with stepping motor translators (see Figure 4-12). The pulses sent to the translator normally represent distance, speed, and direction commands to the motor. The stepping motor interface accepts position commands from the control program. Position is determined by the preset count of output pulses; a forward or reverse direction command; and the acceleration or deceleration command for ramping control, which is determined by the rate of output pulses. These commands are generally specified during program control, and, once the output interface is initialized by a start, it will output the pulses according to the PLC program. Once the motion has started, the output module will generally not accept any commands from the CPU until the move is completed. Some modules may offer an override command that will reset the current position, which must be disabled to continue operation. The module also sends data regarding its status to the PLC processor.

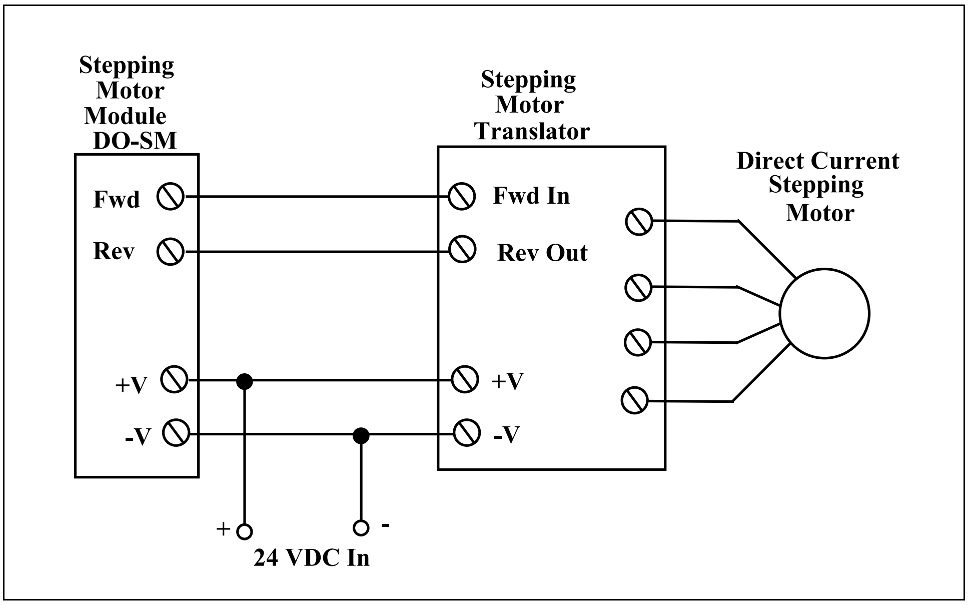

Stepping Motor Module DO-SM
Fwd
Rev
+V
-V
Stepping Motor Translator
Fwd In
Rev Out
+V
-V
Direct Current Stepping Motor
+
-
24 VDC In

Figure 4-12. Typical Stepping Motor Wiring Diagram

Communications Modules

The most common types of communication modules used in PLC systems to communicate between system components are the ASCII interface module, universal remote I/O link module, serial communication modules, PCMCIA interface card, Ethernet interface modules, and fiber-optic converter modules.

ASCII Communications Module

The ASCII communications module is used to send and receive alphanumeric data between peripheral equipment and the controller. Typical peripheral devices with ASCII I/O include printers, digital display instruments, and so on. This special I/O module, depending on the manufacturer, is available with a communications circuitry interface that includes onboard memory and a dedicated microprocessor. The information exchange interface generally takes place via an RS-232C, RS-422, or RS-485 serial interface link, or a 20-mA DC current loop communications link.

The ASCII module will generally have its own RAM, which can store blocks of data that are to be transmitted. When the input data from the peripheral is received at the module, it is transferred to the PLC memory through a data transfer instruction. All the initial communication parameters, such as parity (even or odd) or nonparity, number of stop bits, and communication rate, are hardware selectable or selectable through software.

Universal Remote I/O Link

The remote I/O link modules are used in larger programmable controller systems to allow I/O subsystems to be remotely located from the processor. The remote subsystem is used to interface with a unit process using a standard I/O rack and the required I/O modules. The rack will include a DC power supply to drive the internal circuitry of the I/O modules and a remote I/O adapter module that provides the communications with the processor unit. The I/O capacity of a single subsystem normally ranges from 32 to 256 points.

The subsystems are normally connected to the processor using a bus or star configuration. The distance from the processor to a given remote

I/O rack normally ranges from 1000 feet to several miles, depending on the programmable controller type.

Remote I/O arrangements offer large cost savings on wiring materials (wire and conduit) and labor for large control systems in which the field instrumentation is in clusters at several remote process areas. If the processor is located in a main control room or some other central location, only the communication cable needs to be run between the processor and the field I/O racks instead of hundreds or thousands of field wires.

Remote I/O arrangements also have the advantage of allowing subsystems to be installed and tested independently, as well as allowing maintenance and troubleshooting on individual stations while other units continue to operate.

Serial Communications Module

The serial data communications module is normally used to communicate between the programmable controller and an intelligent instrument with a serial output, such as a weigh scale with a serial communication port. This serial communication module generally has 2 to 4 serial ports to connect to RS-232, RS-422, and RS-485 standard communication interfaces.

PCMCIA Interface Card

The Personal Computer Memory Card International Association (PCMCIA) developed a standard for a credit-card-size PC interface card.

The PCMCIA standard defines an architecture and communications method for these PC interface cards. The interface cards developed under release 2.0 of the standard can be used for both data storage and I/O communication. PLC manufacturers developed PCMCIA cards to be installed in notebook PCs that could communicate with the PLC processor or data highway to perform PLC software and troubleshooting functions.

These PCMCIA interface cards come with diagnostic software to verify proper operation of the card and connection to the PLC communication network.

Ethernet Communications Modules

Ethernet interface modules are designed to allow a number of programmable controllers and other computer-based devices to communicate over a high-speed plant Ethernet communication network. This plant local area network (LAN) is able to transfer data and control information from one system to another at a high data transmission rate. Therefore, the control of an industrial facility can be distributed over a large number of programmable controllers, computers, and intelligent devices. In such a system, information is easily exchanged between control systems, but each system can independently control its part of the industrial plant. This greatly improves the reliability of the plant control system since sections of the plant can be down for modification or maintenance, but the remaining parts of the plant can continue to operate and produce product.

Fiber Optic Signal Converters

The fiber optic converters transform electrical signals to light signals and transmit these signals through fiber optic cables. At the other end of the cable, a second fiber optic converter transforms light signals to electrical signals for use by the PLC system. These modules simply convert from an electrical signal to a light pulse or the reverse from a light signal back to an electrical signal. In most cases these signal converter can save wiring costs and reduce signal noise.

Designing I/O Systems

To correctly design I/O systems, the programmable controller manufacturer's specifications must be consulted and followed to prevent faulty operation or equipment damage. These specifications place limitations not only on the module but also on the field equipment it operates. The specifications fall into three categories: electrical, mechanical, and environmental.

Electrical Specifications

The typical electrical specifications for I/O modules include the following: (1) input voltage rating, (2)input current rating, (3)input threshold voltage, (4) output voltage rating, (5) output current rating, (6) output power rating, and (7) backplane current requirements.

The *input voltage (AC or DC) rating* specification lists the magnitude and type of signal the module will accept. In some cases, a range of input voltages instead of a fixed value is stated in the specification. In this case, the maximum and minimum acceptable working voltages for continuous operation are listed. For example, the working voltage for a 120 VAC input module might be listed as 95 to 135 VAC.

The *input current rating* defines the minimum input current required at the module's rated voltage that the field device must be capable of supplying to operate the input module circuit.

The *input threshold voltage* is the voltage at which the input signal is recognized as being ON or true. Some input modules also have an OFF voltage value at which the input is OFF or false. For example, the ON voltage for TTL input module is defined as any voltage 2.8 VDC or greater and the OFF voltage is defined as any voltage less than 0.8 VDC.

The *output voltage rating* specifies the magnitude and type of voltage source that can be controlled within a stated tolerance. An output module rated at +24 VDC, for example, might have a working range of 20 to 28 VDC.

The *output current rating* defines the maximum current that a single output circuit in a module can safely carry under load. This current rating is normally specified as a function of the output circuit component's electrical and heat dissipation characteristics at an ambient temperature range (typically 0° to 60° C). As the ambient temperature increases, the output current is normally derated. Exceeding the output current rating can result in a permanent short circuit or other damage to the output module.

The *output power rating* specifies the maximum total power that an output module can dissipate with all outputs energized. The output power rating for a single output is calculated by multiplying the output voltage rating by the output current rating. For example, if a 120 VAC output module has a current rating of 2 amps, the power rating is $P = I \times V$ or $P = 2\,A \times 120\,V = 240$ W.

The *backplane current requirement* is the current demand that a particular I/O module internal circuitry places on the rack power supply. The system designer must add up the current requirements of all the installed modules in an I/O rack and compare the calculated value with the maximum current that can be supplied by the system power supply to determine if the power supplied is the correct size. If the rack power supply output is too low, intermittent and faulty operation of the system will result. Some typical specifications for I/O modules are given in Table 4-5.

Model Number	Module Description	Backplane Current	I/O Power Rating
AI-4-20MA	Analog input (4-20 mA)	400 mA	100 mW
AO-4-20MA	Analog output (4-20 mA)	400 mA	100 mW
ACI-120	120 VAC input	200 mA	240 W
ACI-120	220 VAC input	200 mA	440 W
DCI-12	12 VDC input	200 mA	24 W
DCI-24	24 VDC input	200 mA	48 W
DCI-48	48 VDC input	200 mA	96 W
IACI-120	Isolated AC input	200 mA	240 W
ACO-120	120 VAC output	250 mA	240 W
ACO-220	220 VAC input	250 mA	440 W
DCO-12	12 VDC output	200 mA	24 W
DCO-24	24 VDC output	200 mA	48 W
DCO-48	48 VDC output	200 mA	96 W
CC-4NO	4 NO contact output	500 mA	240 W
TTLI	TTL input	150 mA	100 mW
TTLO	TTL output	150 mA	100 mW
IACO-120	Isolated AC output	250 mA	240 mW

Table 4-5. Typical Specifications for I/O Modules

An example problem will illustrate a typical I/O system design application.

EXAMPLE 4-1

Problem: Calculate the backplane current requirements for the programmable controller system with the following modules: five DC input modules, model number DCI-24; three DC output modules, model number DCO-24; two analog input modules, model number AI-4-20MA; and one analog output module mounted, model number AO-4-20MA mounted in a PLC rack. Assume the rack power supply and the backplane power bus are both rated at 5 amps.

Solution: Use the I/O specifications in Table 4-5 to find the total backplane current required, as follows:

(a) DCI-24: 200 mA,

(b) DCO-24: 200 mA,

(c) AI-4-20MA: 400 mA, and

(d) AO-4-20MA: 400 mA.

So that, the total backplane current required is

$$\begin{aligned} 5 \times 200\text{ mA} &= 1000\text{ mA} \\ 3 \times 200\text{ mA} &= 600\text{ mA} \\ 2 \times 400\text{ mA} &= 800\text{ mA} \\ 1 \times 400\text{ mA} &= \underline{400\text{ mA}} \\ &\quad 2800\text{ mA} \end{aligned}$$

Therefore, the total backplane current demand is 2800 mA or 2.8 amps. This is less than the rack power supply and backplane current rating of 5 amps, so I/O system current demand is acceptable. There is also extra current capacity to handle some spare modules in the future.

Mechanical and Environmental Specifications

The typical mechanical specifications are I/O points per module and wire size. The I/O points per module simply define the number of field points that are controlled or sensed by a module. Typically, modules will have 2, 4, 8, 16, or 32 I/O points per module. The higher density modules require higher operating current, so the backplane current must be carefully checked. The number of wires is also increased and it might be a problem for larger gage wires. The wire size specification defines the number of conductors and the largest wire gage that the I/O terminal points will accept. For example, a typical wire specification for an 8-point input or output AC module is 2-14 AWG wires per terminal. However, higher density modules (i.e., 16 and 32 point modules) might only allow a single 14 AWG wire per terminal.

Environmental Specifications

The important environmental specifications are the ambient temperature rating and the humidity rating. The ambient temperature specification defines the maximum temperature of the surrounding air in which the Input/Output system will operate properly. This rating is based on the heat dissipation characteristics of the circuit components inside the I/O modules, which are much higher than the module ambient temperature rating. A typical value of ambient temperature rating is 0°C to 40°C. Exceeding the ambient temperature rating can be dangerous because the internal circuits of the modules will act erratically, resulting in undesirable outputs to the process being controlled. Exceeding the ambient temperature rating will also decrease the life of the module.

The humidity specification is typically 5% to 95% without condensation. The system designer must ensure that the humidity is properly controlled in the control panel where the I/O system is installed.

EXERCISES

4.1 Describe the main purpose of programmable controller I/O system.

4.2 What is the purpose of the backplane on a typical PLC I/O equipment rack?

4.3 Describe the three common I/O system configurations.

4.4 What is a discrete input signal? Give five typical examples of discrete input field devices.

4.5 Give five examples of typical discrete output field devices.

4.6 What is the purpose of the bridge rectifier in the AC input circuit shown in Figure 4-2?

4.7 What is the purpose of the optical coupler in the AC input circuit shown in Figure 4-2?

4.8 Describe the operation of the internal circuits in the typical AC output module shown in Figure 4-5.

4.9 Draw a wiring diagram for a typical +24 VDC input module with four temperature high switches (TSH-100, 101, 102, and 103) and three pressure low switches (PSL-210, 211 and 212) connected to the module.

4.10 Draw a wiring diagram for a typical +120 VAC output module connected to four solenoid valves (TV-100, 101, 102, and 103) and three pump starters (P-100, 200 and 300) with overload switches.

4.11 Describe the basic operation of a four-wire RTD input module circuit.

4.12 Calculate the backplane current requirements for a 16-slot I/O rack with the following modules installed: (a) four 12-VDC input modules (model number DCI-12), (b) six 12-VDC output modules (model number DCO-12), (c) four analog input modules (model number AI-4-20MA), and (d) two analog output modules (model number AO-4-20MA) using backplane current values listed in Table 4-5.

BIBLIOGRAPHY

1. Hughes, T. A., *Basics of Measurement and Control*, ISA – The Instrumentation, Systems, and Automation Society, 3rd edition, 2002.
2. *Control and Information Products Guide,* Rockwell Automation, 2002.

5 Memory and Addressing

Introduction

The three main components of a Programmable Logic Controller (PLC) are the central processing unit (CPU), the I/O system, and memory. In the last chapter, we discussed I/O systems and methods. In this chapter, we will discuss the structure and components of memory, memory hardware devices, types of memory, memory organization, addressing memory, and I/O addressing. Then, PLC hardware to software interface will be covered.

Programmable controller systems store information and control programs in memory. The information and programs stored in memory determine how the input and output data will be processed by the programmable controller.

Memory Components and Structure

Programmable controllers or computer memories can be visualized as a two-dimensional array of storage cells, each of which can store a single bit of data in the form of a logical 1 or a logical 0. This single binary digit or "bit" gets its name from the first two letters of <u>bi</u>nary and the last letter of digi<u>t</u>. A bit is the smallest structural unit of memory and stores information in the form of l's and 0's. Ones and zeros are not actually in each memory cell; each cell has a voltage present at the output of an electronic circuit, indicated by a 1, or voltage close to 0, and indicated by a 0.

The bit is set or on if the stored bit is 1, and off if the stored bit is 0. In most cases, it is necessary for the processor to handle more than a single bit. For example, when transferring data to and from memory, storing numbers, and programming codes, a group of bits called a byte or word is required. A byte is defined as the smallest group of bits that can be handled by the CPU at one time. In programmable controllers, byte size is normally 8 bits and word size is normally 2 bytes or 16 bits but the word size can be smaller or larger depending on the specific microprocessor being used.

Memory capacity is specified in thousands or "K" increments where 1K is 1024 words (i.e., $2^{10} = 1024$) of storage space in most cases. Programmable controller memory capacity may vary from less than 1,000 words to more than 64,000 words (64K words), depending on the programmable controller manufacturer and application. The complexity of the control plan, the number of I/O points, and type of I/O will also determine the amount of memory required.

As mentioned above, word size is usually 2 bytes (16 bits) or more in length. Typical word lengths in programmable controllers are 8, 16, or 32 bits. A 16-bit word is shown in Figure 5-1.

Some programmable controllers use the octal numbering system to identify each bit in a given word as shown in the figure. The most significant bit (MSB) is bit 17, and the least significant bit (LSB) is Bit 00.

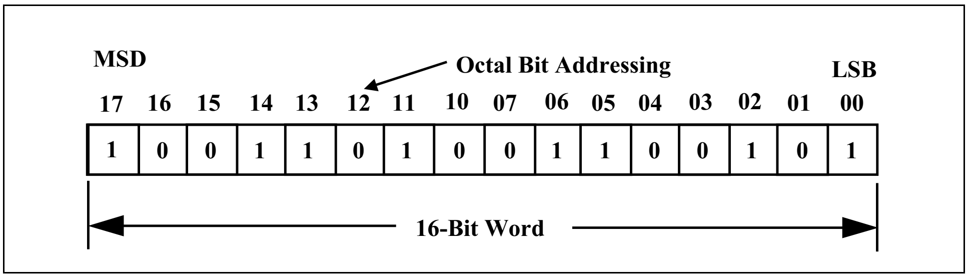


Figure 5-1. Basic Memory Word

A simple 64-cell memory array is shown in Figure 5-2. This array consists of 8 rows and 8 columns. This 64-bit array requires only 6 bits to address a given cell. A cell is normally an electronic circuit called a flip-flop that can have a value of + 5 V (logic 1) or 0 V (logic 0). To retrieve data from the memory array, row and column address decoders select the appropriate cell.

These memory arrays are normally provided by integrated circuits (IC). A typical IC unit contains many thousands of memory cells arranged in various ways. An 8K-bit (8096 cells) memory IC, for example, could be arranged as 8K memory cells of 1 bit each, or 1K bytes of 8 bits each. The number of groups (bits, bytes, or words) addressed is a function of 2^n, for example 1K = 2^{10}, 2K = 2^{11}, 4K = 2^{12}, 8K = 2^{13}, etc. The value *n* is the

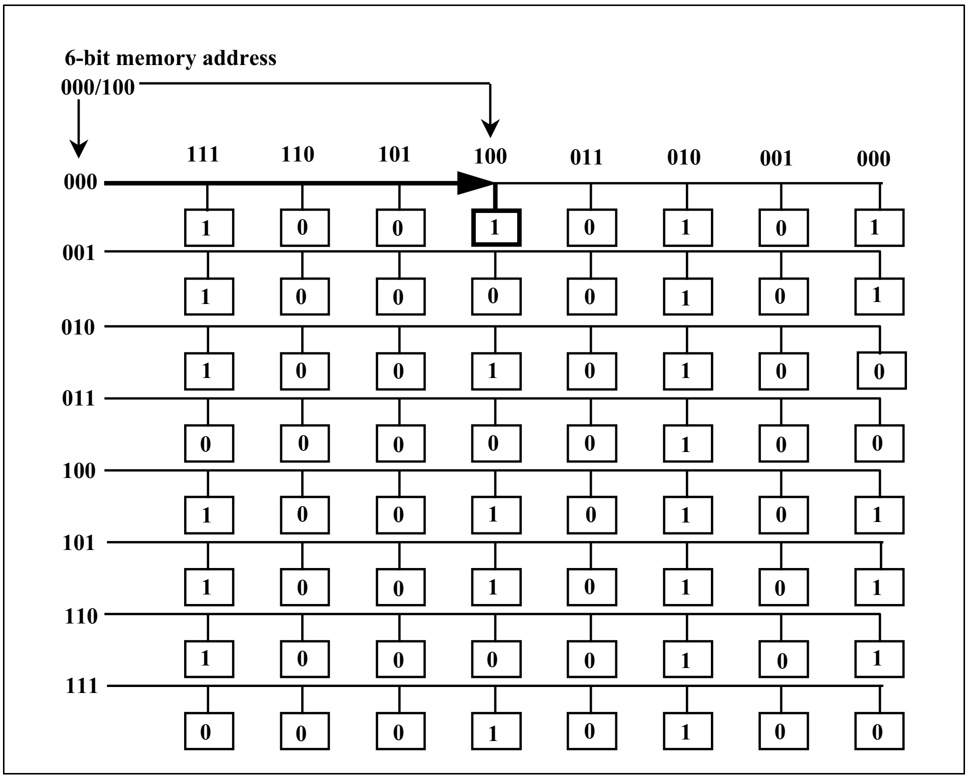


Figure 5-2. Typical Memory Array

number of address bits needed to select each separate group. For 1000 words, it is necessary to use 10 bits to address each word of storage, the word size being 8, 16, and 32 bits. For a 1K x 8 memory, the IC would require 10 address bits to select 1K words of memory. A typical 1K x 8-bit word is shown in Figure 5-3. The IC has 8 pins for input and output data bits, 10 connection pins for selecting addresses, 2 pins for control signals (chip enable and read/write), and 2 pins for DC power. The two power supply pins are used for +5 VDC and ground. The read/write (R/W) control signal is used to determine whether the data bits are read into memory (R/W signal low) or data is sent out from memory (R/W is high). The chip enable control signal is used to select operation of each separate chip when a group of integrated circuits is used to provide a larger memory than is provided by only one chip. These read/write IC chips are also called Random Access Memory (RAM) chips because the data at any location on the memory chip can be accessed at any time. The data on some earlier memory storage devices such as punch paper tapes or magnetic tape units had to be read or written in a sequence manner.

A typical RAM chip used in PLC applications is shown in Figure 5-3. This RAM chip stores individual bits of data in multiple rows and columns of cells. The row and column arrangement allows each cell to have a unique designation, called an *address*. This address consists of a row identifier and a column identifier, both expressed as binary numbers.

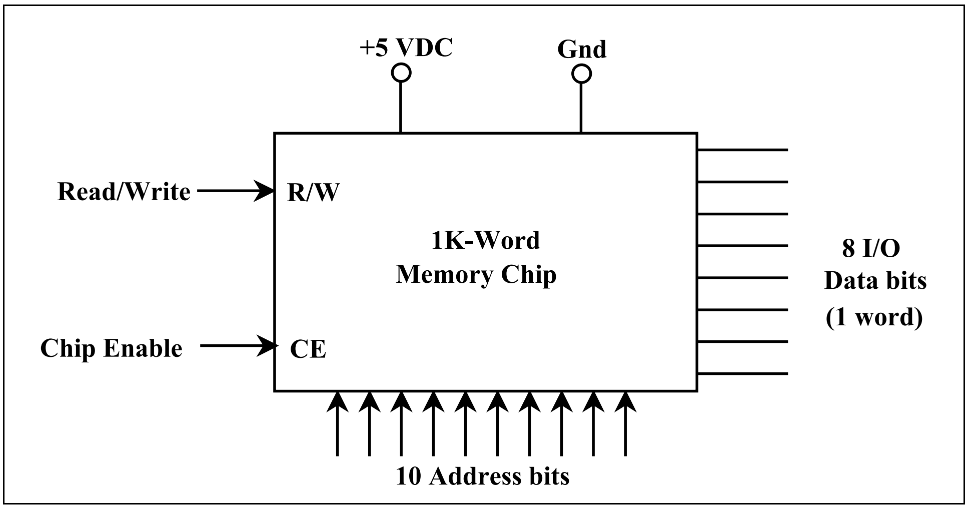


Figure 5-3. Typical 1K-byte RAM Chip

Memory Types

This section will discuss the types of memory generally used in programmable controllers and their applications to the type of data or information stored. In selecting the type of memory to be used, a system designer is concerned with volatility and ease of programming. He is concerned with volatility because memory holds the process control program, and, if this program is lost, production in a plant will be down. Ease in altering the memory is important since the memory is involved in any interaction that takes place between the user and the PLC. This interaction begins with the initial system programming and debugging and continues with on-line changes, such as changing timer and counter preset values.

The most common type of memory used in PLC applications is ICs or chips so they will be discussed first. Other memory storage devices such as floppy disks, hard disks and compact disks (CDs) will also be discussed.

Random Access Memory

RAM is designed so that data or information can be written into or read from any unique memory location in the programmable controller. Figure 5-4 shows that data can be placed into the RAM IC memory chip using the write mode and data can be retrieved from the RAM chip using the read mode. The address input to the RAM chip specifies the location or the address of the data to be read or the location where the information or data is to be stored in memory.

Programmable controllers, for the most part, use RAM with battery backup for application memory. RAM chips provide an excellent means for easily creating and altering a control program as well as allowing data entry. In comparison to some other R/W memory types, such as hard drives, or CDs, RAM chips are relatively fast. The only important disadvantage of battery-supported RAM chips is that they require a battery that might fail at a critical time, but most programmable controllers have battery low lights to alert operations personnel to replace the memory power backup battery and they use long life, rechargeable batteries.

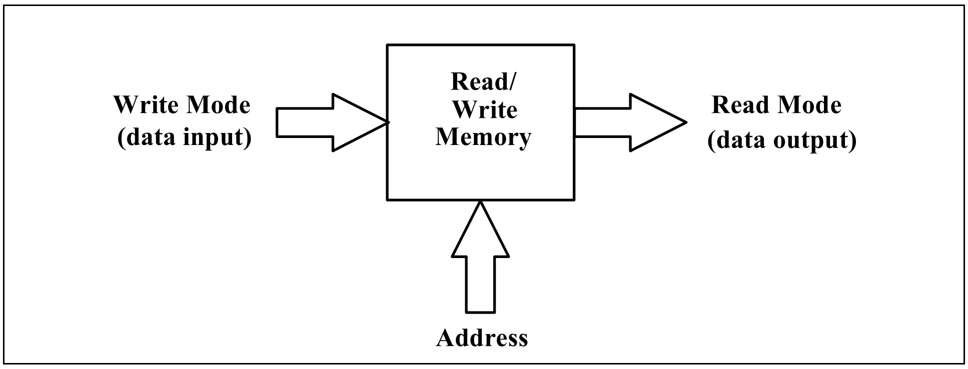


Figure 5-4. RAM Chip Block Diagram

Read-Only Memory

Read-only memory (ROM) is designed to permanently store a fixed program that normally cannot or will not be changed. It gets its name from the fact that its contents can be read only but not written into or altered once the data or program has been stored. Figure 5-5 shows that data can be used only in the read mode. As with RAM chips, a ROM chip has an address input to specify the location of the data to be read. Because of their design, ROMs are generally immune to changes due to electrical noise or loss of power. The executive or operating system program of a PLC is normally stored in ROM.

Programmable controllers rarely use ROM for the control applications program memory. However, in applications that require fixed data, ROM offers advantages where speed, cost, and reliability are factors. Generally, ROM-based PLC programs are produced at the factory by the equipment manufacturer. Once the original set of instructions is programmed, the

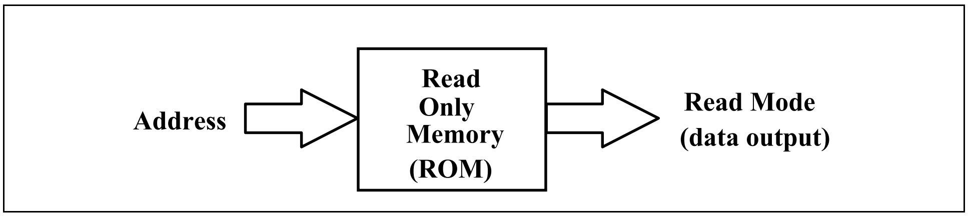


Figure 5-5. ROM Block Diagram

user can never alter it. The manufacturer will write and debug the program using a read/write-based controller or a PC and then the final program is entered into ROM. ROM is also found in the application memory in dedicated programmable controller systems such as microwave ovens, vending machines, clothes washers, and so on.

Programmable Read-Only Memory

Programmable Read-Only Memory (PROM) is a special type of ROM that is rarely used in most programmable controller applications. However, when it is used, it will most likely be a permanent storage to some type of control program that is rarely changed or updated. Although PROM is programmable and, like other ROM, has the advantage of nonvolatility, it has the disadvantages of requiring special programming equipment, and, once programmed, it cannot be erased or altered. Any program change would require a new set of PROM chips. PROM might be suitable for storing a control program that has been thoroughly checked while stored in RAM and would rarely require further changes or on-line data entry after installation.

Electrically Erasable Programmable ROM

The electrically erasable programmable ROM (EEPROM) is a special type of PROM that can be reprogrammed after it is completely erased using an electrical method. The EEPROM can be considered a temporary storage device in that it stores a program until it is ready to be changed. EEPROM provides an excellent storage medium for a control program where nonvolatility is required but program changes are not normally required during the lifetime of the equipment. Many manufacturers of equipment with built-in PLCs use EEPROM-type memories to provide permanent storage of the machine program after it has been developed, debugged, and are fully operational.

A control program composed of EEPROM alone would be unsuitable if online changes and/or data entries are a frequent requirement. However, many PLCs offer EEPROM control program memory as an optional backup to battery-supported RAM. EEPROM, with its permanent storage capability combined with the easily altered R/W memory, makes a suitable memory system.

Memory Storage Devices

While IC chips such as RAM and ROM are used for PLC internal memory, other storage devices such as magnetic floppy diskettes (disks), magnetic hard disks, and optical CDs are generally used for external programs and data storage in PLC applications.

Magnetic Disk Storage

Information is stored on a magnetic disk much as music or video is recorded on magnetic tape by magnetizing small areas on the disk in a predefined direction and manner. One direction represents a logical 1 bit, and the opposite direction represents a logical 0 bit. There are two main magnetic memory storage systems used in PLC systems; the floppy disk and the hard drive type disk. Floppy disks and hard drives have similar structures; if we think of a floppy disk as a two-dimensional version of a hard drive, the similarities are very apparent. The floppy disks are small (3.5 and 5.25 inches in diameter) and portable storage mediums but the fixed hard drives have the advantage of very high data storage capacity.

A typical floppy disk consists of a disk of thin plastic coating on both sides with magnetic material with a protective plastic jacket. The magnetic coating is accessible through an opening in the plastic jacket. A hole in the center of the disk goes around a disk drive motor, which spins the disk so that data can be written or read. The standard floppy disk is 3.5 inch in diameter with a storage capacity of 1.44 megabits (Mb). It has 80 recording tracks per side with 18 sectors per track.

A hard drive is simply a group of magnetic storage plates stacked on top of each other. Each magnetic plate is similar to a floppy disk; it has two sides, is divided into tracks, and each track is subdivided into sectors. Above the surface of each side of the plate is a magnetic R/W head that accesses the data. The plates are carefully aligned so that track 0 on one of the plates is exactly above track 0 of another plate. A mechanical arm links all the R/W heads together. To access a particular track on anyone of the plates, the arm moves all the R/W heads to the specific track. Since this arrangement requires only a single positioning mechanism, it simplifies the design and lowers the size and cost of the hard drive. However, with this arrangement, all the R/W heads must be moved to access data on a different track. For example, to read data on track 20 of one plate, then data on track 40 of a different plate, and finally data on track 20 again on the first plate, the entire R/W arm must be moved twice. Positioning the mechanical arm this way requires a significant amount of time compared to accessing data on electronic RAM or ROM IC memory chips.

Compact Disk Memory Storage

CDs have become one of the most popular storage medium for both PLC programming software and human machine interface (HMI) graphical software because of the large storage space available on a CD. A typical vendor-supplied PLC programming software package might require 10 to 20 floppy disks but normally only a single CD.

A CD has a reflective layer of aluminum that is coated by a protective layer of clear paint. When a typical read-only CD is manufactured, the information and software programs are pressed into the layer of the aluminum in the form of pits (indentation) and lands (elevations), which represent the individual data bits. The pits and lands are arranged along a single spiral that covers the entire CD, winding from the inside of the disk to the outside. Because the pits and lands are only about 0.6 micron (0.6 millionth of a meter) wide, the path this spiral takes is only separated by the microscopically small distance of about 1.6 microns. The track density of a typical CD is almost 16,000 tracks per inch (TPI), this compares to about 135 TPI on a 3.5-inch high-density floppy disk. If the data storage spiral on a typical CD were stretched out in a straight line, it would be approximately 6 kilometers (3.75 miles) long. It typically contains over 2 billion data bits. Naturally, the laser beam used to read the lands and pits of data must be correspondingly narrow. The scanning laser beam is typically about 1 micron in diameter, which makes it only a little wider than the wavelength of the light that forms its beam.

Since all data on a CD is contained on a single continuous spiral, it consists of a single data channel. So data bits on a CD are referred to as "channel bits." The transition from a land to a pit or the transition from a pit to a land represents a binary one. The lands and pits are used to represent a binary zero. The length of a land or the length of a pit determines how many binary zeros are represented. This is basically the same procedure used to record data on magnetic storage devices like hard disks and floppy disks. The only difference is that magnetic flux changes replace the pits and lands.

Memory Organization

There are two main parts of typical PLC memory: the *system memory* and the *application memory*. The *system memory* is a permanently stored collection of programs and registers that consists of the operating system program, diagnostics software, and system status registers. The operating system directs activities such as execution of the control program, communication with the peripheral devices, and other system housekeeping functions. The *application memory* consists of the input area, output area, data or information storage registers, internal storage bit area, and the control program.

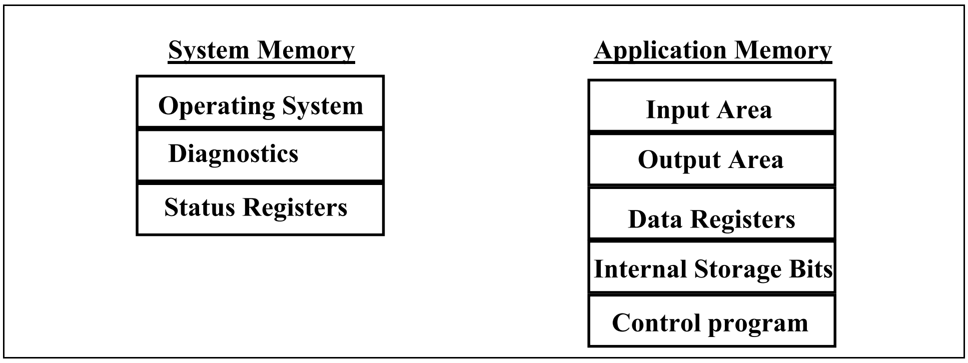


Figure 5-6 Typical PLC Memory Organization

The storage and retrieval requirements are not the same for the system memory and the application memory. The system memory contains the instructions to operate the CPU, a set of diagnostics programs, and status registers. The application memory contains the input image area (designated by an "I" in most PLCs), the output image area (designated by an "O" or "Q" in most PLCs), the control program, and the data registers. They do not use the same types of memory. The operating system portion of the system memory requires a memory that permanently stores its contents and cannot be deliberately or accidentally altered by loss of electrical power or by the user; therefore, some type of ROM would be used. On the other hand, the user would need to alter the control program and/or the I/O data for any given application, so RAM is used in the I/O data and the control program sections of memory.

File Structure

A memory file is defined as a group of words in memory that has a dedicated function. The most common type of files is the input and output files in a PLC. They are the memory words set aside for the external input and output bits assigned to the physical I/O points in a PLC system. The file structure varies with the manufacturer. The file structure for Allen-Bradley PLC5s is listed in Table 5-1. There are 9 file types listed, but additional files can be assigned as needed.

File number 0 is for the output image table and it has the file letter of O. File 1 is the input image table area of memory and it has the file letter of I. File 2 is reserved for the status bits and words in the PLC. There can be a maximum of 32 elements or words in input and output image tables and the status file areas.

File Number	File Letter	File Name	Maximum Words
0	O	Output Image	32
1	I	Input Image	32
2	S	Status	32
3	B	Bits for program use	1000
4	T	Timers	1000
5	C	Counters	1000
6	R	Control Registers	1000
7	N	Integer	1000
8	F	Floating Point	1000

Table 5-1. A-B PLC5 Memory File Structure

A partial listing of memory file structure for Siemens Simatic S7 PLCs is given in Table 5-2 for comparison. In both Allen-Bradley and Siemens PLCs, the input image bits are implemented with an "I" in programming. The output image bits are designated by an "O" in Allen-Bradley PLC5s and a "Q" in Siemens S7 PLCs. Timers and counters are designated by T and C respectfully in both systems but that is where the similarities in file letter designation end between the two systems.

Internal Storage Bits

Most programmable controllers assign an area for internal storage bits. These storage bits are also called internal outputs, internal coils, or internal control bits. The internal output operates just as any output that is controlled by programmed logic; however, the output is used strictly for internal logic programming and does not directly control an output to the process. Internal outputs are used for interlocking logic purposes in the control program.

Internal outputs include the "done" bits on counters and timers as well as internal logic bits of various types. Each internal output bit, referenced by an address in the control program, has a storage bit of the same address. When the control logic is TRUE, the internal (output) storage bit turns ON.

User Program Memory Area

The user program memory area of the application memory is used to store the process control logic program. All of the controller instructions that control the machine or process are stored here. The addresses of the real

File Letter	File Type	Description	Max. Address Range
I	Input Image	Input bit	0.0 to 65535.7
IB		Input byte	0 to 65535
IW		Input word	0 to 65534
ID		Input double word	0 to 65532
Q	Output Image	Output bit	0.0 to 65535.7
QB		Output byte	0 to 65535
QW		Output word	0 to 65534
QD		Output double word	0 to 65532
M	Bit Memory	Memory bit	0.0 to 255.7
MB		Memory byte	0 to 255
MW		Memory word	0 to 254
MD		Memory double word	0 to 252
PIB	External Inputs	Periph. Input byte	0 to 65535
PIW		Periph. Input word	0 to 65534
PID		Periph. In double word	0 to 65532
PQB	External Outputs	Periph. Input byte	0 to 65535
PQW		Periph. Input word	0 to 65534
PQD		Periph. Out double word	0 to 65532
T	Timers	Timer	0 to 255
C	Counters	Counter	0 to 255
DBX	Data block	Data bit	0.0 to 65535.7
DBB		Data byte	0 to 65535
DBW		Data word	0 to 65534
DBD		Data double word	0 to 65532

Table 5-2. File Structure for Siemens Simatic S7 PLCs

and internal I/O bits are specified in this section of memory. When the PLC is in the Run mode and the control program is executed, the CPU interprets these memory locations and controls the bits in the data table that corresponds to a real or internal I/O bit. The interpretation of the control program is accomplished by the processor's execution of the control program.

The maximum amount of user program memory available is normally a function of the controller memory size. In medium and large programmable controllers, the user program size is normally flexible through altering the data table size so that it meets the minimum data storage requirements. In small PLCs, however, the user program size is normally fixed.

Application Memory Size

The size of the application memory is an important factor in designing programmable controller-based control systems. Specifying the correct memory size can save hardware costs and avoid lost time later. Proper calculation of memory size means avoiding the possibility of purchasing a PLC that does not have adequate capacity or that is not expandable.

Application memory size may be expandable to some maximum point in some controllers, but it is not expandable in some of the smaller PLCs. (Smaller PLCs are defined in this context as units that control 10 to 64 I/O devices.) Programmable controllers that handle 64 or more I/O devices are usually expandable in increments of 1K, 2K, 4K, etc. (Recall that each K represents 1024 or 2^{10} word locations in memory.) The memory in the larger controllers is generally 64K or higher.

The stated memory size of a PLC is only a rough indication of the memory space available to the user, since some of the memory is used by the controllers for internal and diagnostic functions.

The main problem with determining memory size for an application is that the complexity of the control program is not determined until after the equipment is purchased. However, we generally know the number of I/O points in the system before the hardware is procured. Application memory size can be estimated by multiplying the number of I/O points by 20 words of memory. For example, if the system has 100 I/O points, the program will generally be equal to or less than 2000 words. We should keep in mind that program size is affected by the sophistication of the control program. If the application requires data handling or complex control algorithms, such as PID control, then additional memory will be required.

After the system designer determines the minimum memory required for an application, he would normally add an additional 25% to 50% more for program changes, modification, or future expansion.

I/O Addressing

Since one of the main purposes of a PLC is to control I/O of field devices, these I/O must occupy a location in the processor memory so they can be addressed in the PLC control program. Each terminal on an input or output module that can be wired to a field device occupies a bit within PLC memory. The part of memory that houses I/O addresses is called input image table and output image table.

EXAMPLE 5-1

Problem: Determine memory size needed for a programmable controller system with 175 input points and 125 output points. Assume that 25% spare memory capacity is required for the system.

Solution: Total I/O points = 175 + 125 = 300 points

Memory required = 300 points x 20 words/points + 25%

= (6000 + 6000 x 0.25) words

= (6000 + 1500) words

= 7.5 K words

Input Image Table

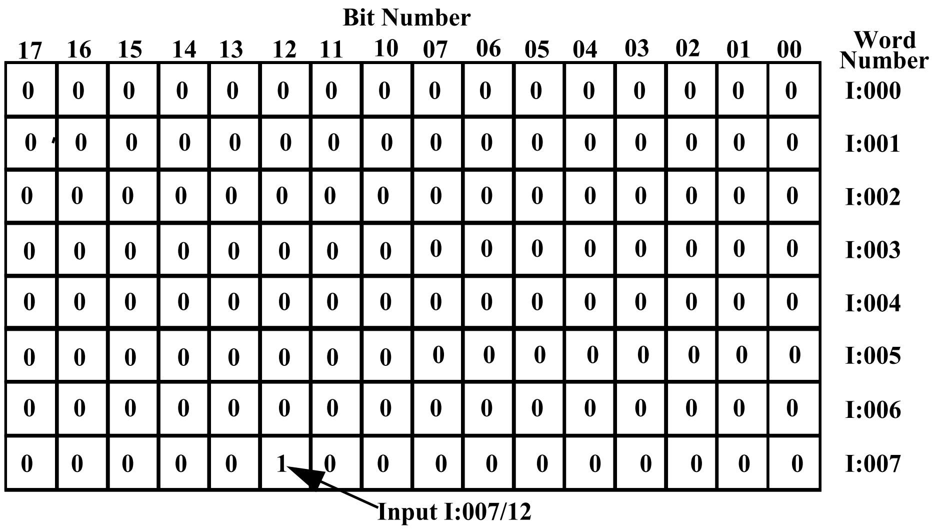
The input image table is an array of bits that stores the status of discrete inputs from the process that are connected to inputs. The number of bits in the table is equal to the maximum number of inputs. A controller with a maximum capacity of 64 inputs would require an input table of 64 bits. Each connected input has a bit in the input table that corresponds exactly to the terminal to which the input is connected. If the input is “on”, its corresponding bit in memory is “on” or logical 1. In most programmable controllers, the input is “on” if there is voltage present on the input terminal. If no voltage is present, the corresponding bit is cleared or turned “off” or set to logical 0. The input table is continuously being changed to reflect the current status of the connected input devices. This status information is used by the control program to determine the true or false state of the program instructions.

Figure 5-7 shows a typical example of a single input bit in an input image table. In this example, input point I:007/12 is identified on the memory map.

Output Image Table

The output image table is an array of bits that controls the status of discrete output devices that are connected to output interface circuits.

Each connected output has a bit in the output image table that corresponds exactly to the terminal to which the output is connected. The PLC processor controls the bits in the output table as it interprets the control program and is updated accordingly during the I/O scan. If a bit is turned “on” or logic 1, then the connected output electrical circuit is


Bit Number
Input I:007/12

17	16	15	14	13	12	11	10	07	06	05	04	03	02	01	00	Word Number
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I:000
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I:001
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I:002
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I:003
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I:004
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I:005
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	I:006
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	I:007

Figure 5-7. Input Bit Example in an Allen-Bradley PLC5 Input Image Table

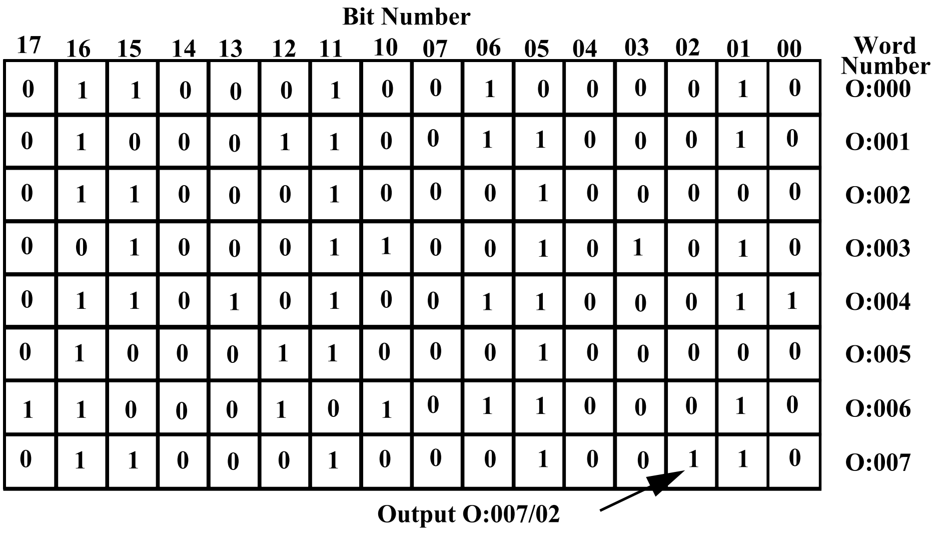
activated and there is a voltage present on the output terminal. If a bit is cleared or turned "off" or logic 0, the output is switched off.

Figure 5-8 shows a typical example of a single output bit in an output image table. In this example, output bit location O:017/16 is shown on the output memory map. The letter "O" indicates an output, the word address is 017 and the bit is 16 or the last bit in the memory word.

Hardware-to-Software Interface

Probably one of the more important things to understand about programmable controllers is how process inputs are sensed by the hardware input circuits, and then used by the control logic program to activate output devices to control a process or machine connected to the PLC. This hardware-to-software interface occurs in the I/O image tables and was briefly discussed earlier. The I/O image bit address is what connects the software control program to the hardware I/O terminations.

The addressing of the PLC I/O connects the physical location of an I/O module terminal to a memory bit location. The structure or density of an I/O module (i.e., 8-, 16-, or 32-point), relates directly to the bits the module occupies in PLC memory. For example, in the example 8-point input module as shown in Figure 5-9, the 8 bits, 00 through 07 occupy 8 positions in an input memory table in the processor memory.



Bit Number

17	16	15	14	13	12	11	10	07	06	05	04	03	02	01	00	Word Number
0	1	1	0	0	0	1	0	0	1	0	0	0	0	1	0	O:000
0	1	0	0	0	1	1	0	0	1	1	0	0	0	1	0	O:001
0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	O:002
0	0	1	0	0	0	1	1	0	0	1	0	1	0	1	0	O:003
0	1	1	0	1	0	1	0	0	1	1	0	0	0	1	1	O:004
0	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	O:005
1	1	0	0	0	1	0	1	0	1	1	0	0	0	1	0	O:006
0	1	1	0	0	0	1	0	0	0	1	0	0	1	1	0	O:007

Output O:007/02

Figure 5-8. Output Bit Example in an Allen-Bradley PLC5 Output Image Table

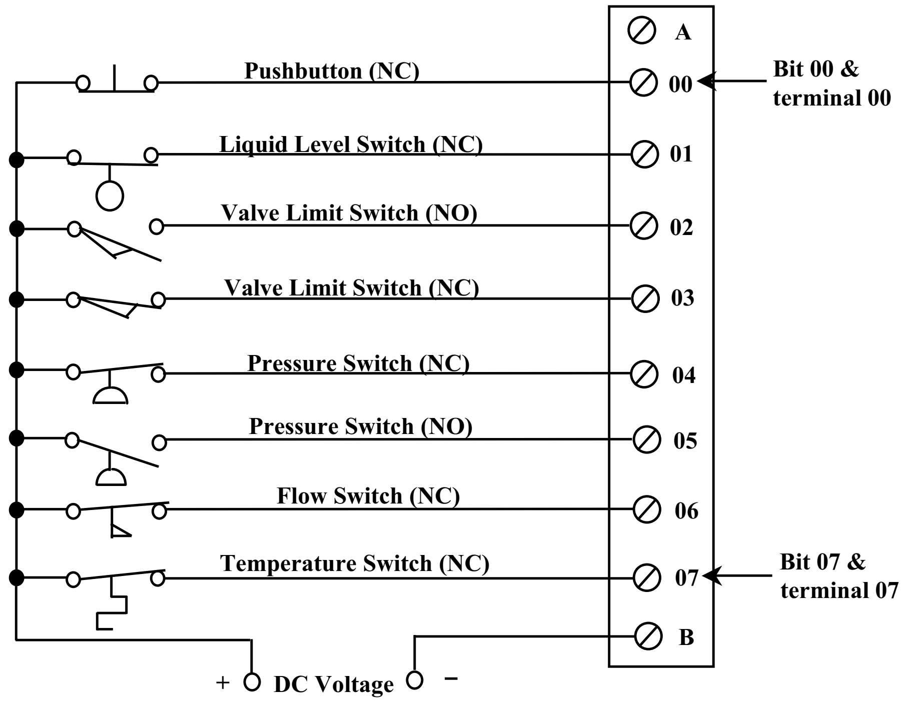


Figure 5-9. Typical 8-Point DC Input Module

We will present several typical I/O addressing methods used in programmable controller systems. The first addressing scheme to be presented is used in the A-B PLC5 family of PLCs.

Allen-Bradley PLC5 Discrete I/O Addressing

The A-B PLC5 Discrete I/O addressing method uses a 6-position code (a:bbc/dd) to reference both an I/O memory address and a hardware physical location. In this system, the left-most position (a) is the letter "I" for a discrete input and the letter "O" for a discrete output. The next two digits (bb) are the rack number. The next digit (c) is the I/O group number (0-7). The remaining two digits represent the input or output bit or terminal numbers, 00-07 or 10-17.

For example, the address I:001/07 indicates an input device connected to rack 00 and I/O group number 1 at terminal 07 and memory bit 07. The address O:074/10 indicates an output device connected to rack 07 at terminal 10 in I/O group 4.

A typical example will be used to illustrate the Allen-Bradley PLC5 addressing scheme.

EXAMPLE 5-2

Problem: List the memory bit address for a discrete signal input connected to an A-B PLC5/ 15 rack 03, I/O group number 4, and terminal 10.

Solution: I:034/10

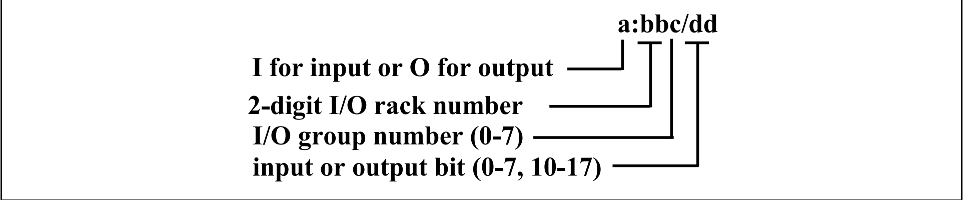


Figure 5-10. A-B PLC5 Discrete I/O Addressing Scheme

A hardware-to-software interface for an Allen-Bradley PLC5 application is illustrated in Figure 5-11. This application shows the operational relationship between the field devices, the discrete I/O memory bits, and the user ladder logic program.

In the example shown in Figure 5-11, if the high-level switch connected to an input module in rack 0, I/O group 0, and terminal 7 is closed, the internal software bit I:000/07 will be set to 1. The dotted line from terminal 7 to memory bit location I:000/07 indicates a connection within the PLC5 system. If, at the same time, the valve open position switch that is connected to terminal 13 of the same input module is closed, then input bit I:000/13 will be set to 1 and the ladder logic rung will have logic continuity and output bit O:001/03 will be set to 1. This will energize the solenoid valve connected to terminal 3 of rack 0 and I/O group 1.

The ladder logic rung in the bottom of Figure 5-11 shows an example of two external input bits being used to set an external output bit. This is the most basic case but it is not common application encountered. In most cases, the control program is more complex and uses both external input bits and internally set bits to control outputs. For example, the level of a process tank might be measured using a level instrument that continuously inputs the tank level. This analog input can be compared to an operator selected level (i.e., 80% of full) and an internal bit can be set by the control program if the tank level equal or exceeds this set value. So that the high-level external bit could be replaced by an internal high-level bit set by the software comparison operation.

Siemens Simatic S7-300 Discrete I/O Addressing

In this section, we will discuss the discrete I/O addressing for Siemens Siematic S-7 family of programmable controllers for comparison and contrast to the Allen-Bradley PLC5 discrete I/O addressing.

Figure 5-12 shows an 11-slot equipment rack for a Siemens Simatic S7-300 PLC. The slot numbers in the rack influence the addressing scheme for the S7-300 family of PLCs. The first I/O address in the module is determined by its location in the rack. The first slot is reserved for the rack power supply and there are of course no I/O addresses needed for the power supply. The CPU must be located next to the power supply in slot number 2 and no I/O addresses are assigned to the CPU. Slot number 3 in Figure 5-12 contains a communications interface module (CIM). This module is used to connect the CPU in the main I/O rack with the I/O in the expansion racks. No I/O addresses are assigned to the interface module in slot 3. Even if no interface module is installed in slot 3, it cannot be used for other I/O modules. In other words, slot 3 in the Simatic S7 is logically reserved in the CPU for communication interface modules. Slot number 4 is the first slot used for I/O signal modules.

The relationship between physical rack and slot position and the I/O module position is shown in Figure 5-13. The discrete addresses for both

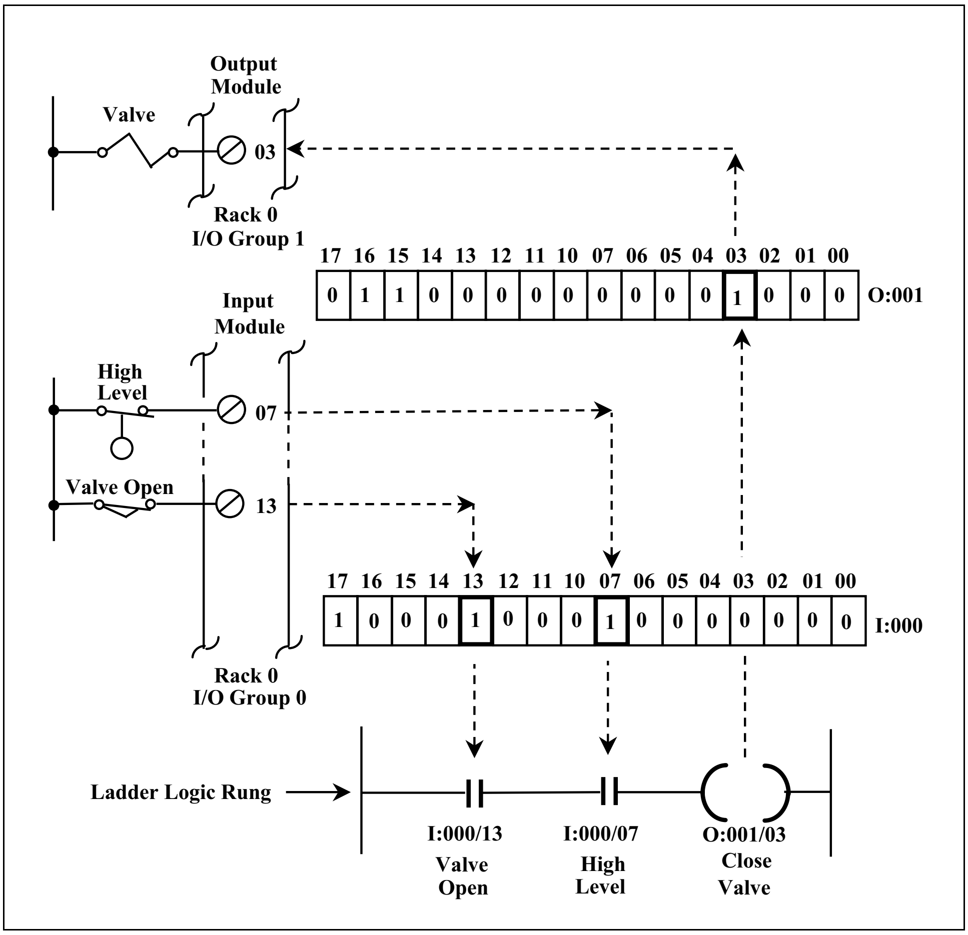


Figure 5-11. Hardware-to-software Interface Diagram

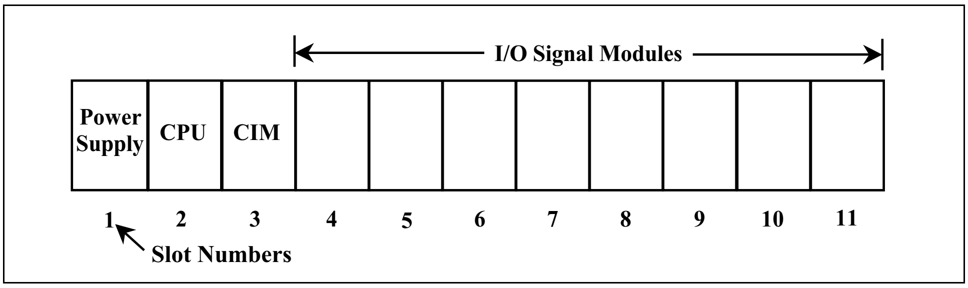
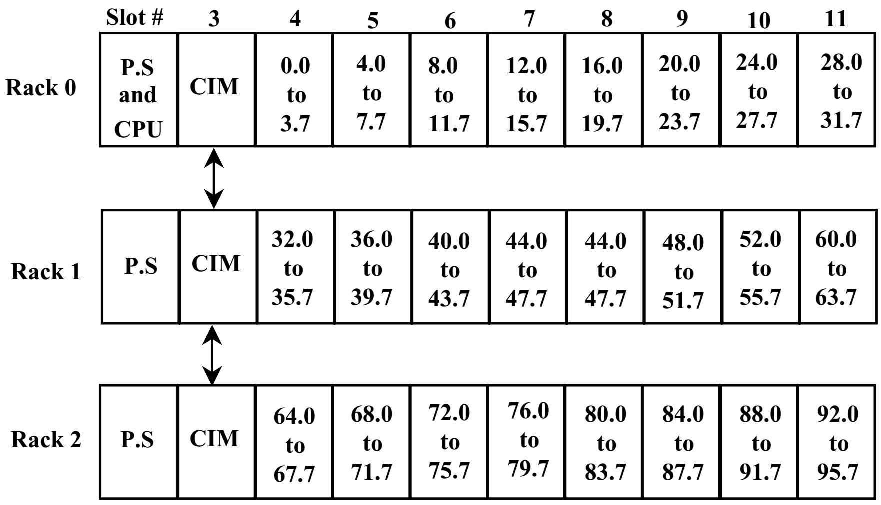


Figure 5-12. Siemens S7-300 Rack Arrangement and Slot Assignments

inputs and outputs begins with word address 0 and bit address 0 (0.0) in slot 3 of rack 0 and continue to word 95 and bit 7 in slot 11 of rack 2. Each discrete I/O module is allocated 4 bytes (32 bits) of a memory word addresses; regardless of the actual I/O point count of the module.

The Siemens discrete I/O addressing method uses a three-position code (abb.cc) to reference an I/O memory address. In this system, the left-most position (a) is the letter "I" for a discrete input and the letter "Q" for a discrete output. The next two digits (bb) are the memory byte number assigned to the I/O slot and the digit (c) to the right of the dot is the I/O image table bit number (0 through 7).

For example, if the I/O module in slot 5 in Rack 0 is a 16 discrete input module the first 8 inputs would be assigned the addresses of 4.0 to 4.7 and the second 8 inputs would be assigned the bit addresses of 5.0 to 5.7 as shown in Figure 5-13.



Slot #		3	4	5	6	7	8	9	10	11
Rack 0	P.S and CPU	CIM	0.0 to 3.7	4.0 to 7.7	8.0 to 11.7	12.0 to 15.7	16.0 to 19.7	20.0 to 23.7	24.0 to 27.7	28.0 to 31.7
Rack 1	P.S	CIM	32.0 to 35.7	36.0 to 39.7	40.0 to 43.7	44.0 to 47.7	44.0 to 47.7	48.0 to 51.7	52.0 to 55.7	60.0 to 63.7
Rack 2	P.S	CIM	64.0 to 67.7	68.0 to 71.7	72.0 to 75.7	76.0 to 79.7	80.0 to 83.7	84.0 to 87.7	88.0 to 91.7	92.0 to 95.7

Figure 5-13. Typical Siemens S7-300 Discrete I/O Address Configuration

A typical example will be used to illustrate the addressing method used for Siemens S7-300 programmable controllers.

EXAMPLE 5-3

Problem: List the input bit addresses available for use on a 32-bit discrete input module installed on the Siemens S7-300 system shown in Figure 5-13 into slot 10 of rack 2.

Solution: Discrete input can be connected to I/O points: 88.0 to 88.7, 89.0 to 89.7, 90.0 to 90.7, and 91.0 to 91.7.

EXERCISES

5.1 List the most common types of memory used in programmable controller applications.

5.2 Describe the operation of a typical random access memory integrated circuit.

5.3 What is the main purpose of read-only memory in PLC applications?

5.4 List the most common integrated circuit memory devices used in PLC applications.

5.5 What are the most common magnetic memory storage devices used in PLC control applications?

5.6 Describe the function and purpose of the two main parts of programmable controller memory.

5.7 What are the input and output image tables used for in programmable controllers?

5.8 What is the memory size needed for a programmable controller system with 235 input points and 195 outputs? Assume that 25% spare memory capacity is required in the application.

5.9 What is the memory bit address for a discrete field device connected to an Allen-Bradley PLC5 input module in rack 1 at I/O group 3 and terminal 1?

5.10 What is the memory bit address for a discrete field device connected to an Allen-Bradley PLC5 output module in rack 2 at module group 1 and terminal 7?

5.11 What is the memory bit address for a discrete field device connected to an Allen-Bradley PLC5 output module in rack 2 at I/O group 2 and terminal 13?

5.12 List the bit addresses available for use by a 16-bit discrete output module installed in slot 4 of rack 1 of the Siemens S7-300 system shown in Figure 5-13.

5.13 List the bit addresses available for use by a 16-bit discrete input module installed in slot 5 of rack 2 of the Siemens S7-300 system shown in Figure 5-13.

6 Ladder Diagram Programming

Introduction

Programming languages let the user communicate with the programmable logic controller (PLC) via a programming device and software package. PLC manufacturers use several different programming languages, but they all convey to the system, by means of instructions, a basic control plan or software program.

A software program is written by combining instructions in a certain order. Rules govern the manner in which instructions are combined and the actual form of the instructions. These rules and instructions combine to form a software language.

The three most common types of software languages encountered in programmable controller systems are: Ladder Diagram, Instruction List, and Function Block Diagram. The Ladder Diagram (LD) is the most common programmable controller language, and it consists of a set of instructions that will perform the most basic type of control functions: relay type logic, timing and counting, and basic math operations. However, depending on the programmable controller model, the instruction set may be extended or enhanced to perform other operations. These additional functions are used for analog control, data manipulation, reporting, complex control logic, and other functions.

We will discuss ladder diagram in this chapter, as well as a typical PLC programming software package and control program documentation.

Basic LD Instruction Set

LD language is a symbolic instruction set that is used to create a programmable controller program. It is composed of six categories of instructions that include relay type, timer/counter, data manipulation, arithmetic, data transfer, and program control. The ladder diagram instruction symbols can be formatted to obtain the desired control logic that is to be entered into memory.

The main function of the LD program is to control outputs based on logical combinations of internal and external logical bits. This control is accomplished through the use of what is referred to as a ladder rung. Figure 6-1 shows the basic structure of a ladder logic program. A ladder logic rung consists of a set of input conditions (internal and external) represented by relay contact-type instructions and an output instruction at the end of the rung represented by the relay coil symbol.

Coils and contacts are the basic symbols of the LD instruction set. The contact symbols programmed in a given rung represent conditions to be evaluated in order to determine the control of the output; all discrete outputs are represented by coil symbols.

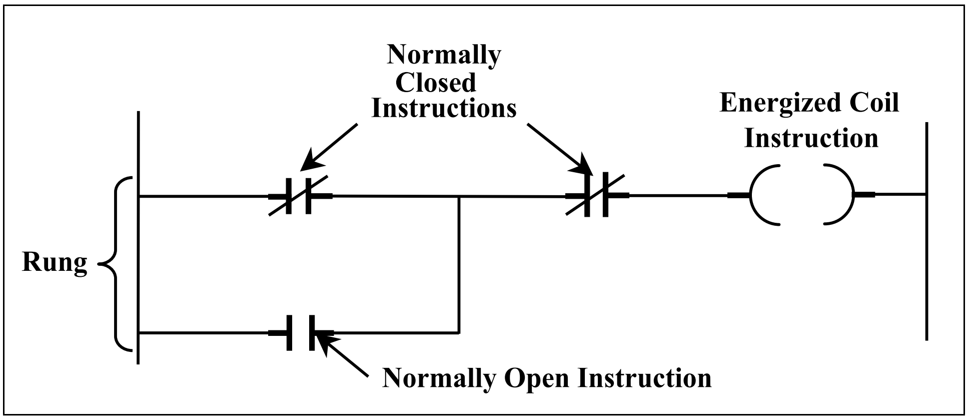

Normally Closed Instructions
Energized Coil Instruction
Rung
Normally Open Instruction

Figure 6-1. Basic Ladder Diagram Instructions

When programmed, each contact and coil is referenced with an address number that identifies what is being evaluated and what is being controlled. Recall that these address numbers reference the data table location of either an internal bit or a connected input or output. A contact, regardless of whether it represents an input/output connection or an internal bit, can be used throughout the program whenever that condition needs to be evaluated.

The format of the rung contacts is dependent on the desired control logic. Contacts may be placed in configurations such as series, parallel, or series and parallel that is required to control a given output. For an output to be activated or energized, at least one left-to-right path of contacts must be closed. A complete closed path is referred to as having logic continuity. When logic continuity exists in at least one path, it is said that the rung condition is true or on. The rung condition is false or off, if no path has continuity.

In the early years, the standard ladder instruction set was limited to performing only relay equivalent logic functions using the basic relay-type contact and coil symbols similar to those illustrated in Figure 6-1. A need for greater flexibility, coupled with developments in technology, led to extended LD instructions that perform data manipulation, arithmetic, and program flow control. We will discuss relay-type instructions first and the more advanced instructions later.

Relay-Type Instructions

The relay-type instructions are the most basic of programmable controller instructions. They provide the same capabilities as hardwired relay logic discussed earlier, but with greater flexibility. These instructions primarily provide the ability to examine the on or off status of a specific bit addressed in memory and to control the state of an internal or external output bit. The following is a description of relay-type instructions that are most commonly available in any controller that has a LD instruction set.

Normally Open Instruction

The normally open (NO) instruction is programmed when the presence of the input signal is needed to turn an output on. When evaluated, the referenced address is examined for an “on” (logic 1) or an “off” (logic 0) condition. The referenced address may represent the status of an external input, external output, or internal output. If, when examined, the referenced address bit is on or logic 1, then the normally open instruction will allow logic flow as shown in Figure 6-2a. If the normally open instruction is off or logic 0, then the normally open instruction will assume its normal state of open, thus breaking logic continuity and preventing logic flow as shown in Figure 6-2b. As an aid in troubleshooting control programs almost all PLC software programming packages will highlight logic bits to indicate the “on” state of the input and output bits and some programming packages will highlight the entire rung as shown in Figure 6-2a, if the logic continuity or flow of the entire rung is activated or on.

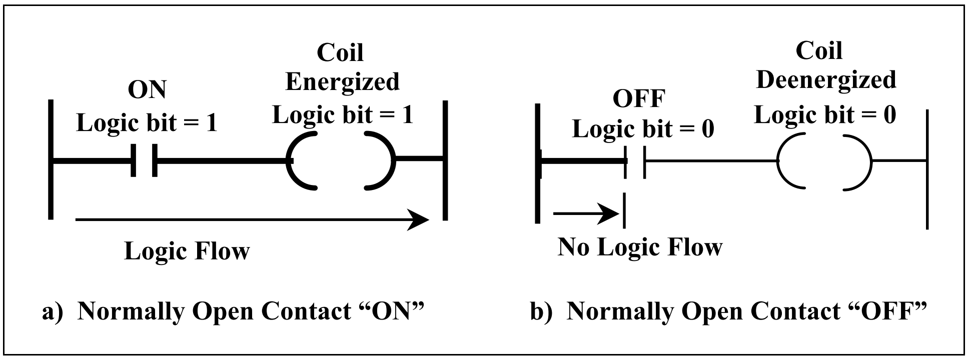


Figure 6-2. Logic Flow for Normally Open Contact Instruction

Normally Closed Instruction

The normally closed (NC) instruction is used when the absence of the referenced signal is needed to turn an output on. When evaluated, the bit address of the NC instruction is examined for an on (1) or an off (0) logic condition. The address referenced by the NC instruction may represent the status of an external input, external output, or an internal output. If, when examined, the referenced bit address is off or logic bit 0, then the normally closed instruction will remain closed, allowing logic continuity as shown in Figure 6-3a. If the referenced input address is on (logic bit 1), then the normally closed contact will open and logic continuity or flow will be broken as shown in Figure 6-3b.

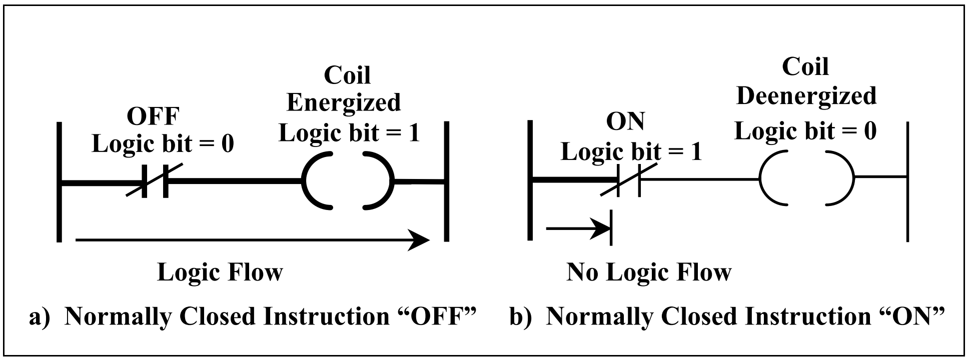


Figure 6-3. Logic Flow for Normally Closed Instruction

Output Coil Instruction

The output coil or energize coil instruction is programmed to control either an output connected to the controller or an internal output bit. The letter "O" or the letter "Q" designates the output coil instruction bit in most programmable controller systems. If any rung path has logic continuity, the referenced output is energized or set true (logic bit = 1). The output bit is turned off, if logic continuity is lost to the output coil. When the output is on, a normally opened instruction of the same address will close, and a normally closed instruction will open. In Figure 6-4, the output instruction O:0/01 is energized or true if either input A or input B is true or if both inputs are true.

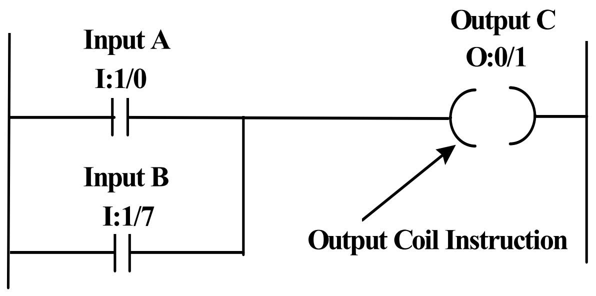


Figure 6-4. "Or" LD Program Using an Output Instruction

Several example applications will help to illustrate LD programming.

EXAMPLE 6-1

Problem: Write a LD program to start and stop a pump. In this application, the normally open contacts of a local panel-mounted start pushbutton (PB) are wired to input bit address I:1/1, and the normally closed contacts of a stop PB are connected to input bit address I:1/0. The pump starter relay is connected to PLC output O:3/1 and the auxiliary (aux.) motor start contacts (NO) are connected to PLC input I:1/2.

Solution: The application can be implemented using the LD program shown in Figure 6-5.

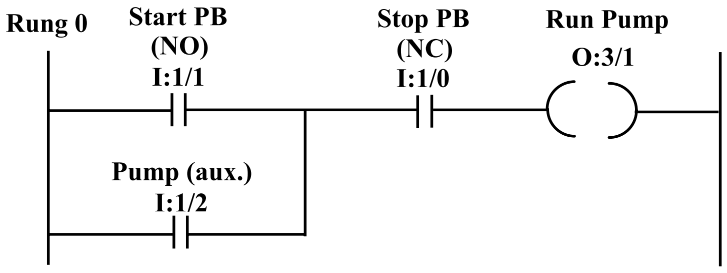


Figure 6-5. Solution for Example 6-1, Pump Start/Stop LD Program

When the NO start PB is depressed, input I:1/1 is true and since the NC stop PB is not depressed, input I:1/0 is also true so that there is logic continuity in rung 0, and the output bit O:3/1 is energized or set to 1. Output O:3/1, energizes the pump start relay causing the pump auxiliary (aux.) contacts to close. This, in turn, sets input bit I:1/2 to seal-in the start PB bit and hold the pump ON until the stop PB is depressed. After the stop PB is depressed, the stop bit, I:1/0 is set to zero and Run Pump output bit is turned off or set to 0, so the pump will be turned off and the auxiliary contacts on the pump starter will open and set the input bit I:1/2 to zero.

It is important to note that NC contacts of stop PBs are always used for safe operation of moving equipment applications. The NC contacts are used in the stop circuit so that if a wire from the stop PB to the PLC is cut or removed, the moving equipment will stop and it cannot be restarted. On the other hand, if the NO contacts of the stop PB were used and the control wires from the PB to the PLC were cut or removed when the moving equipment was energized, the equipment could not be stopped by depressing the stop PB.

EXAMPLE 6-2

Problem: There is only a single field-mounted pushbutton available to turn an electric warning beacon on and off. Write a LD program to control the beacon. Assume the NO contacts of the pushbutton are wired to input point I:1/1, and the beacon is connected to output O:3/0.

Solution: The warning beacon can be controlled using the LD program shown in Figure 6-6.

In the example program shown in Figure 6-6, when the beacon control pushbutton (PB) is depressed for the first time, output bit (O:3/0) is energized and it turns ON the warning beacon. This output control bit (O:3/0) also seals itself in. If the pushbutton is depressed again, the beacon is turned off. The second rung (rung 1) of the program detects the first time the pushbutton is depressed, while the first rung (rung 0) senses the second time the pushbutton is depressed. The last rung (rung 2) is used to control internal bit 3 (B3/03). The normally closed contacts of internal bit 3 are used in rungs 0 and 1 to help perform the "push to start" and "push to stop" function in the LD program.

Latch Coil

The latch coil instruction is programmed, if it is necessary, for an output to remain energized even though the status of the input bits that caused the

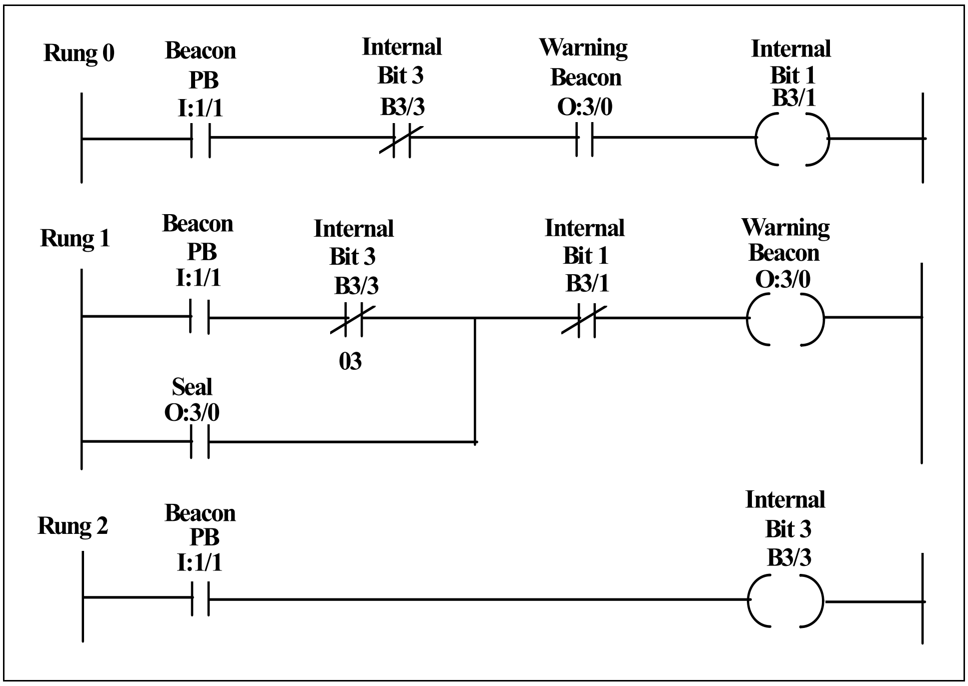


Figure 6-6. Alarm Beacon LD Program

output to energize may change. If any rung path has logic continuity, the output is turned on and retained on even if logic continuity or system power is lost. The latched output will remain latched on until it is unlatched by an output instruction of the same reference address. The unlatch instruction is the only automatic (programmed) means of resetting the latched output. Although most controllers allow latching of internal or external outputs, some are restricted to latching internal outputs only.

Unlatch Coil

The unlatch coil instruction is programmed to reset a latched output of the same reference address. If any rung path has logic continuity, the referenced address is turned off. The unlatch output is the only automatic means of resetting a latched output, other than clearing the program. Figure 6-7 illustrates the use of the latch and unlatch coils to start or stop a process batch.

In this example, the NO contacts of the start PB and the NC contacts of the Stop PB are connected to discrete PLC inputs. The NC contacts of the stop PB are connected to input bit I:001/00 and the NO contacts of the start PB are connected to input I:001/01. If the start PB is depressed, output O:003/01 is latched on. When the start bit (I:001/01) goes false, the output instruction for the pump starter will remain ON until the stop bit (I:001/

00) is depressed to unlatch the output. Note that the output unlatch instruction has the same address as the latched bit.

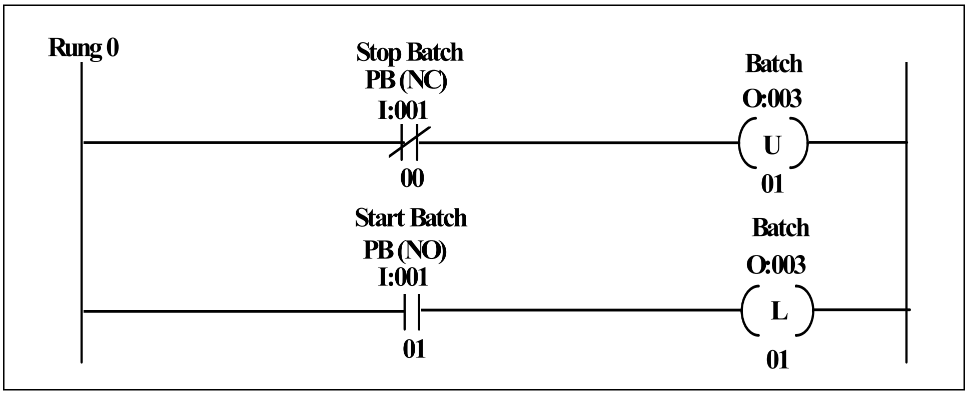


Figure 6-7. LD Program Using Latch and Unlatch Instructions

The LD program in Figure 6-7 is a simpler method to produce a standard start/stop logic function than the standard Start/Stop LD program shown in Figure 6-5.

One Shot

The one shot (ONS) is an input instruction that goes true for one PLC program scan, if there is a false to true transition in the conditions preceding it in the rung. The ONS instruction is generally used to start operations that are triggered by momentary pushbutton action, such as obtaining values from thumbwheel switches or freezing rapidly displayed LED data. In the Allen-Bradley series 5 PLCs, the bit address must be either a binary file (B3) or an integer file (N7) bit address. A typical example is shown in Figure 6-8. In this application, when the data in pushbutton (PB) is depressed, it sets the input bit (I:001/02) to 1 and the ONS bit (B3/04) conditions the rung so that the output (B3/05) turns on for one scan. The output turns off for successive scans until the input goes from false to true again.

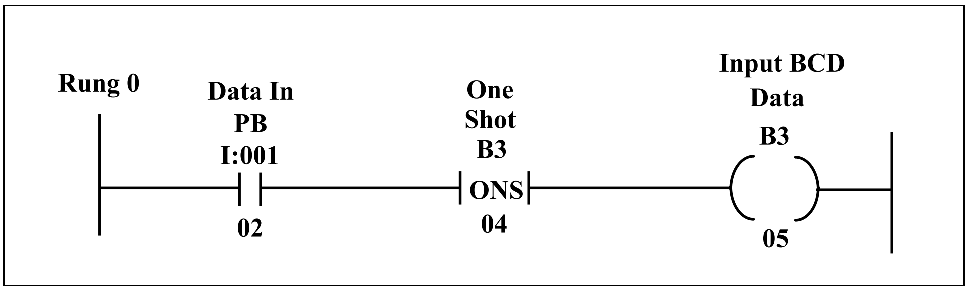


Figure 6-8. One Shot Application

Timer and Counter Instructions

Timers and counters are output instructions that provide the same functions as would hardware timers and counters. They are used to activate or deactivate a device after an expired interval or count. A typical application for a counter is to count the number of parts produced on an assembly line. A typical application for a timer is to delay an operation for a fixed interval. For example, the starting of a pump might be delayed for several seconds until a valve on the discharge line of the pump is completely opened.

The operations of timers and counters are quite similar in that they are both counters. Timers count the number of times that a fixed interval of time elapses. For example, to time an interval of 3 seconds, a timer might count three 1-second intervals. A counter simply counts the occurrence of an event.

Timer Memory Word Structure

Most timer instructions require three memory registers or words: a control word or register, an accumulator (ACC) word to store the elapsed time, and a preset (PR) memory word to store a preset timer value. The preset value will determine the number of time-based intervals that are to be counted. When the accumulated value equals the preset value, a status bit is set on and can be used to turn on an output bit.

A typical example of timer memory word structure is shown in Figure 6-9 for an Allen-Bradley PLC5 programmable controller system.

The left-most three bits (bits 13, 14, and 15) in the timer control word are used as status bits. Bit 15 is the timer enable (EN) bit and it is set when the logic to the timer is logical 1 or true. Bit 14 is the timing bit (TT) and it is set when the timer rung goes true, and it indicates that a timing operation is in progress. Bit 13 is the timer done (DN) bit and it is true when the accumulated value is equal to the preset value.

Timer On Delay

The timer on delay (TON) output instruction is programmed to provide time-delayed action or to measure the duration for which some event is occurring. If any rung path connected to the input side of the timer has logic continuity, as shown in Figure 6-10, the timer begins counting time-based intervals and counts until the accumulated (ACCUM) time equals the preset value as long as the rung conditions remain true. When the accumulated time equals the preset time, a timer done (DN) bit in the

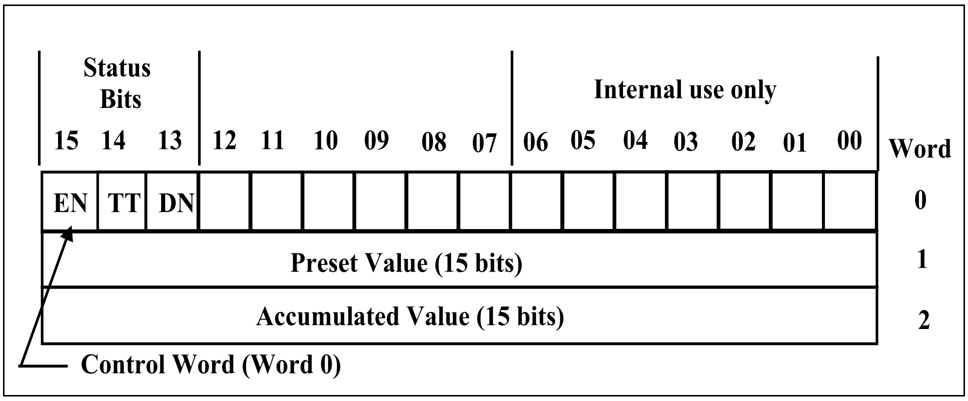


Figure 6-9. A-B PLC5 Timer Word Structure

timer word is set to 1. Whenever the rung logic conditions for the TON instruction go false, the accumulated value is reset to all zeros.

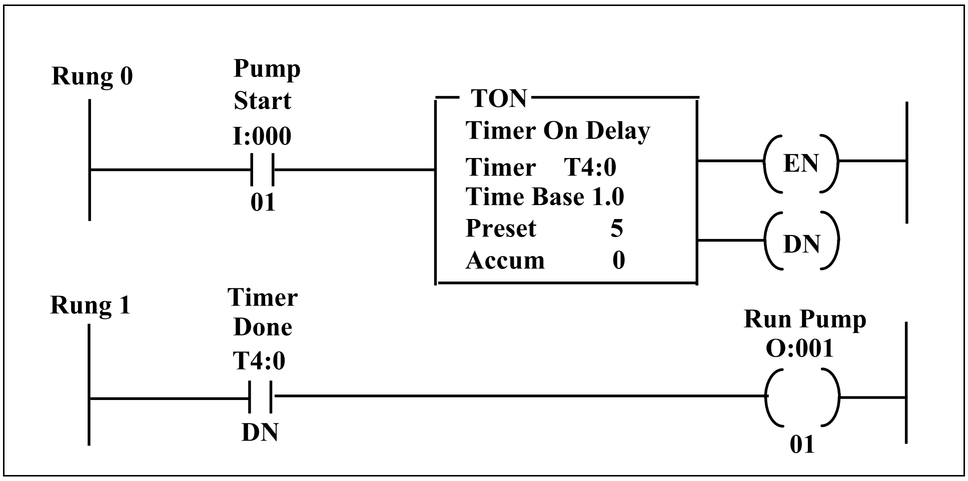


Figure 6-10. Ladder Diagram Using TON Timer Instruction

In the example application shown in Figure 6-10, when the pump start switch is ON, bit I:000/01 is set to 1 and the timer (T4:0) begins to count time-based intervals. As long as the switch remains closed or ON, the timer increments its accumulated value word for each time interval. When the accumulated value equals the preset value of 5 seconds, the timer stops incrementing its accumulated value and sets the timer done (DN) bit to on. This DN bit (T4:0/DN) is then used in rung 1 to energize the run pump output bit (O:001/01).

Timer Off Delay

The timer off delay (TOF) output instruction provides another form of timer action. If logic continuity is lost, the timer begins counting time-based intervals until the accumulated time equals the programmed preset value. When the accumulated time equals the preset time, the timing is complete, and the DN bit (bit 13) is reset to zero. The DN bit can be used throughout the program as a NO or NC instruction. If logic continuity is gained before the timer is timed out, the accumulator word is set to zero and the DN bit is set to a logic 1. An example program for a TOF instruction with a preset value of 5 seconds is shown in Figure 6-11.

In rung 0, when input I:000/01 is true, the DN bit is set to "1", turning on the output bit, O:001/001, if the input switch (I:000/01) is open for 5 seconds or more the timer will count up to 5. When the preset value is equal to the accumulated value of 5, the timer done bit (T4:1/DN) will be set to 0 and the output bit O:001/01 in rung 1 will be turned off.

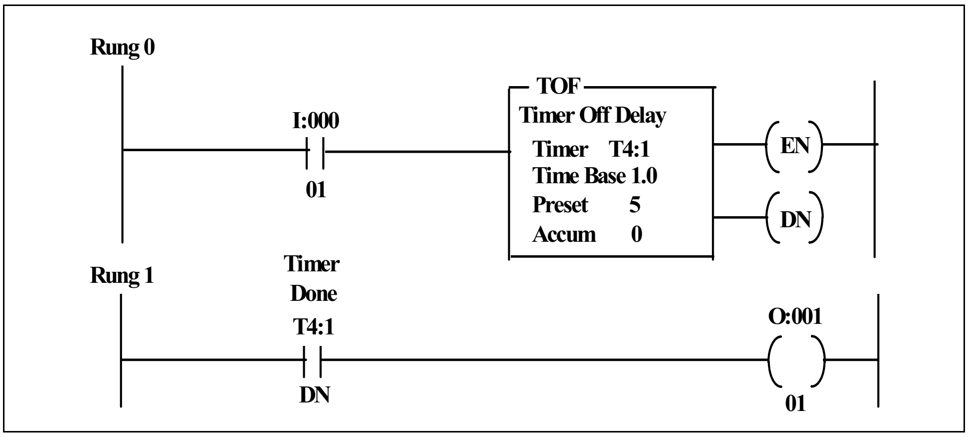


Figure 6-11. Ladder Diagram Using a TOF Timer Instruction

Retentive Timer On

The retentive timer on (RTO) output instruction is used if it is necessary for the timer-accumulated value to be retained, even if logic continuity or power is lost. If the timer-rung path has logic continuity, the timer begins counting time-based intervals until the accumulated time equals the preset value. The accumulator register retains the accumulated value even if logic continuity is lost before the timer is timed out or if power is lost, as shown in Figure 6-12. When the accumulated time equals the preset time, the done bit is set. The timer done bit can be used throughout the rest of the program as NO or NC instructions. The retentive timer accumulator value is reset to zero by a reset (RES) instruction. The RES instruction

(T4:2/RES) in rung 1 will set to logical 1, if the input bit I:000/03 is set to 1. This will reset the accumulator in timer T4:2 to zero and reset the DN bit.

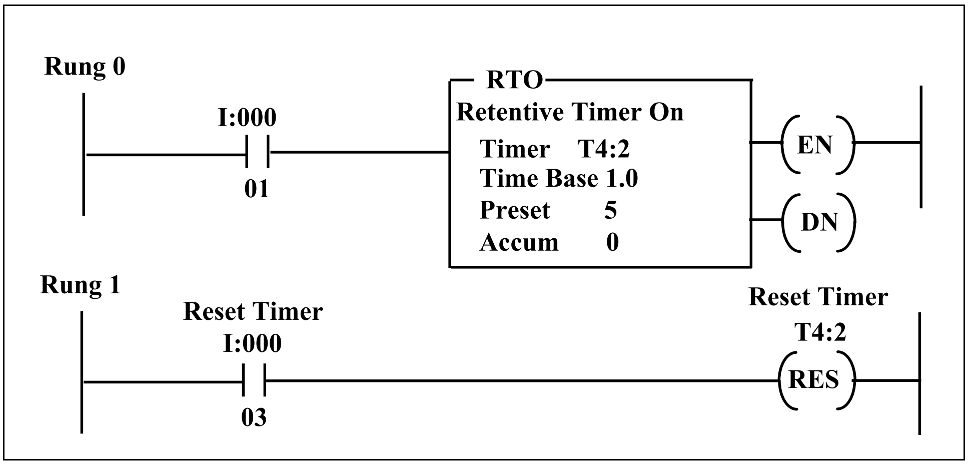


Figure 6-12. Ladder Diagram Using a Retentive Timer on Instruction

A typical application for timers is to produce alternating pulses for a flashing alarm light as illustrated in the following example problem.

EXAMPLE 6-3

Problem: Design a timing circuit that can be used to generate an alternating signal to produce a flashing alarm light connected to output at bit location O:3/1. Select a time period of 0.5 seconds off and 0.5 seconds on.

Solution: The LD program consists of three logic rungs with 0.5 second timers in rungs 1 and 2 and a timer done bit driving the alarm light in rung 3 as shown in Figure 6-13 below.

Up Counter

The accumulated value in the up counter (CTU) output instruction will increment by one, each time there is a 0 to 1 transition of the input logic. A typical control application of a counter is to turn a device ON or OFF, after reaching a certain count. Since CTUs increment their accumulated value only when the CTU logic input makes a 0 to 1 transition, the rung condition must go from true to false and then back to true before the next count is registered.

When the accumulated (AC) value reaches the preset (PR) value, the count done (DN) bit is set to 1. Unlike a timer instruction, the counter instruction continues to increment its accumulated value after the preset value has

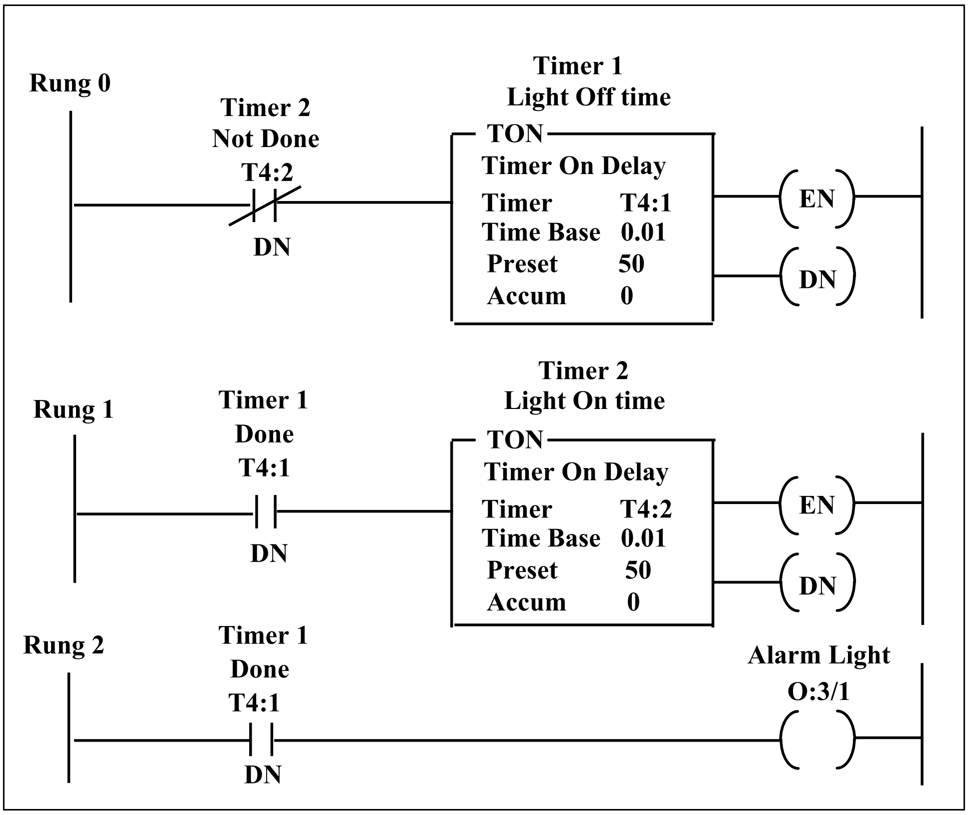


Figure 6-13. LD Program Solution to Problem in Example 6-3

been reached. If the accumulated value goes above the maximum range of the counter, an overflow (OV) bit will be set. This overflow bit can be used to cascade counters for counter applications greater than the maximum value of the counter.

Down Counter

The down counter (CTD) output instruction will count down by one each time a false to true (0 to 1) transition occurs in the input logic to the counter. In some applications, a down counter is used in conjunction with an up counter to form an up/down counter.

Figure 6-14 shows a typical example of an up/down counter using A-B PLC5 up and down counter instructions. Note that the same word address, C5:0, is used for both counters.

EXAMPLE 6-4

Problem: Write a LD program using counter instructions to count the number of parts produced on an assembly line. Assume input I:000/12 of the PLC is activated by each part leaving the final assembly line; input I:000/13 is activated by a part being rejected in the final test, and input I:000/00 is energized at the end of a production run.

Solution: The ladder program to calculate the number of parts produced is shown in Figure 6-15. The number of parts produced is the value found in the accumulator of the counter C5:0.

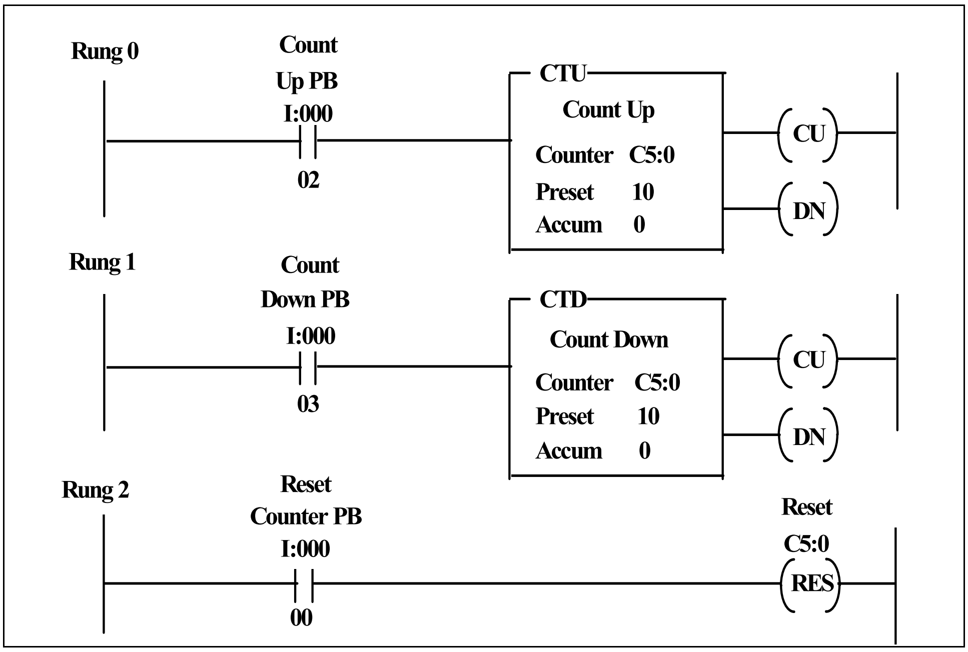


Figure 6-14. Ladder Diagram for Up/Down Counter

Data Transfer Operations

Data transfer instructions involve the transfer of the contents from one register to another. Data transfer instructions can address any location in the memory data table, with the exception of areas restricted to user application. Pre-stored values can be automatically retrieved and placed in any new location. That location may be the preset register for a timer or counter or even an output register that controls a seven-segment display.

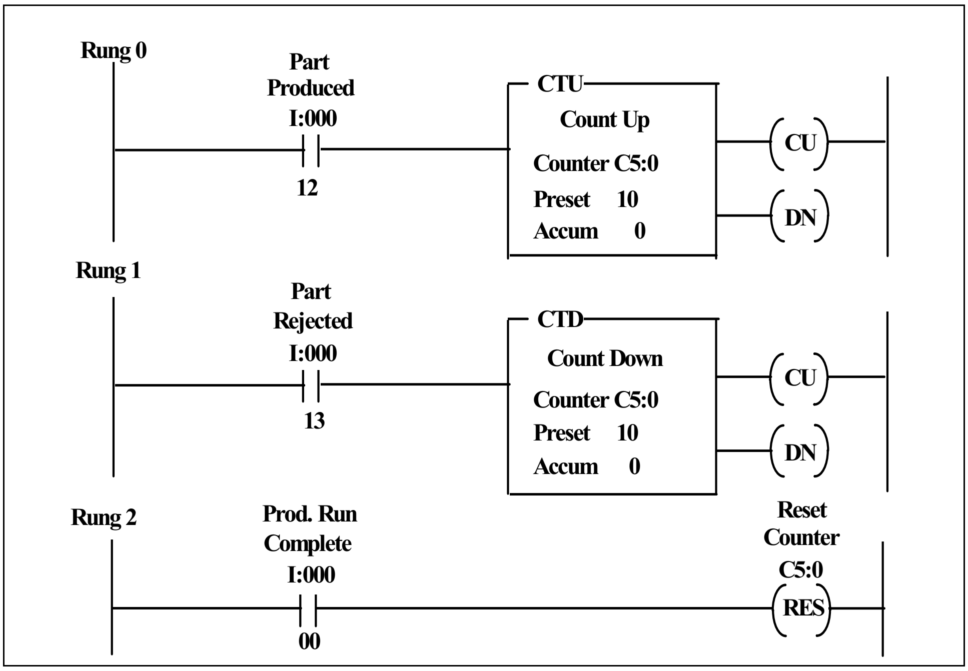


Figure 6-15. Ladder Diagram for Production Part Count (Solution for Example 6-4)

The Allen-Bradley PLC5 programming system uses three data bit and word transfer instructions: bit distribute (BTD), move (MOV), and masked move (MVM). These data transfer instructions are typical for most PLC manufacturers.

Bit Distribute

The bit distribute (BTD) instruction is an output instruction that moves up to 16 bits of data within or between words and the source remains unchanged. Figure 6-16 shows a BTD example of moving bits within a word. The instruction writes over the destination with the specified bits. If the length of the bit field extends beyond the destination word, the processor does not save the overflow bits. These overflow bits are lost because they do not wrap into the next word.

On each processor program scan, when the rung that contains the BTD instruction is true, the processor moves the bit field from the source word to the destination word. To move the data within a word, the programmer selects the same word address for both the source and the destination as shown in Figure 6-16. In this example, four bits are moved from the left-hand side (bits 00 through 03) of word N70:00 to the middle of the word (bits 08 through 11).

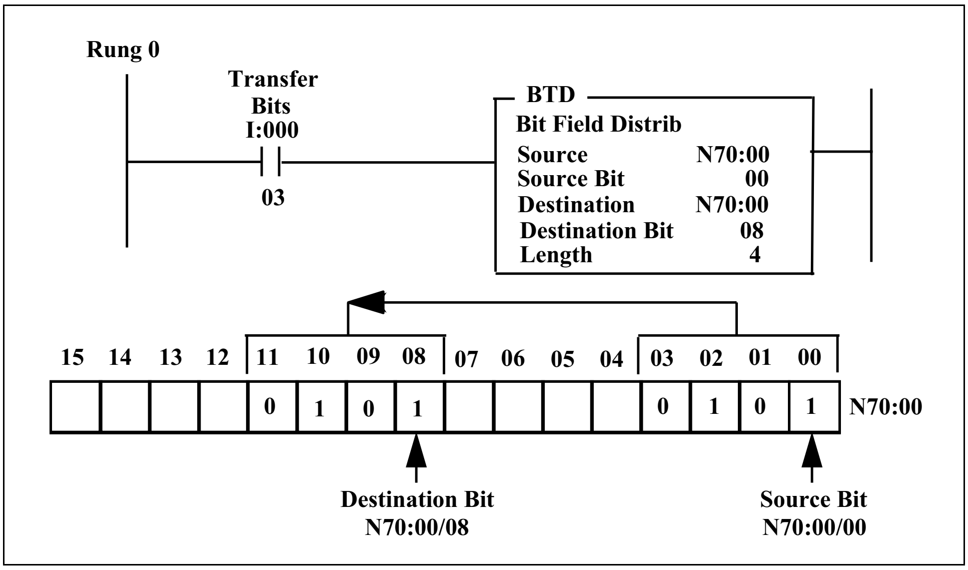


Figure 6-16. Example of BTD Instruction Moving Bits Within a Word

Move and Masked Move

The move (MOV) instruction is an output instruction that copies the data in a source address to a destination address. As long as the rung remains true, the instruction moves the contents each scan of the PLC processor to the destination address. The source can be a program constant or data address from which the instruction reads an image of the value. The destination is the data address to which the instruction writes the result of the operation. The instruction writes over any data stored at the destination. For example, in rung 0 of Figure 6-17 when input I:000/02 is true, the data stored at address N7:10 is copied and written into location N7:12.

The masked move (MVM) instruction is an output instruction that copies the source to a destination and allows portions of the data to be masked. As long as the rung remains true, the instruction moves data each scan.

The MVM instruction can be used to copy I/O image table, binary, or integer values. For example, the MVM instruction can be used to extract bit data such as status or control bits from an address that contains bit and word data. The source is a program constant or data address from which the instruction reads an image of the value and the source remains unchanged.

The mask can be an address or hexadecimal value that specifies which bits to pass or block. The programmer must set the mask bits to 1 to allow the data to pass to the destination. The moved data overwrites the destination data. The destination is the data address to which the instruction writes the result of the operation. The instruction writes over any data stored at the destination.

Figure 6-17 shows an example of both MOV and MVM instructions. In rung 0, when the input bit I:000/02 is true, the contents of memory location N7:10 are moved to integer word location N7:12. In rung 1, when the input bit I:000/02 is true the contents of higher 8 bits of memory word N7:10 are moved to integer word location N7:12, since there are all ones (two hex "F") in the higher 8 bits of the mask (FF00). The lower 8 bits are blocked because there are all zeros in the lower 8 bits of the mask.

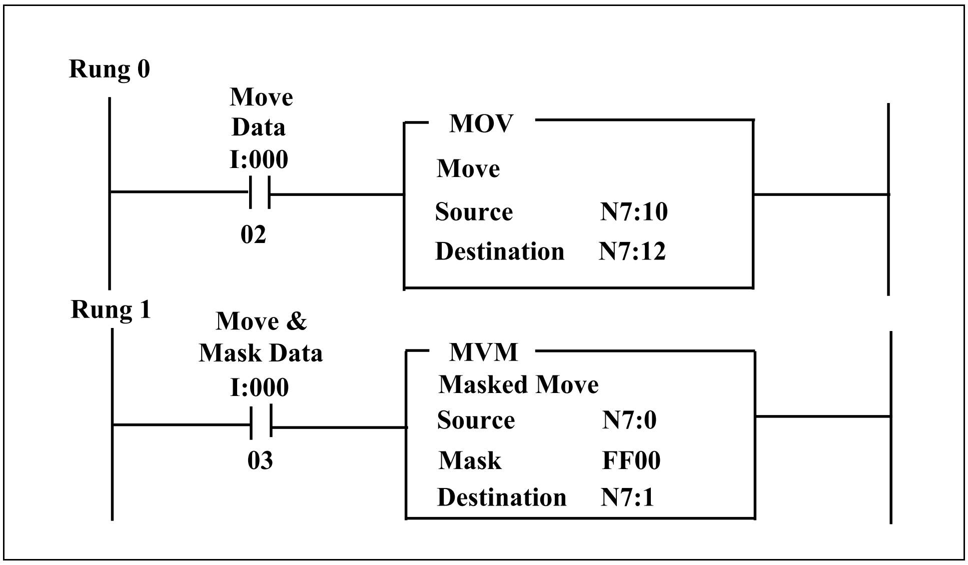


Figure 6-17. Typical Example of MOV and MVM Instructions

Arithmetic Operations

The arithmetic operations include the four basic operations of addition, subtraction, multiplication, and division. These instructions use the contents of two word locations to perform the desired function.

The arithmetic instructions are output instructions that may or may not have input logic instructions. They use either one or two data words to store the result. The add and subtract instructions use one word. Multiply and divide need two words for the computed result.

Addition

The addition (ADD) instruction performs the addition of two values stored in two different memory locations, source A and source B, and the result is stored in the destination register. Source A and source B can be either values or addresses that contain values. In the example shown in Figure 6-18, if the input I:000/01 is true, the instruction adds the value stored in N7:0 to the value stored in N7:1 and stores the result in address N7:2.

Subtraction

The subtraction (SUB) instruction performs the subtraction operation of two numbers located in source A and B. In addition, if a condition to enable the subtraction is set to 1, the result is placed in a destination. In Figure 6-18, looking at rung 1, if the input bit at address I:000/02 is set to 1, the number in address N7:4 is subtracted from the number in address N7:3 and the result is placed in register N7:5.

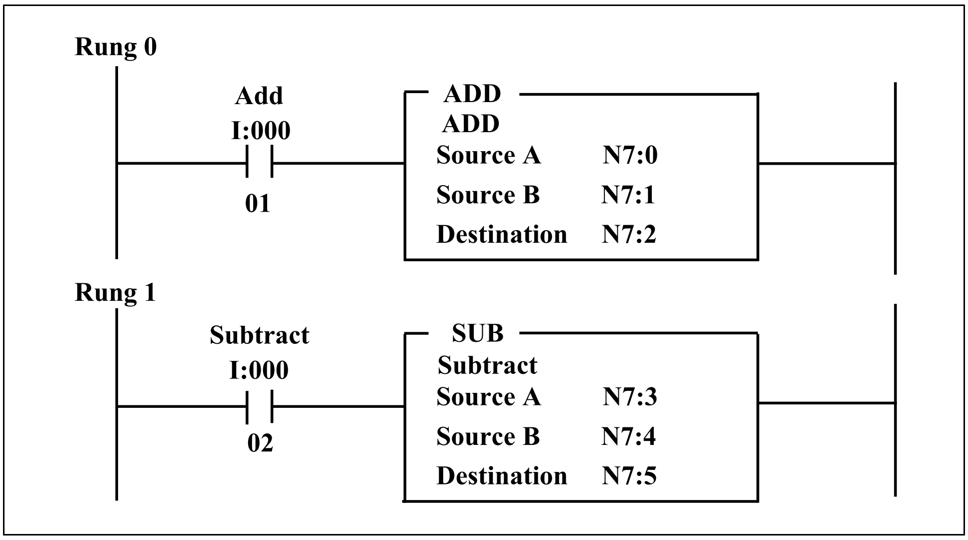


Figure 6-18. Typical Example of ADD and SUB Instructions

Multiplication

The multiplication (MUL) instruction is used to multiply one value (Source A) by another value (Source B) and to place the result in the destination. Source A and Source B can be values or addresses. In the typical MUL instruction example of Figure 6-19, if input bit I:000/03 in rung 0 is true, the processor will multiply the value in N7:3 by the value in N7:4 and store the result in N7:20.

Divide

The divide (DIV) instruction is used to divide one value (Source A) by another value (Source B) and place the result in the destination. Source A and Source B can be values or addresses. In the typical DIV instruction example of Figure 6-19, if input bit I:000/02 in rung 1 is true, the processor will divide the value in N7:0 by the value in N7:1 and store the result in N7:21.

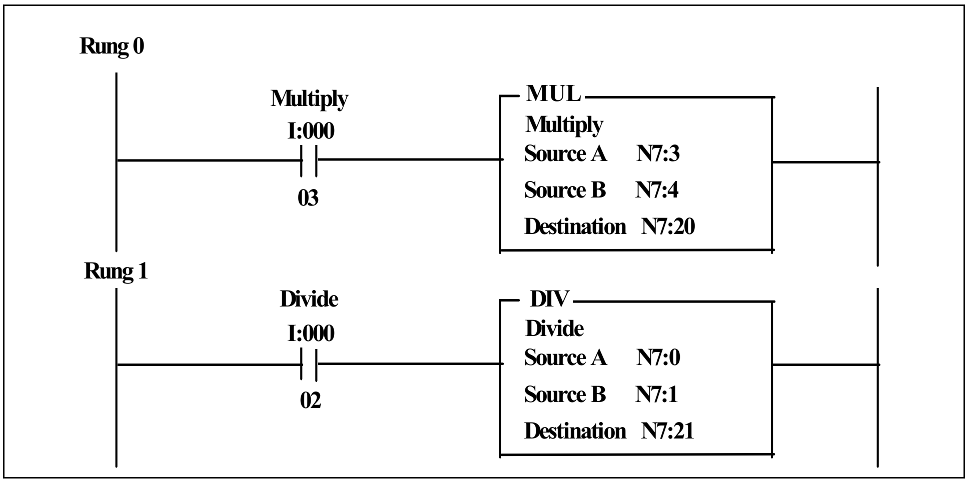


Figure 6-19. Typical Examples of MUL and DIV Instructions

Data Comparison Operations

In general, the manipulation of data using ladder diagram instructions involves simple register (word) operations to compare the contents of two registers. In the ladder diagram language, there are three basic data comparison instructions: equal to, greater than, and less than. Based on the result of a greater than, less than, or equal to comparison, an output can be turned ON or OFF, or some other operation can be performed.

Equal To

The equal to (EQU) instruction is used to test whether two values are equal. Source A and Source B can be either values or addresses that contain values. For example in rung 0 of Figure 6-20, if the equality is true the output coil is energized.

Less Than

Similar to the EQU instruction, the less than (LES) instruction tests the contents of the value of one location (Source A) to see if it is less than the value stored in a second location (Source B). If the test condition is true, the output coil in rung 1 in Figure 6-20 is energized.

Greater Than

The greater than (GRT) instruction operates like the LES operation, with the exception that the test is performed for a GRT condition. If the test condition is true, the output coil in rung 2 of Figure 6-20 is energized.

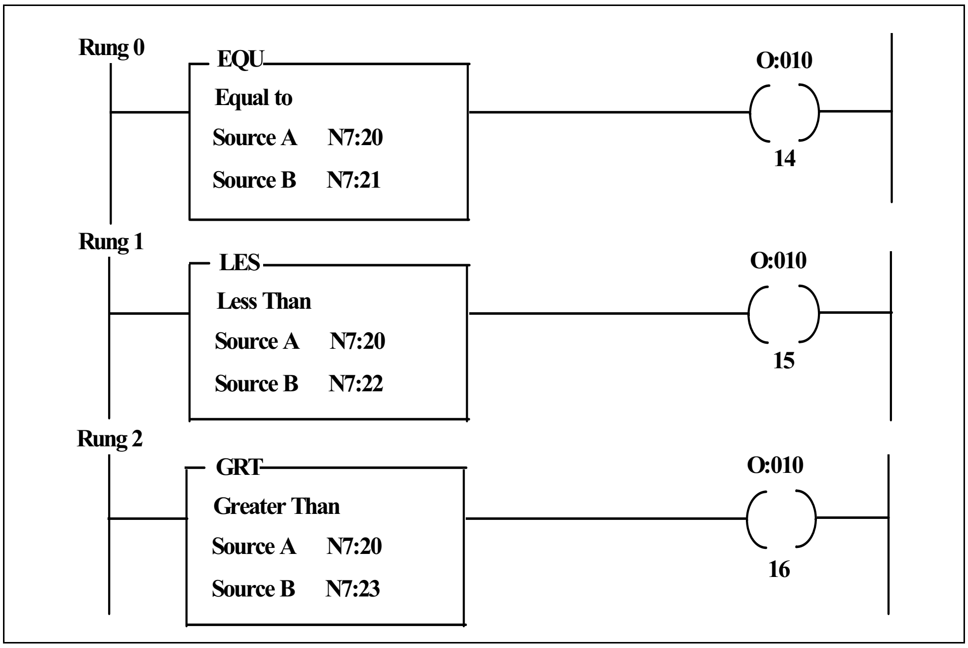


Figure 6-20. Examples of Data Comparison Instructions

Program Control Instructions

The program control functions are used to perform a series of conditional and unconditional jump and return instructions. These instructions allow the program to execute only certain sections of the control logic if a fixed set of logic conditions are met. The following instructions are a representative selection of some of the program control instructions available in most controllers.

Master Control Relay

The master control relay (MCR) instruction is used in pairs to activate or deactivate the execution of a group or zone of ladder rungs. A conditional MCR instruction is used in conjunction with another unconditional MCR coil to place a fence around the group of rungs. For example, in Figure 6-21, if the input I:000/03 is true, the conditional MCR coil in rung 0 will be energized and the logic inside the zone will be executed according to the logic in each rung inside the MCR zone. If the conditional MCR instruction is turned OFF, all non-retentive outputs inside the zone will be deenergized.

The rungs within an MCR zone are still scanned, but the PLC scan time is reduced due to the false state of the non-retentive outputs. Non-retentive outputs are reset when their rung goes false.

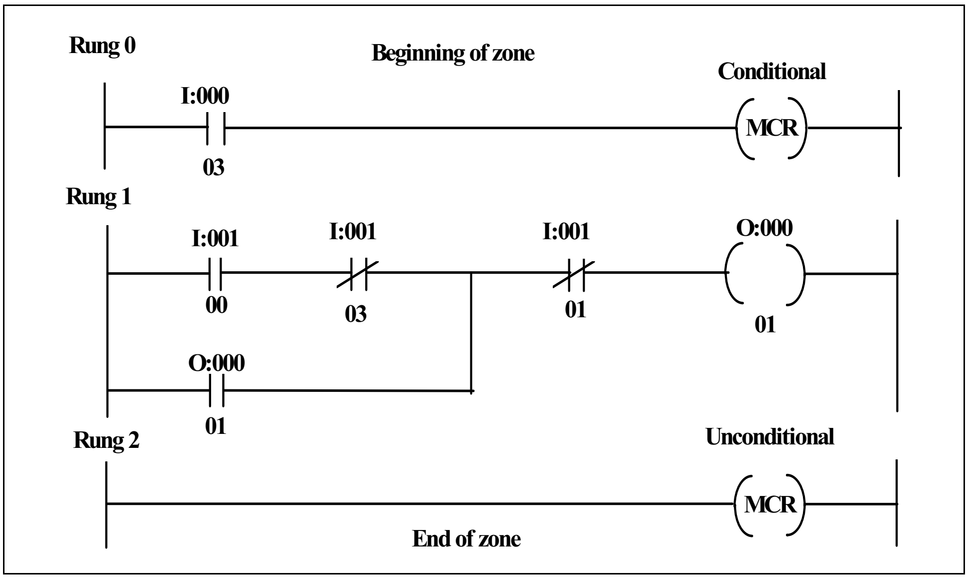


Figure 6-21. A typical LD Program Using MCR Instructions

Jump and Label

The jump (JMP) and label (LBL) instructions are used in pairs to skip portions of the LD program. The JMP instruction allows the normal sequential execution to be altered so that the CPU will jump to a new position in the LD program. If the jump-rung logic is true, the jump coil instructs the CPU to jump to and execute the rung labeled with the same reference address as the jump coil. This allows the program to execute rungs out of the normal sequential flow of a standard ladder program.

The label instruction is used to identify that ladder rung number that is the destination of the jump instruction. The label reference must match that of the JMP instruction with which it is used. The LBL instruction does not contribute to logic continuity, and it is always logically true. It is placed as the first logic condition in the rung. A LBL instruction referenced by a unique address can be defined only once in a program.

Jump to Subroutine, Subroutine, and Return

Subroutines are used in programming to produce a more structured program and to reduce the amount of memory used for a program. Subroutines are used to store recurring logic functions that can be accessed from different parts of the main LD program. This saves memory space because the function has to be programmed only once, but it is used many times in the control application.

The Allen-Bradley PLC5 has three subroutine instructions: the jump to subroutine (JSR), the subroutine (SBR), and the return (RET). These instructions direct the processor to go to a separate subroutine file within the LD program, scan that subroutine file once, and return to the point of departure.

The JSR instruction directs the processor to the specified subroutine file and, if required, defines the data passed to and received from the subroutine. The optional SBR instruction is the instruction that stores incoming data. The SBR instruction is only used if the LD requires that data is passed to and from the subroutine. The RET instruction ends the subroutine and, if required, stores data to be returned to the JSR instruction in the main program. If the SBR instruction is used, it must be the first instruction on the first rung in the program file that contains the subroutine.

Programming Software Packages

The most common method used to program PLCs is to use a Windows® software package that lets a user or software engineer monitor and modify the activity of a PLC using a PC. The main advantage of using a Windows-based software package is that Windows is a widely used software platform with a large variety of highly developed standardized software tools.

A typical example of Windows-based PLC programming software is the Rockwell Automation RSLogix5™ software package that is used to program Allen-Bradley PLC5 processors. The system-programming screen for RSLogix5 is shown in Figure 6-22; the window on the left is the project window and the window on the right is the ladder programming window. The two instruction bars above the ladder window are used to enter or modify any ladder program. The programmer uses the lower instruction bar to select the type of ladder instruction, such as bit, timer/counter, and input/output to be used. The user or programmer then selects the particular instruction to be entered into the control program.

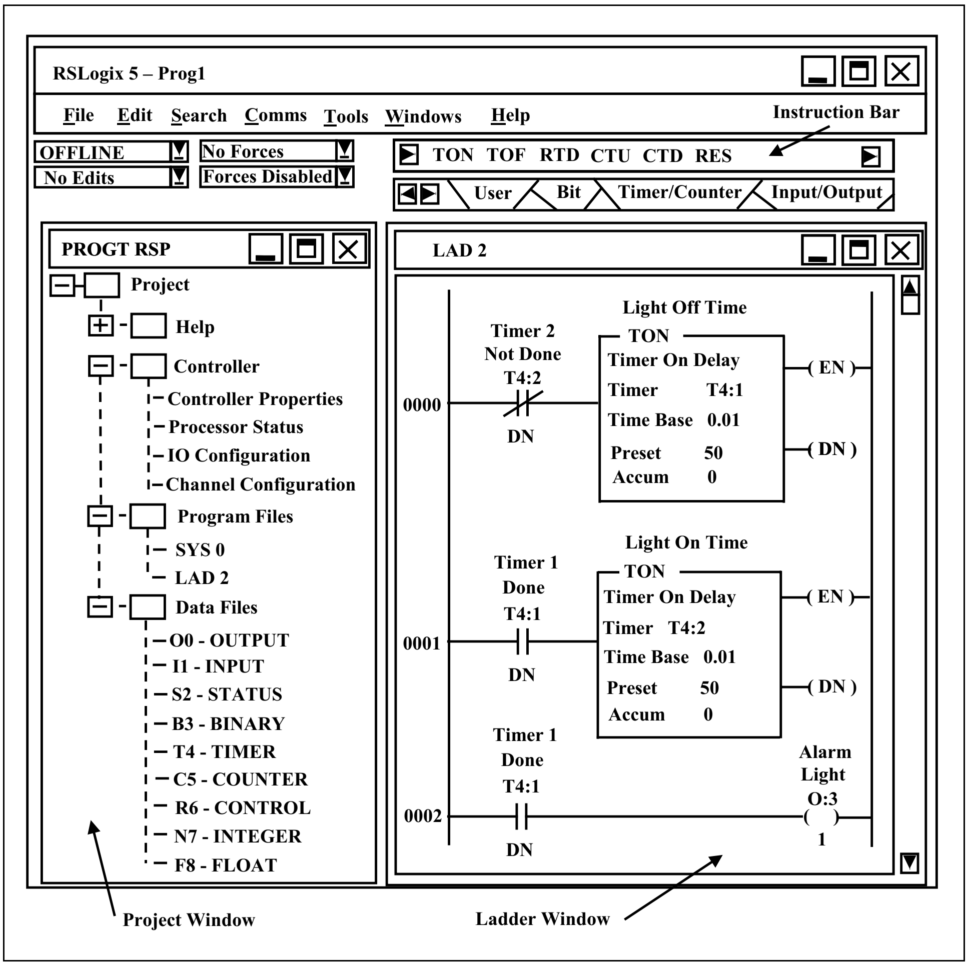


Figure 6-22. RSLogix5 Software Overview

For example, if the user clicks on the Timer/Counter tab in the lower bar, the available timer and counter instructions such as TON, TOF, or RTD will appear in the upper bar. The programmer then clicks on the particular instruction needed in the control program, such as the TON (Timer On Delay) instruction shown in Figure 6-22 and drops it into the ladder program window below.

The first step in using RSLogix5 to program an Allen-Bradley PLC5 programmable controller is to select a project filename. This will display the project window shown on the left in Figure 6-22. This window lists all the files associated with the project. The project window operates just like the Windows Explorer program and these project file types fall into two general categories: processor or database files. The processor files consist of software programs and process and machine information used by the PLC5 processor to control a process or machine. These files are transferred from a PC to the PLC processor and can be changed or monitored from the PC while the PLC processor is using them. The database files contain process or machine descriptions and other data entered by the user to assist in troubleshooting or reviewing the control program but this data is not needed by the PLC processor to control the process or machine. Rather, this other data remains in the PC memory so that it does not use valuable and limited processor internal memory.

PLC Control Program Documentation

An important part of PLC system design and programming is the proper documentation of the control program. PLC manufacturers provide a means to print out a hard copy of the control program stored in the PLC's memory. Whether stored in ladder diagram form or some other language, the hardcopy will be an exact replica of the control program stored in memory. This hardcopy printout will show each programmed instruction with the associated address of each input and output. However, the information on the function or purpose of each field device or internal control bit or instruction is not readily apparent; additional documentation is generally required. This additional documentation is critical because in many cases the people responsible for troubleshooting a project are different than the people who programmed it.

Most PLC manufacturers provide software documentation programs that allow the programming device, generally a PC, to enter labels, mnemonic nomenclature, or descriptive text for each program element or instruction. The example ladder diagram programs in this chapter illustrate documented elements or instructions such as input, output, timer, counter, and so on in the ladder rungs. Most PLC programming software packages also allow for comments on each rung of logic as shown in Figure 6-23.

There are five types of descriptive text used by RSLogix5 software package to identify the function or purpose of a part of a ladder diagram program. The five types are page titles, rung comments, address descriptions, instruction comments, and symbols. A page title is the text that describes the function or purpose of one or more rungs.

A rung comment is text used to detail the purpose of the individual rung to which it is connected. For example the comment for rung 1 in Figure 6-23, states that the purpose of the rung is to energize a warning beacon if the logic is true.

Address descriptions are used to describe the purpose of an address regardless of the type of instruction that is associated with the address. Once assigned, an address description appears with all instances of that address.

An instruction comment is the text used to describe the function of a given address and the specific instruction that it is tied to. Because an instruction comment is specifically associated with the address and instruction combination, it normally overrides any address descriptions.

A symbol is a specifically formulated single word that is tied directly to a bit or software word in the data table to identify the function of a bit or word. Unlike the other types of documentation, symbols have addressing properties. For example, a user or programmer can use symbols in place of an address to enter or edit ladder logic, to monitor data, or to search for an address or word. For example, the output bit O:3/0 for the warning alarm beacon listed in the ladder program of Figure 6-23 could be assigned the symbol address of Alarm_Beacon.

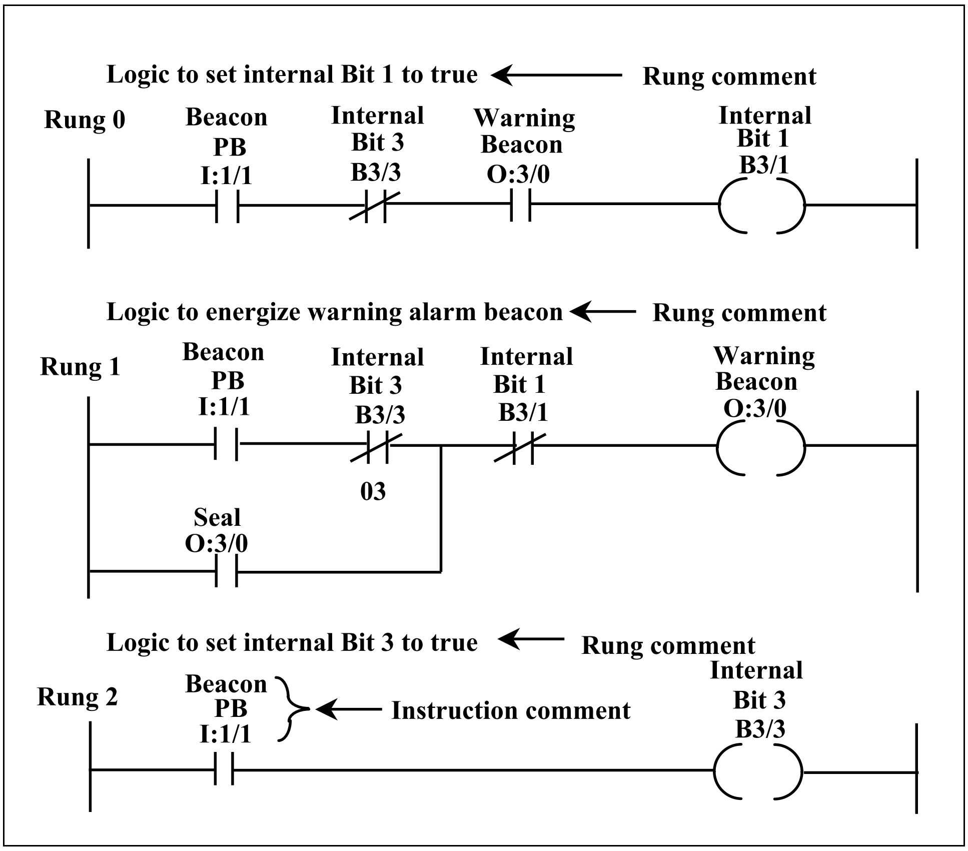


Figure 6-23. Alarm Beacon Application Program with Rung Comments

Some caution should be used in testing, modifying, or troubleshooting a PLC control program. The PLC processor will always have the latest revision of the control program stored in memory, so before testing a program on-line, the user should save the program and print out the latest revision. During start-up and testing, frequent changes are made to the control program that should be immediately documented with both rung and instruction comments. It is also good practice to obtain the latest hardcopy of the PLC program before performing maintenance on a system.

EXERCISES

6.1 What are the three most common types of programming languages encountered in programmable controller systems?

6.2 What is the purpose of a typical “unlatch coil” instruction in a LD program?

6.3 Write a LD program to start a pump that fills a process tank with fluid until a high level is reached in the tank. Assume that the fill tank pushbutton (PB1) is wired to an Allen-Bradley PLC5 discrete input module at bit I:010/00 and a NC tank high-level switch is connected to input bit I:010/01. Also assume that the pump start relay is connected to output O:000/01 on the PLC.

6.4 Write a LD program to control an electric motor assuming that a NO start pushbutton is wired to an input module at I:000/00 and a NC stop pushbutton is connected to the same input module at address I:000/01. Also, assume that PLC5 output O:001/03 is connected to the motor starter and that the NO auxiliary contacts on motor start contactor are connected to PLC input I:000/02.

6.5 Write a LD program to control a process pump under the following conditions: the pump is turned on 5 seconds after both the inlet and outlet valves to the pump have been opened and the pump is turned off, if either the inlet or the outlet valve are closed. Assume the following: a pump starter is connected to output bit O:001/00, a valve open position switch on the inlet valve is wired to input I:000/02, and a valve open position switch on the outlet valve is wired to input I:000/03.

6.6 Write a LD program to open the control valve (LV-1) on the outlet line of the process tank that is shown in Figure 6-24, if the NO level switch (LSH-1) closes when the automatic position on the HOA (HS-1) is selected or if the operator places the HOA in the hand position. Assume that the high-level switch is connected to input I:002/01 and the output valve is connected to output O:003/01 on an Allen-Bradley PLC5. Also, assume that the “Auto” position of the HOA switch is connected to input I:002/02 and the “Hand” position of the switch is wired to input I:002/03 on the PLC5.

6.7 Write a LD program using timer instructions to flash two process alarm lights on a control panel. The two alarm lights are driven by

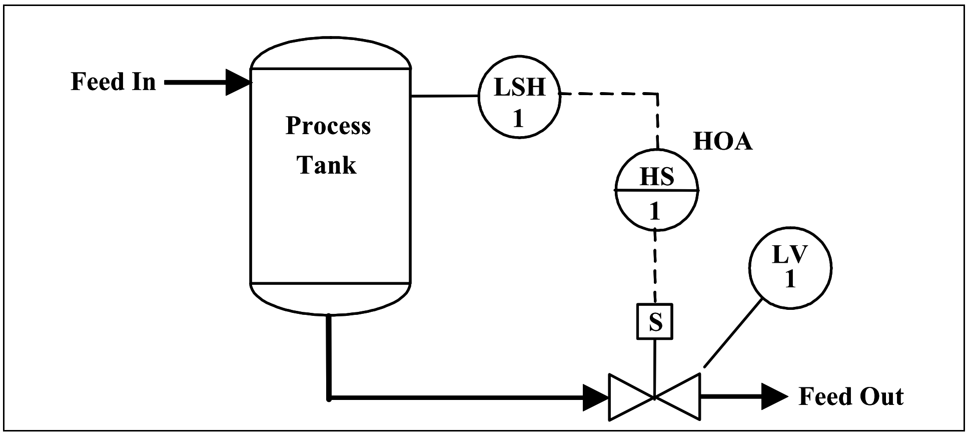


Figure 6-24. Process Tank Application for Exercise 6.6

PLC output signals O:010/00 (alarm 1) and O:010/01 (alarm 2). Assume that alarm 1 is activated by input I:000/01 and alarm 2 is activated by input I:000/02.

6.8 Write a LD program to control the temperature of the fluid in a process tank close to 400°C. Assume that the heater contactor is connected to PLC5 output O:001/01, a temperature low switch set at 395°C is connected to input I:000/00 and temperature high switch set at 405°C is connected to input I:000/01. The temperature low switch is closes if the fluid temperature is below 395°C and opens at a temperature of 395°C or higher. The temperature high switch is closes if the fluid temperature is below 405°C and opens at 405°C or higher temperatures.

6.9 Write a LD program for an Allen-Bradley PLC5 to turn off a conveyor belt on a production line after 50 parts have been produced. Assume the following: 1) when output bit O:001/12 set to 1 the conveyor belt is turned on, 2) that input I:002/01 changes from 0 to 1 and then back to 0 each time a production part is rejected, 3) input I:002/02 changes from 0 to 1 and then back to 0 each time a new part is produced, 4) a NO pushbutton connected to input I:002/03 is used to set the production count to 50, and 5) a NO pushbutton connected to input I:002/04 is used to reset the counter to zero and to stop the conveyor belt.

6.10 Write an A-B PLC5 program to add the integer number in word N7:2 to the integer number in word N7:4, then divide the result stored in N7:20 by 5 and store the result in word N7:6.

6.11 Write a LD program to transfer the integer number in word N7:0 to an output display at word N7:10, if the number in word N7:0 is greater than the number in word N7:1 and less than the number in location N7:2.

6.12 Explain the purpose and benefits of using subroutines in PLC programming applications.

6.13 What are the two types of project files used in the Rockwell Automation RSLogix5 software package?

6.14 List the five types of descriptive text used to document a typical PLC control program.

BIBLIOGRAPHY

1. *PLC-5 Programmable Controller: Instruction Set Reference,* Rockwell Software, Inc., 1996.
2. Reis, R. A., and Webb, J. W., *Programmable Controllers: Principles and Applications*, Prentice-Hall, Inc., 3rd edition, 1995.

7 Advanced LD Programming

Introduction

The advanced ladder diagram (LD) instructions are required to perform more powerful functions beyond simple ON/OFF control, timing, counting, and data manipulation. These advanced instructions are used for analog control, data file operations, sequencer operations, data reporting, complex logic functions, and other functions that are not possible with the basic LD instructions.

Advanced LD Instructions

Advanced LD instructions let the user program more complex PLC control functions. The most common advanced instructions—file, shift register, sequence, and block transfer—for the Allen-Bradley PLC5 family of PLCs will be discussed.

File Instructions

A file is a group of consecutive data table words used to store PLC information. A file instruction is used to perform arithmetic, logical, search, copy, and compare operations. We will discuss the Allen-Bradley PLC5 file instructions—file arithmetic and logical (FAL), file search and compare (FSC), file copy (COP), and file fill (FLL).

The structure of a typical file instruction is shown in Figure 7-1. This figure illustrates a FAL file instruction with its parameters of control, length, position, mode, destination, and expression.

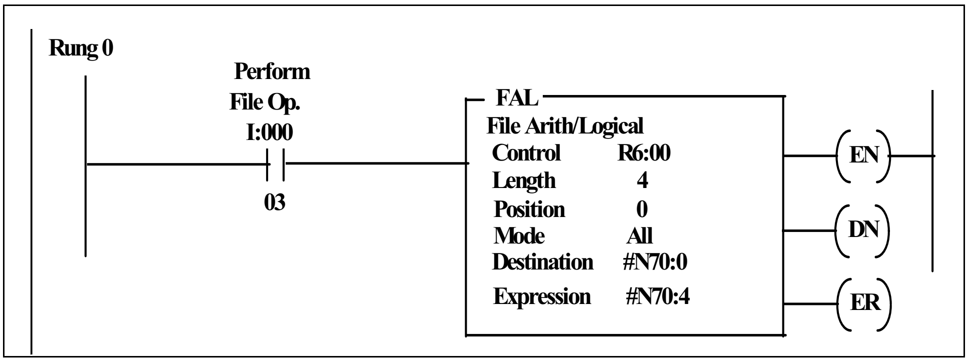


Figure 7-1. Typical A-B PLC5 File Operation Instruction

The *control* is the address of the control structure in a control type (R) file. The processor uses this information to run the instruction. The *length* is the number of words (0 to 999) in the data block on which the file instruction operates. The *position* is the current element within the data block that the processor is accessing. The *mode* is the number of file elements operated on each time the rung is scanned in the program. There are three modes: All, Numerical, and Incremental. In the All mode, the entire file is operated on before continuing on to the next rung of the program. The Numerical mode distributes the file operation over a number of program scans. The Incremental mode manipulates one word of the file each time the rung goes from false to true. The *destination* is the address where the processor stores the result of the operation. The instruction converts to the data type specified by the destination address. The *expression* contains addresses, program constants, and operators that specify the source of data and the operations to be performed.

The output coils to the right of the file instruction are the enable (EN), done (DN), error (ER) bits. These bits have the same word address as the instruction control. The processor automatically sets the address of these status bits when the programmer enters the control address. The EN bit is set by a false-to-true rung transition and indicates the instruction is enabled. In the Incremental mode, the EN bit follows the rung condition. In the Numerical and All modes, the EN bit remains set until the instruction completes its operation, regardless of the rung condition. The enable bit is reset when the rung goes false and the instruction completes its operation.

The done (DN) bit is set after the instruction has operated on the last set of words. In the Numerical mode if the instruction is false at completion, it resets the done bit one program scan after the operation is complete. If the instruction is true at completion, the done bit is reset when the instruction goes false.

The ER bit is set when the operation generates an overflow. The instruction stops until the ladder program resets the error bit. When the processor detects an error, the position value stores the number of the word that faulted.

File Arithmetic and Logic

The file arithmetic and logic (FAL) instruction performs copy, arithmetic, logic, and function operations on the data stored in files. The FAL instruction is an output instruction that performs the operations defined by the source addresses and the operators listed by the programmer in the expression field. The instruction writes the results into a destination address. The FAL instruction automatically converts the data type at the source addresses to the data type that is specified in the destination address. The FAL instruction performs operations such as zero a file, copy data from one file to another, make arithmetic or logic computations on data stored in files, and unload a file of error codes one at a time for display. Table 7-1 lists the operations performed by the A-B PLC5 FAL instruction.

Type	Operator	Description	Example
Copy	none	Copy from A to B	
Clear	none	Set a value to 0	
Arithmetic	+	Add	2+2
	-	Subtract	8-5
	*	Multiply	3*6
	/	Divide	12/4
	-	Negate	-N7:0
	SQR	Square root	SQR N7:1
	**	Exponential	10**2
Bitwise	AND	Bitwise AND	D9:3 AND D10:4
	OR	Bitwise OR	D9:4 OR D9:5
	XOR	Bitwise exclusive or	D9:6 XOR D9:7
	NOT	Bitwise complement	NOT D10:11
Conversion	FRD	Convert from BCD to binary	FRD D10:0
	TOD	Convert from binary to BCD	TOD N7:1

Table 7-1. FAL Operators for A-B PLC5 Family

To illustrate the operation of a FAL instruction, we will perform a FAL copy operation as shown in Figure 7-2.

In this example, when the rung goes true (bit I:000/02 set to 1), the processor reads the data stored in four words of integer file N71 starting at

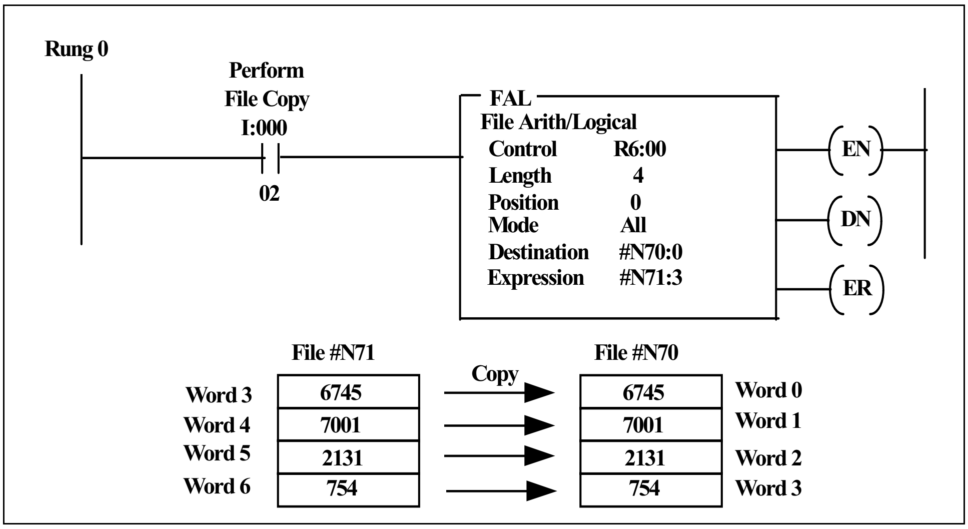


Figure 7-2. A-B PLC5 File-to-File Copy Example

word 3, and writes the data to integer file N70 starting at word 0. It writes over any data in the destination file.

EXAMPLE 7-1

Problem: Write a PLC5 LD program to copy the data in integer file N30, words 5, 6, and 7 to file N31 starting at word 2, if input bit I:000/03 is true.

Solution: The ladder diagram program to copy the data is shown in Figure 7-3.

File Search and Compare

The file search and compare (FSC) instruction is an output instruction that compares values in source files, word by word, for the logical operation that is specified in the expression. When the processor finds the specified comparison is true, it sets the found (FD) bit, and records the position where the true comparison was found. The inhibit (IN) bit is set to prevent any further searching of the files.

This instruction is used to perform operations such as set high and low process alarms for multiple analog inputs and compare batch variables against a reference file before starting a batch operation.

The FSC instruction performs the comparisons listed in Table 7-2 on file data according to the equation listed in the expression portion of the instruction.

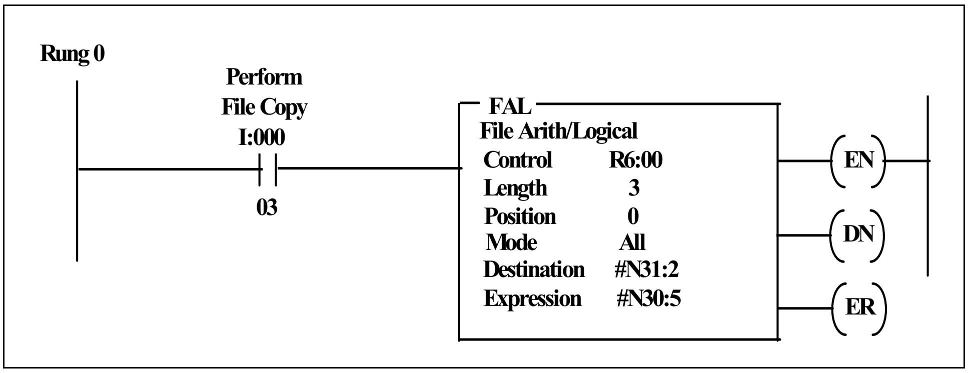


Figure 7-3. File-to-File Copy LD for Example 7-1

The processor compares files of different data types by internally converting data into its binary equivalent before performing the comparison.

Comparison	Example Expression
Search equal	#N50:0 = #N51:0
Search not equal	#N51:0 <> #N53:10
Search less than	#N51:0 < #N53:10
Search less than or equal	#N51:0 <= #N53:10
Search greater than	#N51:0 > #N53:10
Search greater than or equal	#F61:0 >= #N63:10

Table 7-2. FSC Comparison Instructions for A-B PLC5s

In file search, when the rung condition is true, the desired comparison is performed on data addressed in the expression. Words are compared in ascending order starting at the beginning. The rate is determined by the mode of operation that is specified in the FSC instruction. The done (DN) bit is set after the processor has compared the last pair. If the rung is true at completion, the done bit is turned off when the rung is no longer true. In the numerical mode, however, if the rung is not true at completion, the DN bit stays on one program scan after the operation is complete.

To illustrate the operation of a FSC instruction, we will perform an FSC "search not equal" operation as shown in Figure 7-4. When bit I:000/03 goes to true, the processor performs the "not-equal-to" comparison between words, starting at B3:0 and B15:0. The number of words compared per program scan is 10 in this example because the mode is set to 10.

When the processor finds that corresponding source words are not equal (words B3:4 and B15:4), the processor stops the search and sets the found

(FD) and inhibit (IN) bits. To continue the search comparison, the LD program must turn off the inhibit bit.

EXAMPLE 7-2

Problem: Write a PLC5 LD program to search the data in integer file N40, words 0 through 99 and compare for an equal condition to data in file N50 starting at word 0, if input bit I:000/03 is true.

Solution: The LD program to search and compare the data files is shown in Figure 7-5.

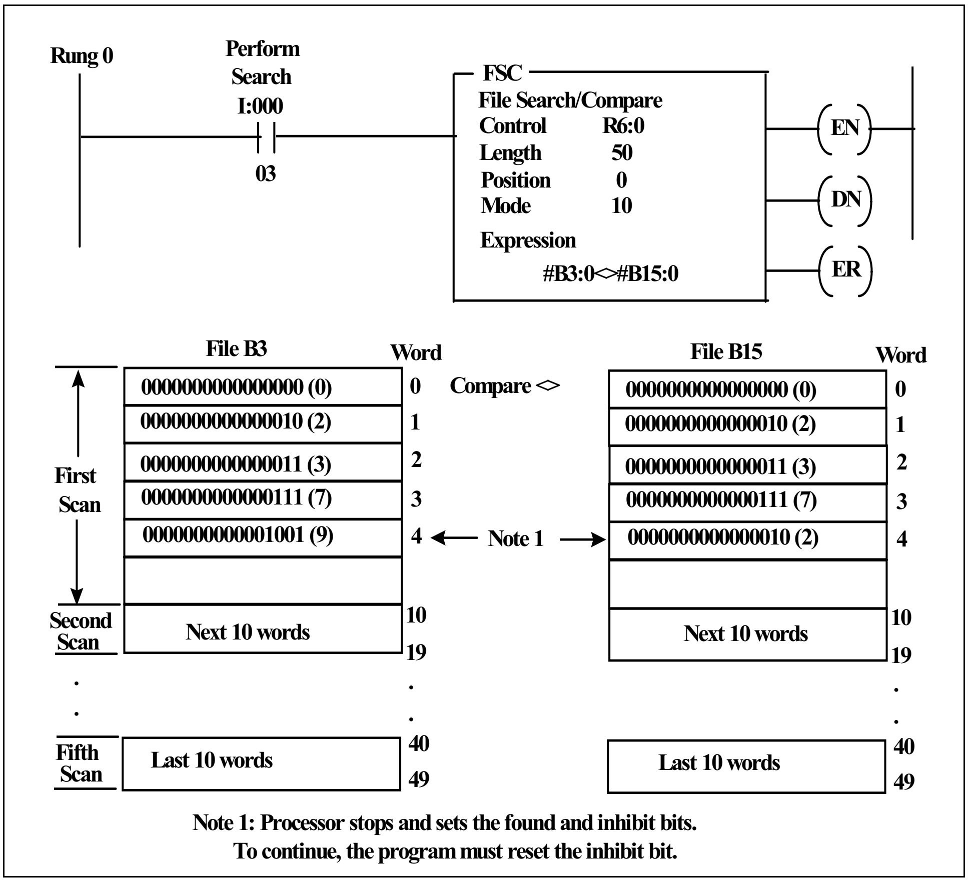


Figure 7-4. Typical A-B PLC5 FSC Instruction Example

File Copy

The file copy (COP) instruction is an output instruction that copies the values in the source file into the destination file. The source file remains unchanged. The COP instruction does not use status bits.

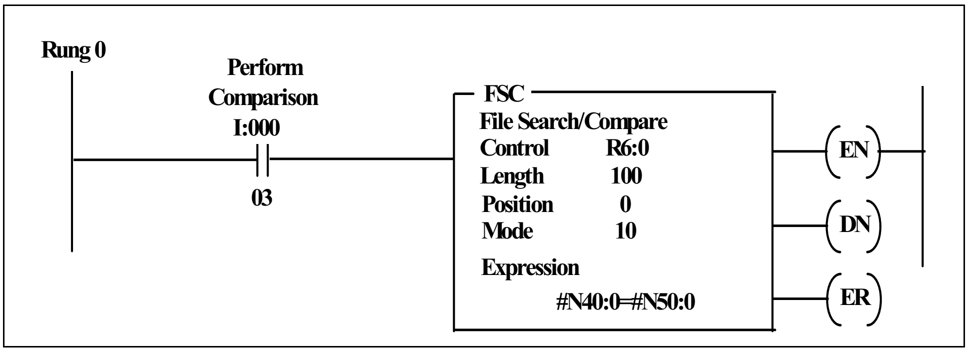


Figure 7-5. LD Program for Example 7-2

The COP instruction will not write over file boundaries so any overflow data will be lost. Also, no data conversion occurs so the source and destination files should use the same data type.

An example LD program using a COP instruction is illustrated in Figure 7-6. In this example, if input bit I:000/03 is true, the processor will copy the first 10 words starting at file N50:0 into the first 10 words of file N60:0.

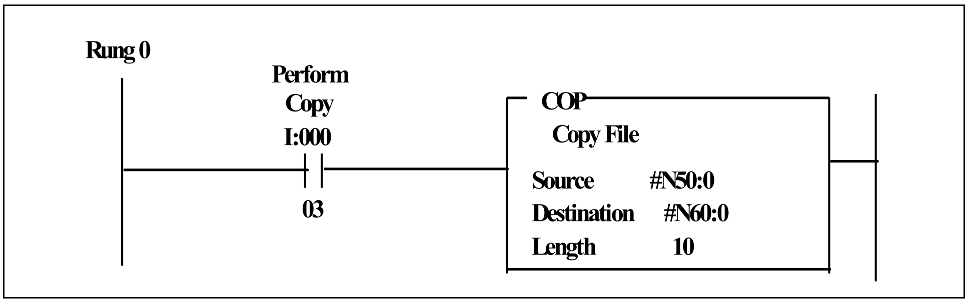


Figure 7-6. PLC5 File Copy Instruction

File Fill

The file fill (FLL) instruction is an output instruction that fills the words of a file with a source value. The source file remains unchanged. Like the COP instruction, the FLL instruction does not use status bits.

The FLL instruction will not write over file boundaries, so any overflow data will be lost. Also, no data conversion occurs so the source and destination files should use the same data type.

An example LD program that uses a FLL instruction is illustrated in Figure 7-7. In this example, if input bit I:000/03 is true, the processor will copy the first 10 words starting at file N50:0 into the first 10 words of file N60:0.

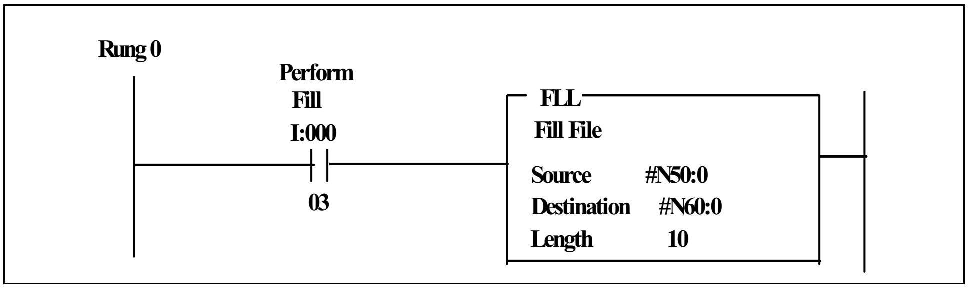


Figure 7-7. A-B PLC5 File Fill Instruction

Shift Register Instructions

Shift register instructions are used to track the movement or flow of parts and information in industrial applications. We will discuss four A-B PLC5 shift register instructions in common use: bit shift left (BSL), bit shift right (BSR), first-in, first-out load (FFL), and first-in, first-out unload (FFU).

The bit shift instructions, BSR and BSL, are used to load bits into, shift bits through, and unload bits from a bit array 1 bit at a time, such as for tracking bottles through a bottling line where each bit represents a bottle. The load and unload shift instructions, FFL and FFU, are used to load and unload values in the same order, such as for tracking parts through an assembly line where parts are represented by values that have a part number and assembly code.

Bit Shift Instructions

Bit shift instructions shift all bits within the specified address one bit position with each false-to-true ladder rung transition. There are two shift instructions: bit shift left (BSL) and bit shift right (BSR).

The structure of a BSL instruction is shown in Figure 7-8. This figure illustrates a BSL instruction with its parameters of file, control, bit address, and length.

The *file* is the address of the bit array that will be manipulated. The array must start at a 16-bit word boundary. For example, use bit 0 of word number 1, 2, 3, etc. The array can end at any bit number up to 15,999. However, the remaining bits in the last word of the array cannot be used because the instruction invalidates them.

The *control* is the address of the control structure (48 bits or three 16-bit words) in a control type (R) file that stores the instruction's status bits, the size of the array (number of bits), and the bit pointer. The processor uses this information to run the instruction.

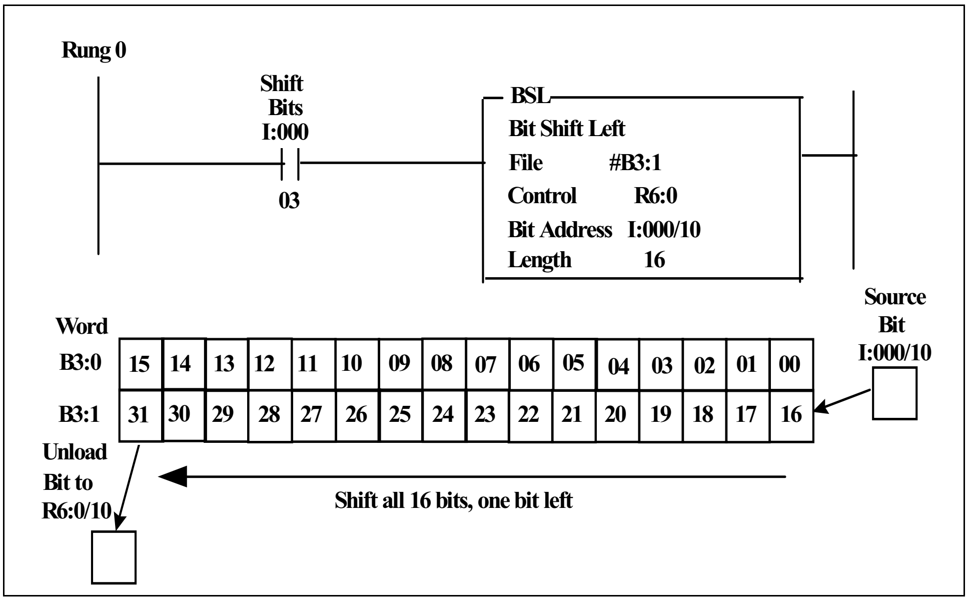


Figure 7-8. A-B PLC5 Bit Shift Left Instruction

The *bit address* is the address of the source bit. The bit shift instruction inserts the value (0 or 1) of this bit in either the first (lowest) bit position (for the BSL instruction) or the last (highest) bit position (for the BSR instruction) in the array.

The *length* is the decimal number of bits to be shifted. In A-B PLC5s the bits in I/O files are numbered in octal 00 to 07 and 10 to 17, but all other files are numbered in decimal 0 to 15.

The output coils to the right of the file instruction are the enable (EN) and done (DN) bits. These bits have the same word address as the instruction control element. The processor automatically sets the address of these status bits when the programmer enters the control address. The EN bit is set by a false-to-true rung transition and indicates that the instruction is enabled. The DN bit is set to indicate that the bit array shifted one bit position.

The control element also contains two other status bits: error (ER) at bit 11 and unload (UL) at bit 12. The ER bit is set to indicate that the instruction detected an error, such as if you entered a negative file length. The unload (UL) bit is the instruction's output. The UL bit stores the status of the bit removed from the array each time the instruction is enabled.

In the LD program shown in Figure 7-8, when the rung containing the BSL instruction goes from false to true, the processor sets the EN bit. Then the processor shifts 16 bits (length = 16) in bit file B3, starting with bit 16, to the left (higher bit number) one bit position. The last bit shifts out at bit position 31 into the control register UL bit (R6:0/10). The source bit comes from I:000/10 and shifts into the first bit position of B3:1.

After the processor completes the shift operation in one program scan, when the rung control bit I:000/03 goes false, the instruction resets the status bits EN, ER (if set), and DN.

For wraparound operation, the source address is assigned the same address as the highest (outgoing) bit address.

The operation of a shift instruction can be illustrated by an example problem.

EXAMPLE 7-3

Problem: Modify the BSL instruction shown in Figure 7-8, so that the bit in the left-most position of word B3:1 will be returned to the right-most bit position of the word during each shift operation.

Solution: The bit address in the BSL instruction must be changed to B3:1/00.

The structure of a BSR instruction is shown in Figure 7-9. This figure illustrates a BSR instruction with its parameters of file, control, bit address, and length. The instruction works in the same manner as a BSL instruction except the bits move to the right.

In the example in Figure 7-9, when the rung containing the BSR instruction goes from false-to-true, the processor sets the EN bit. Then the processor shifts 16 bits (length = 16) in bit file B3, starting with bit 31, to the right (to a lower bit number) one bit position. The last bit shifts out at bit position 16 into the control register UL bit (R6:0/10). The source bit comes from I:000/10 and shifts into the highest bit position, bit 31 of bit file B3.

After the processor completes the shift operation in one program scan, when the rung control bit I:000/03 goes false, the instruction resets the status bits EN, ER (if set), and DN.

For wraparound operation, the source address is assigned the same address as the highest (outgoing) bit address.

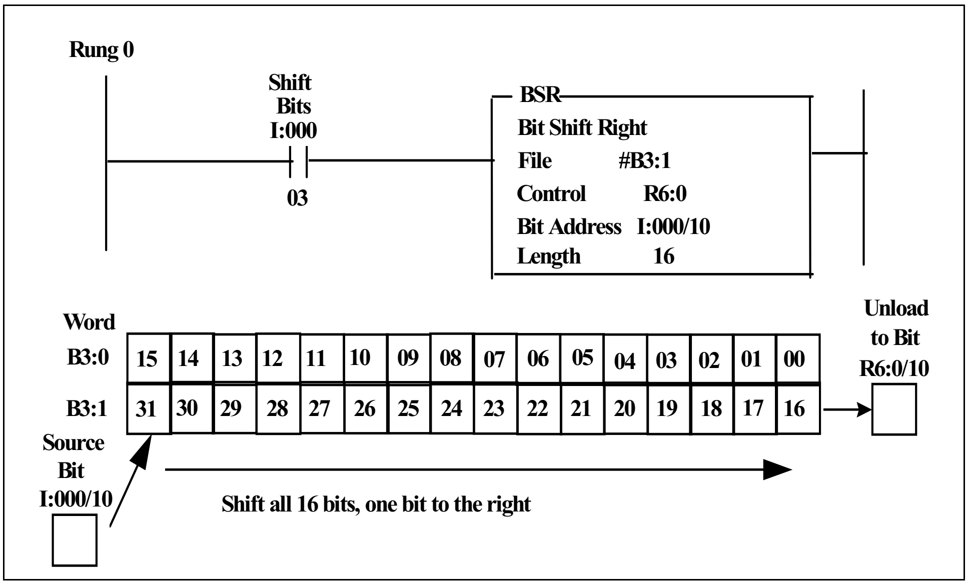


Figure 7-9. PLC5 Bit Shift Right Instruction

First-In, First-Out Instruction

There are two first-in, first-out (FIFO) instructions: the FIFO load uses the mnemonic FFL and the FIFO unload uses the mnemonic FFU. These two instructions, FFL and FFU, are used in pairs to store and retrieve data in a prescribed order. When used in pairs, these instructions establish an asynchronous shift register or stack.

The two FIFO instructions, load and unload, must use the same file and control addresses, length, and position values as shown in the example of Figure 7-10.

This figure illustrates the FFL and FFU instructions with their parameters of source, destination, FIFO, control, length, and position. Data is loaded starting at word 0 and increasing to word n. It is unloaded from word 0 and all words are shifted up one position toward word 0.

The *Source* is the address that stores the "next in" value to the stack. The FIFO load instruction (FFL) retrieves the value from this address and loads it into the next word in the stack.

The *Destination* is the address that stores the value that exits from the stack.

The *FIFO* is an indexed address of the stack. The same FIFO address is used for the associated FFL and FFU instructions.

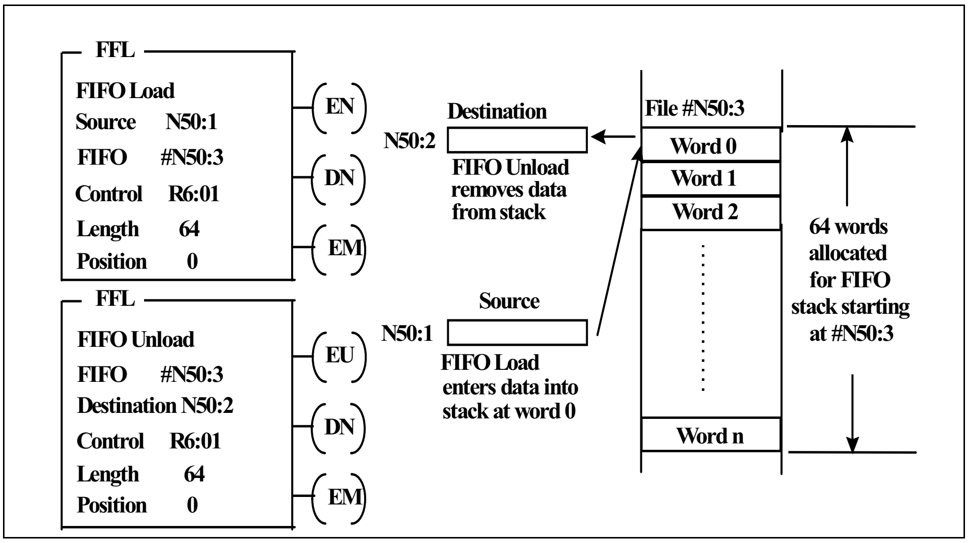


Figure 7-10. PLC5 FIFO Load and FIFO Unload Instructions

The *Control* is the address of the control structure (48 bits, or three, 16-bit words) in the control area of memory. The control structure stores the instruction's status bits, stack length, and next available position (pointer) in the stack.

The *Length* specifies the maximum number of words in the stack. The length is addressed by adding the mnemonic .LEN to the control address (R register).

The *position* indicates the next available location where the instruction loads data into the stack. The position is addressed by adding the mnemonic POS to the control address (R register). Normally the position value is 0, unless the programmer wants the instruction to start at an offset at power-up.

The output coils to the right of the FIFO instructions are the enable load (EN), enable unload (EU), empty (EM), and done (DN) bits. These bits have the same word address as the instruction control element (R). The PLC processor automatically sets the address of these status bits when the programmer enters the control address. The enable load (EN) bit used in FFL instruction is set by a false-to-true rung transition and indicates that instruction is enabled. The enable unload (EU) bit used in FFU instruction is set by a false-to-true rung transition and indicates the instruction is enabled. The done (DN) bit is set to indicate that the stack is full. The DN bit inhibits loading the stack until there is room. Empty (EM) is set by the processor to indicate that the stack is empty. The FIFO unload command should not be enabled if the EM bit is set.

The operation of the FIFO instructions can be illustrated by an example problem.

EXAMPLE 7-4

Problem: An assembly line produces 100 machine parts on each production run. The serial number for each part is located in word N40:1. Write a LD program to place the serial numbers into File N50:3. Download serial numbers to word N40:2.

Solution: The required LD is shown in Figure 7-11.

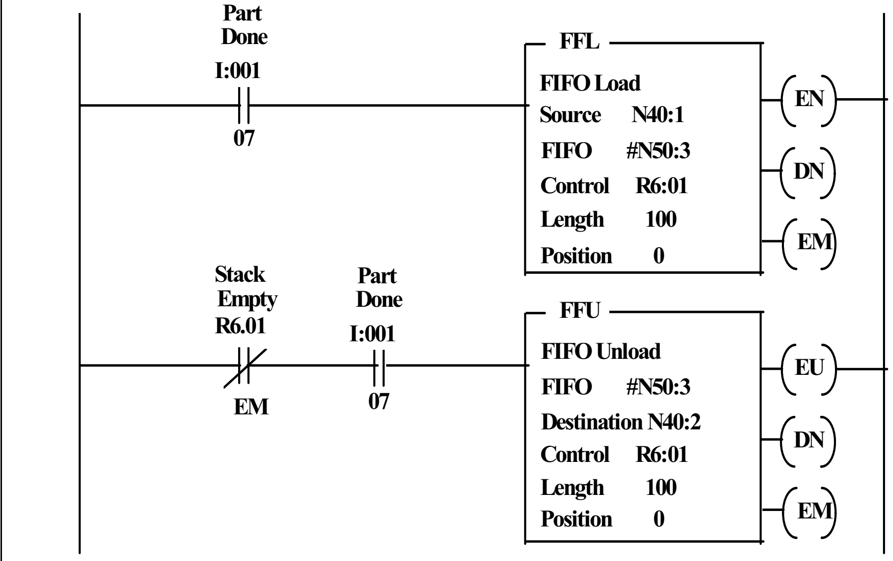


Figure 7-11. FFL and FFU Ladder Diagram for Example 7-3

Sequencer Instructions

The sequencer instructions are typically used to control automatic assembly machines that have a consistent and repeatable operation. There are three common sequence instructions: sequencer input, sequencer output, and sequencer load.

The sequencer instructions are generally used to transfer data from the memory to discrete output modules for the control of sequential process operations or sequential batch operations (sequencer output). They are also used to compare I/O word data with data stored in tables so that process-operating conditions can be examined for control and diagnostic purposes (sequencer input). The instruction is also used to transfer I/O word data into the memory (sequencer load).

The sequencer instructions can conserve program memory because they can monitor and control multiples of 16 discrete outputs at a time in a single rung of logic.

The sequencer input instruction (SQI) and the sequencer output instruction (SQO) are used in pairs to respectively monitor and control a sequential operation.

Figure 7-12 illustrates the three A-B PLC5 sequencer instructions. The parameters used in these instructions are file, mask, source, destination, control, length, and position.

The *file* is the indexed address of the sequencer file to or from which the instruction transfers data. Its purpose depends on the instruction.

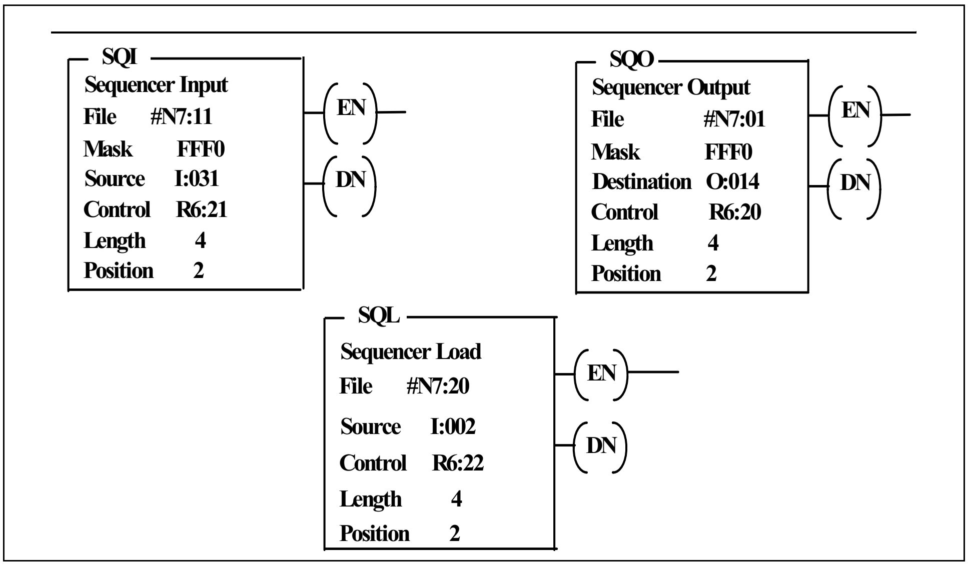


Figure 7-12. Sequencer Instructions SQI, SQO, and SQL for A-B PLC5s

The *mask* (for SQO and SQI) is a hexadecimal code or the address of the mask element or file through which the instruction moves data. Set mask bits to 1 to pass data; set mask bits to 0 to prevent the instruction from operating on corresponding destination bits. The programmer can specify a hexadecimal value to obtain a constant mask value. The programmer can also store the mask in an element or file if he wants to change the mask according to a control application requirement.

The *source* (for SQI and SQL) is the address of the input element or file from which the instruction obtains data for its sequencer file.

The *destination* (for SQO, only) is the destination address of the output word or file to which the instruction moves data from its sequencer file.

The *control* is the address of the control structure in the control area (R) of memory (48 bits, or three, 16-bit words) that stores the instruction's status bits, the length of the sequencer file, and the instantaneous position in the file.

The programmer should use the control address with the mnemonic to address the *length* (LEN) and *position* (POS) parameters. Length is the length of the sequencer file and Position is the current position of the word in the sequencer file that the processor is using.

The *length* is the number of steps of the sequencer file starting at position 1. Position 0 is the start-up position. The instruction resets to position 1 at each completion.

Note that the address assigned for a sequencer file is step zero. Sequencer instructions use (length +1) words of data for each file referenced in the instruction. This also applies to the source, mask, and destination values if addressed as files.

The *position* is the word location in the sequencer file. The position value is incremented internally by SQO and SQL instructions.

To illustrate the operation of sequencer instructions, we will perform a sequencer output (SQO) example in Figure 7-13. In this example, the SQO instruction moves the data of the current step (1) through a mask to an output word that is connected to an A-B PLC5 output module in rack 1, I/O group 4.

The SQO instruction steps through the sequencer file of 16-bit output words whose bits have been set to control the various output devices connected to the discrete output module shown. When the rung goes from false to true, the instruction increments to the next step (word) in the sequencer file #N7:1. Word N7:1 is the "home" position or step 0 and N7:2 is the word for step 1. When the sequence is finished, it will start over again at Step 1. The data in the sequencer file is transferred through a fixed mask (0F0F, HEX) to the destination address O:014. Current data is written to the destination element for every scan that the rung remains true.

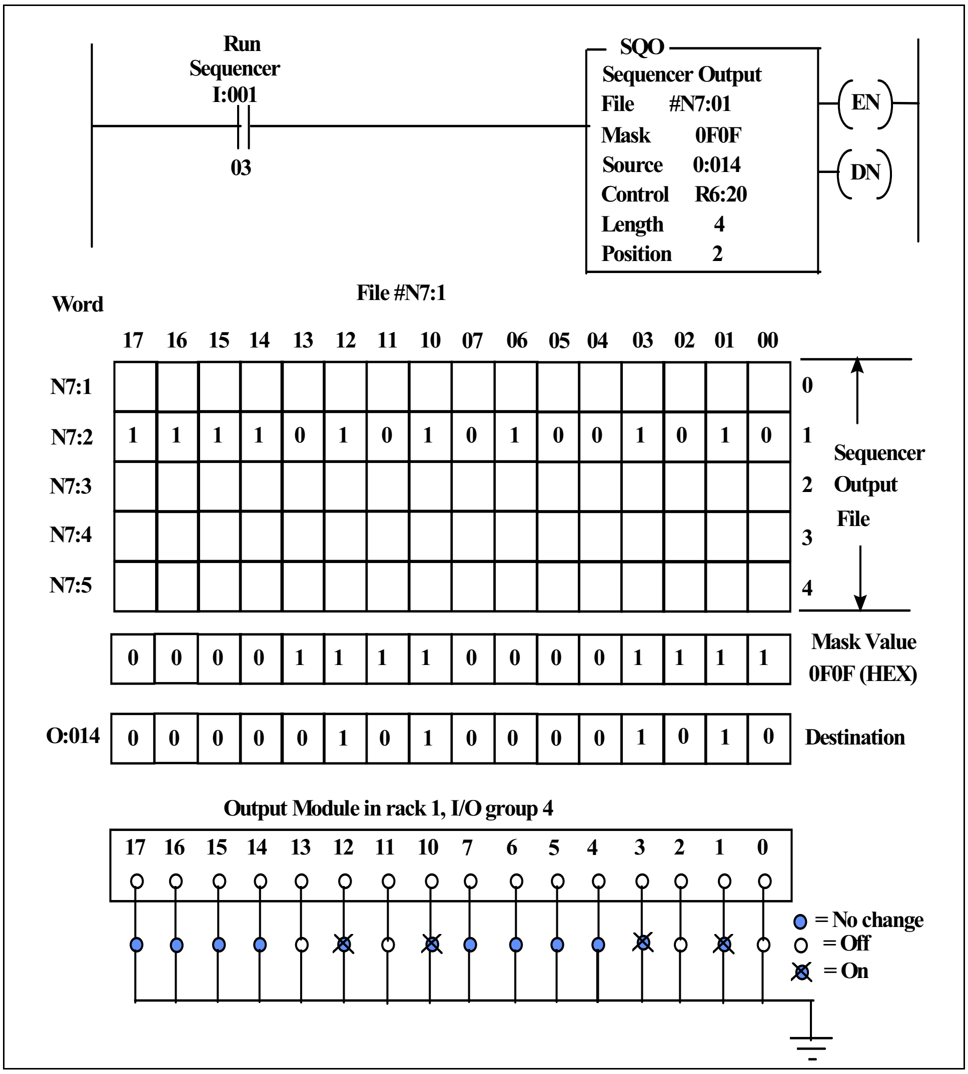


Figure 7-13. Sequencer Output (SQO) Example

The operation of a sequencer instruction can be illustrated by an example problem.

EXAMPLE 7-5

Problem: Assume the sequencer output instruction (SQO) shown in Figure 7-13 is used to control solenoid valves on two process tanks. Outputs 0 through 7 are connected to 8 solenoids on tank 1 and outputs 10 through 17 are connected to the solenoid valves on tank 2. To control or changes the status of the tank 2 only during process operation, what is the required value of the mask in the output instruction of Figure 7-13.

Solution: The mask value must be set to FF00.

Block Transfer Instructions

In larger PLC systems, there is a requirement to transfer large blocks of data between supervisor PLC processors and remote processors on a high-speed data communications network. In the A-B PLC5s, block transfers are performed using block-transfer write (BTW) and block-transfer read (BTR) instructions.

The basic A-B PLC5 processor in scanner mode can transfer up to 64 words at a time to or from a block transfer (BT) module in a local or remote I/O) chassis. Typical BT modules are thermocouple input modules, analog I/O modules, BCD I/O modules, and pulse counters. The block diagram in Figure 7-14 illustrates block transfers between a supervisory PLC5 in scanner mode to a remote I/O rack with BT modules installed. The remote I/O rack has an A-B adapter module, model number 1771-ASB that communicates internally to the BT modules using the rack data bus.

The A-B PLC5s can also transfer up to 64 words at a time between a supervisory process in scanner mode and a processor configured for adapter mode as shown in Figure 7-15. In this application, both processors simultaneously execute the opposite block transfer instruction.

Figure 7-16 shows a bi-directional alternating block transfer example. Using rungs like this example ensures that the block transfer requests are executed in the order in which they were sent to the queue. The processor alternates between the BTR and BTE instructions in the order in which they are scanned by virtue of the enable bit conditions in the rungs. Using the NC enabled bits from the two block transfer functions prevents the read and write block transfer instructions from queuing simultaneously. In this example, preconditioned NO instructions are used to the left of the

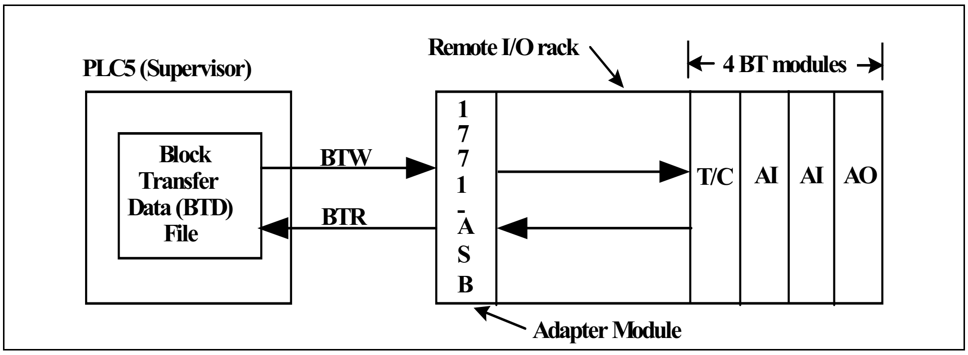


Figure 7-14. A-B PLC5 Block Transfer Operation in Scanner Mode

two EN bit contacts. These precondition bits allow time-driven or event-driven transfers.

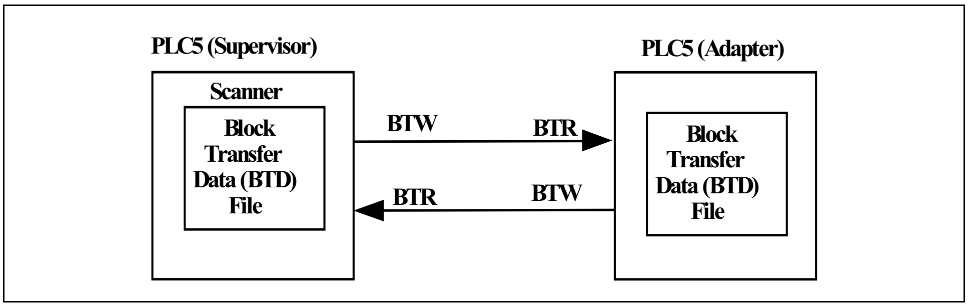

PLC5 (Supervisor)
Scanner
Block Transfer Data (BTD) File
BTW
BTR
BTR
BTW
PLC5 (Adapter)
Block Transfer Data (BTD) File

Figure 7-15. A-B PLC5 Block Transfer Operation in Adapter Mode

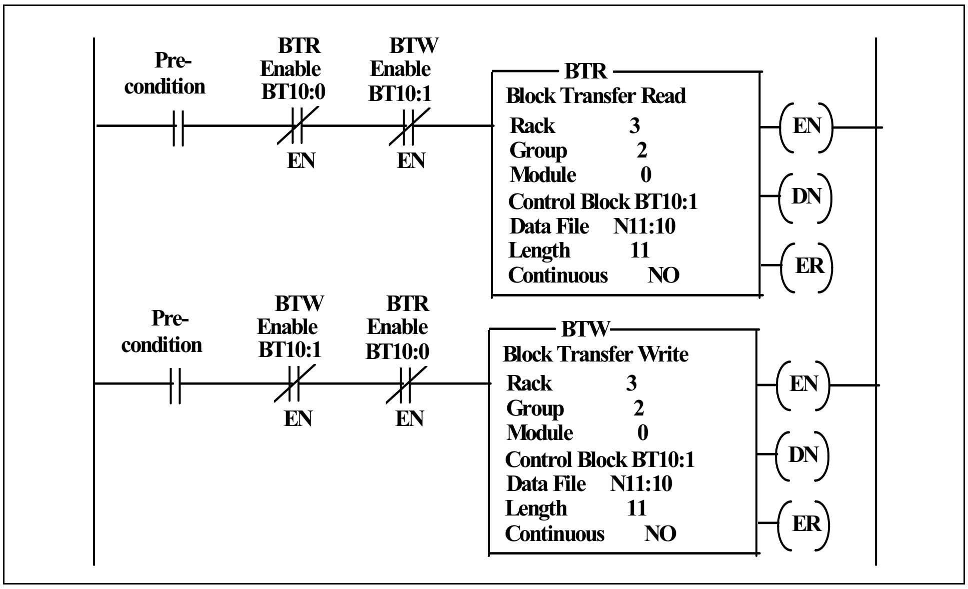


Figure 7-16. A-B PLC5 Bi-Directional Alternating Block Transfer

EXERCISES

7.1 List the most common advanced LD instructions encountered in the Allen-Bradley PLC5 family.

7.2 What is the definition of a file in a PLC memory system?

7.3 What is an Allen-Bradley PLC5 FAL instruction?

7.4 Write a PLC5 LD program to copy the process data located in integer file #N30, words 0, 1, 2, and 3 to integer file #N31 starting at word 0, if input bit I:001/01 is true.

7.5 Write an Allen-Bradley PLC5 LD program to review the data in integer file #N50, words 0 through 20 and compare the data for an equal condition to the data stored in file #N60 starting at word 1.

7.6 Explain the purpose and function of the control, length, position, mode, destination, and expression in a FAL file instruction.

7.7 What are Allen-Bradley PLC5 LD shift register instructions typically used for in industrial applications?

7.8 List the three common sequencer instructions and explain their purpose and use.

7.9 Define the parameter length in a sequencer file.

7.10 What is the position parameter in a sequencer file?

7.11 Modify the bit shift right (BSR) instruction shown in Figure 7-9 so that the bit in the left-most position of word B3:1 will be returned to the right-most bit position of the word during each shift operation

7.12 Assume the sequencer output instruction (SQO) shown Figure 7-13 is used to control solenoid valves on two process tanks. Outputs 0 through 7 are connected to 8 solenoids on tank 1 and outputs 10 through 17 are connected to the solenoid valves on tank 2. To control or change the status of tank 1 only during process operation, what is the required value of the mask in the output instruction of Figure 7-13?

BIBLIOGRAPHY

1. *PLC-5 Programmable Controller: Instruction Set Reference*, Rockwell Software, Inc., 1996.
2. Reis, R. A., and Webb, J. W., *Programmable Controllers: Principles and Applications*, Prentice-Hall, Inc., 3rd edition, 1995.
3. Bryan, E. A., and Bryan, L. A., *Programmable Controllers: Concepts and Applications*, Industrial Text Co., 1st edition, 1988.

8 Standard PLC Programming Languages

Introduction

In this chapter, we will discuss the international standard for PLC Programming Languages. This standard lists five PLC languages: Ladder Diagram (LD), Function Block Diagram (FBD), Sequential Function Chart (SFC), Instruction List (IL) and Structured Text (ST). We will cover three of the standard languages, Sequential Function Chart, Instruction List, and Structured Text. LD is the most widely used PLC language and it was discussed in Chapters 6 and 7. Functional Block Diagram language will be covered in the next chapter.

International Standard for PLC Languages

In the early 70s, different national and international committees proposed numerous PLC programming standards to develop a common interface for programmable controllers. Then, in 1979, a working group of international PLC experts was appointed by various national committees to write a first draft of a comprehensive PLC standard. The first committee draft was issued in 1982.

After an initial review of the document by the national committees, it was decided that the standard was too complex to handle as a single document. As a result, the working group was split into five task forces, one for each part of the standard. The subject of each part is as follows: Part 1, General Information; Part 2, Equipment and Testing Requirements; Part 3, Programming Languages; Part 4, User Guidelines; and Part 5, Communications. Each task group consisted of several international experts, each backed by a national advisory group. IEC 61131-3, the standard for PLC programming languages, was issued by the IEC in March of 1993.

The IEC 61131-3 standard has three graphical languages: Ladder Diagram (LD), Function Block Diagram (FBD), and Sequential Function Chart (SFC), and two text-based languages: Instruction List (IL) and Structured Text (ST). The PLC language standard allows different parts of an application to be programmed in different languages that can be combined into a single executable program.

LD is the most common programmable controller language and it consists of a set of instructions that will perform the most basic type of control functions: relay type logic, timing and counting, and basic math operations. However, depending on the programmable controller model, the instruction set may be extended or enhanced to perform other operations. These additional functions are used for analog control, data manipulation, reporting, complex control logic, and other functions.

FBD is a graphical programming language that uses logic blocks similar to those used in Boolean algebra to represent basic logic. It also uses more complex function blocks to perform operations such as timing, counting, math, loading data, transferring data, and data comparison. The programmer is able to build complex control schemes by using functions from the FBD library and then interconnecting them in a graphical diagram area.

IL is a low-level programming language. It is very effective for small, simple applications or for optimizing parts of an application. Instructions always relate to the current result and the operator indicates the operation that must be made between the current value and the operand. The result of the operation is stored again in the current result.

ST is a high-level structured language designed for automation processes. This language is used mainly to implement complex procedures that cannot be easily expressed with graphical languages. ST is the default language for the description of the actions within steps and conditions attached to the transitions of the SFC language.

SFC is a graphical language used to describe sequential operations. The control process is represented as a set of well-defined steps linked by transitions. A Boolean logic condition is attached to each transition. Actions within the steps and the logic transitions between the steps can be performed by using instructions from the other standard PLC languages. However, the LD and IL logic instructions are mainly used to perform the logic transitions between steps.

Sequential Function Chart Language

The SFC language is used to describe operations of a sequential nature. It uses a simple graphic representation of the different steps of a process, and logical conditions that enable the transitions between active steps. The SFC language is a core or main IEC 61131-3 standard PLC language. Instructions from other standard languages are used to describe the actions within the steps and the logical conditions for transitions between steps within a SFC program.

The example SFC program shown in Figure 8-1 is used to illustrate the symbols used in a typical program. The top of the program contains a step block that is the *initial step*, where the programmable controller starts function chart execution and returns to this step from the end of the program unless directed otherwise by the program logic. A double-sided box identifies this block.

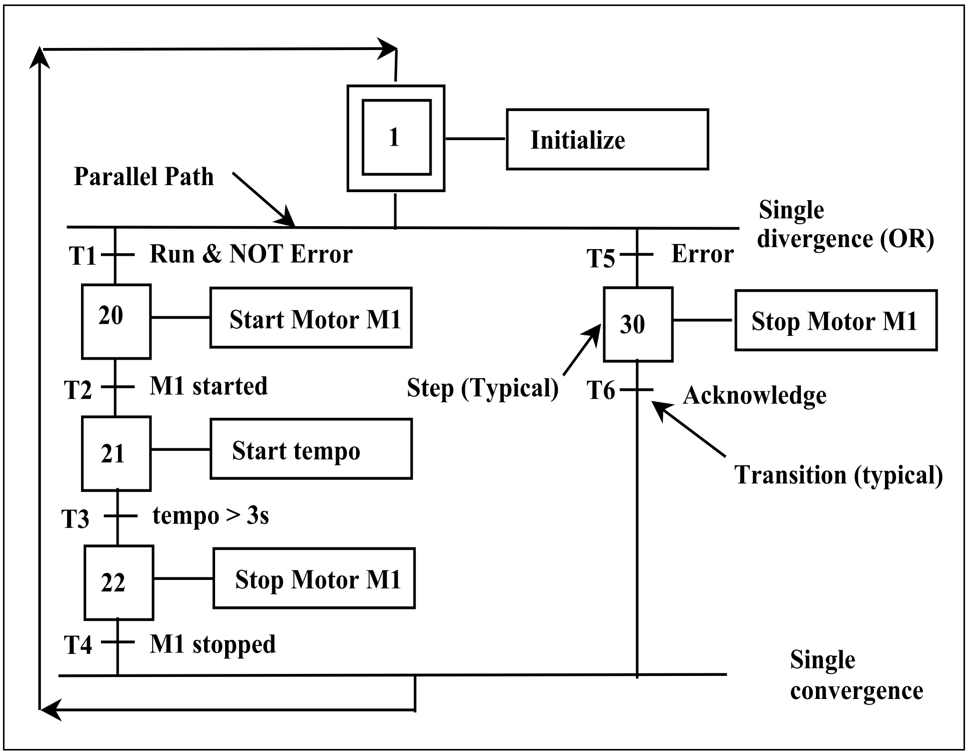


Figure 8-1. Typical SFC with Single Divergence and Single Convergence

The *step* block is the function chart's basic unit and contains ladder logic for each independent stage of the process or machine operation. A single-sided box identifies it with the step number inside the box.

The *transition* is the logic condition that the processor checks after completing the active step. When the transition logic is true, the step preceding the transition is disabled and the step following it becomes active. The transition is normally a single LD logic rung or IL statement identified by a short horizontal line below its corresponding step. In Figure 8-1, there are six transitions, labeled T1 through T6.

The *OR path* is identified by a single horizontal line at the beginning and end of a logic zone as shown in Figure 8-1. The processor selects one of several parallel paths depending on which transition goes true first.

The *AND path* is identified by a horizontal double line at the beginning and end of a zone as shown in Figure 8-2.

In SFC programs, steps and transitions are arranged in series and parallel paths, and they are numbered with the file numbers that contain their logic. The programmable controller scans the logic of a step repeatedly until its transition logic goes true. Then the program scan moves to the next step or steps and the previously active step is turned off.

There are three basic rules for standard sequential function chart programs. The first is that the *initial* step is always activated at start-up. When restarting from the beginning of the chart and on subsequent passes through the flowchart program, the programmer does have the option of restarting from the beginning of the last active step(s), following or changing the programmable controller's mode from run to program and back to run again.

The second rule is that a transition is tested after its associated step and operations pass from one step to the next through a transition when the transition goes true. The third rule is that after a true transition, the processor scans the step once more to reset all timer instructions and then executes the next step. This extra processor scan is called *postscan*. It is important to note that the processor never postscans a transition file so timers probably should not be used in transition files.

The processor scans a SFC program from left to right and top to bottom. When the SFC program scan encounters active parallel steps, it executes the ladder logic in the left-most step first, then moves to the ladder logic in the next parallel step across the screen from left to right.

SFC Application

To illustrate an SFC application, consider a simplified example of a semiautomatic punch shown in Figure 8-3. We assume the punch starts in the raised position or top position. When the operator depresses the start

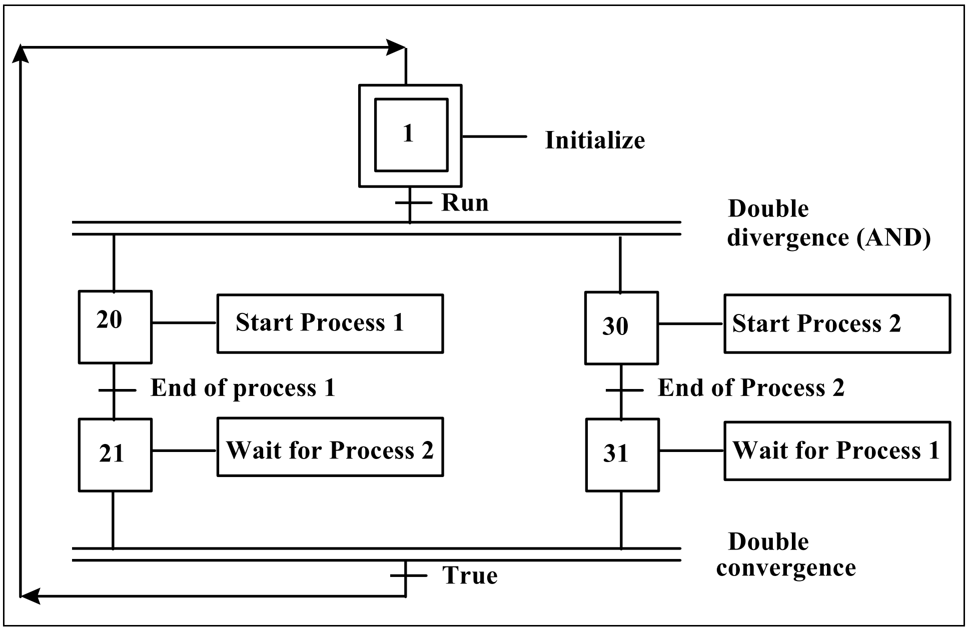


Figure 8-2. Typical SFC Program with Double Divergence and Convergence

pushbutton (PB), the punch is lowered and it pierces the metal part at the lowest or bottom position. The cycle is completed when the control system raises the punch back to the top position and the operator removes the punched metal part and inserts a new piece of metal for the next operation.

The semiautomatic punch control system shown in Figure 8-3 has four process states or steps. The first process step is the punch at rest in the raised position. The second step is the punch at rest waiting for a fault to be acknowledged. The third step is the punch descending and the final step is the punch ascending.

In the SFC program that controls the operation of the punch (see Figure 8-4), the first process step is labeled as 1. When the punch and associated control system are activated, the programmable controller places the system in this wait state (step 1) and waits for the first transition logic (T1) to be satisfied. If the start PB is depressed AND the punch is in the top position AND there is no fault present, the first transition is true and step 20 is activated, so the punch moves down. At the same time, the programmable controller turns OFF the first state.

When the punch activates the bottom limit switch, the second transition (T2) becomes true and the punch raises (Step 21). Finally, when the punch reaches the top, the T2 transition bit is set and the control program returns to the wait state (Step 1).

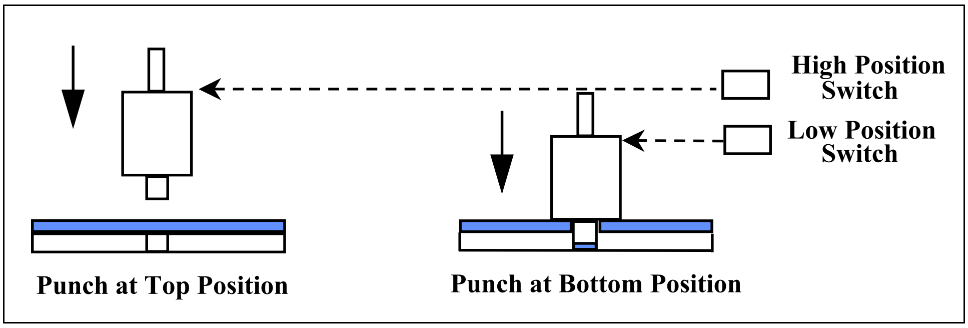


Figure 8-3. Semiautomatic Metal Punch Example

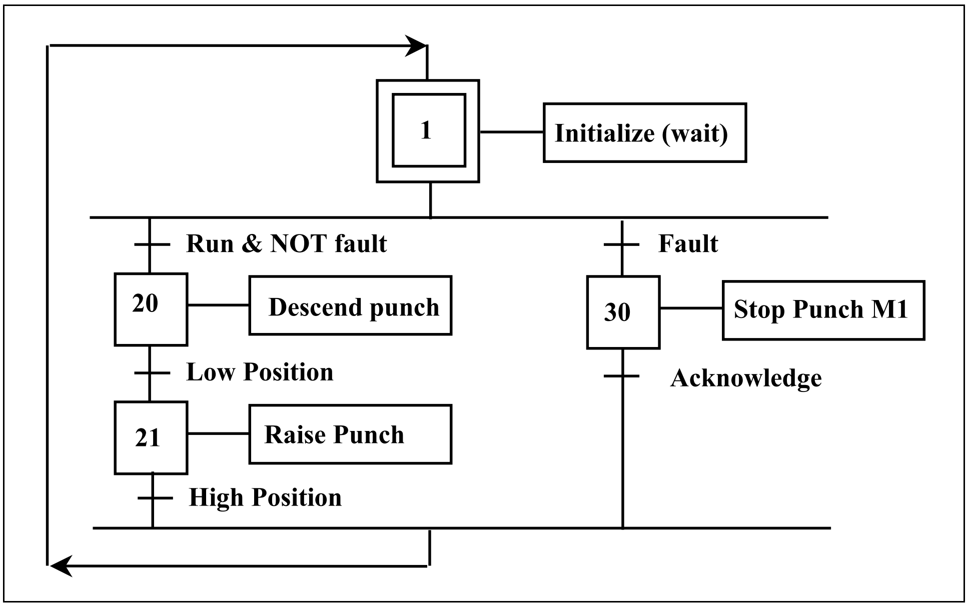


Figure 8-4. SFC Program for Semiautomatic Metal Punch Control

Step 30 (Punch Stopped) occurs when the punch is activated but there is a fault present. In this case, the third transition (T3) is true and step 30 is activated, and the punch is stopped waiting for the fault to be acknowledged. If the fault is acknowledged, the control program returns to the initialized state (Step 1) and waits for a new start command.

A typical example of the modification or expansion of a SFC program is given in Example 8-1.

EXAMPLE 8-1

Problem: Modify the SFC program shown in Figure 8-4 to remove completed production part and to insert a blank part into position under the punch.

Solution: The modified SFC program is shown in Figure 8-5.

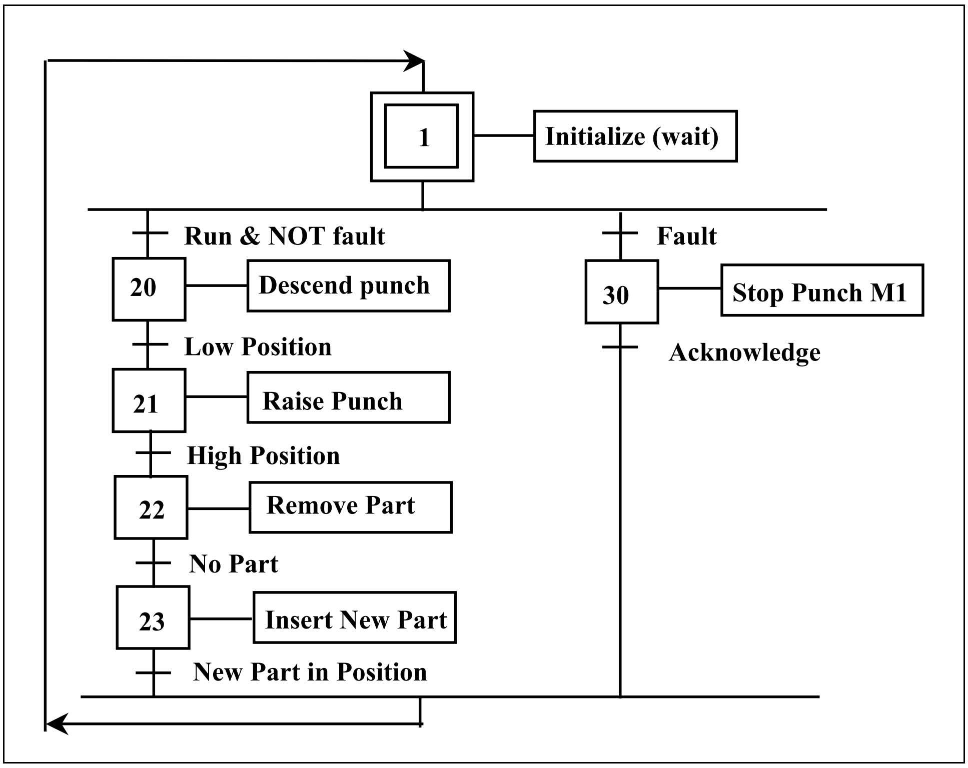


Figure 8-5. Modified SFC Program for Metal Punch

Structured Text Language

In this section, we will discuss structured text (ST) a high-level language designed for complex automation processes. This language is used mainly to implement complex procedures that cannot be easily expressed with graphical languages, such as LD and FBD or the simpler IL language. The ST can also be used to implement the process steps and transition conditions in SFC programs.

An ST program is a list of programming statements. Each statement ends with a semi-colon (";") separator. Names used in the source code, such as variable identifiers, constants, language keywords, are separated with inactive separators (space character, end of line, or stops) or by active separators that have a well-defined significance (e.g., the ">" separator indicates a "greater than" comparison). Programming comments can be freely inserted into a program list for clarity, but a comment must begin with the two charters '"*" and end with the two characters "*)". Each statement in the program terminates with a semi-colon (";") separator character.

The following are basic statement types for ST programs:

1. *Assignment* statement (e.g., variable:=expression;),
2. *Subprogram or function* call,
3. *"C" function block* call,
4. *Selection* statements (IF, THEN, ELSE, CASE, etc.),
5. *Iteration* statements (FOR, WHILE, REPEAT, etc.),
6. *Control* statements (RETURN, EXIT, etc.), and
7. *Special* statements for links with other languages such as SFC.

Inactive separators may be freely entered between active separators, constant expressions, and identifiers to improve readability. The ST inactive separators are *space* (blank character), *tabs*, and *end of line* characters. Unlike line-formatted languages, such as IL, end of lines may be entered anywhere in the program. This greatly increases the readability of the program. The programming tips listed below should be followed when using inactive separators to increase ST program readability:

1. Do not write more than one statement on one line,
2. Use tabs to indent complex statements, and
3. Insert comments to increase readability of lines.

Figure 8-6 shows an example of a ST program with low readability on the left side and the same program with high readability. This simple example shows how to improve the readability of a ST program.

ST Expressions and Parentheses

ST expressions combine operators and variables or constant operands. For each single expression (combining operands with one ST operator), the

Low readability ST Program

```
imax:=max_ite;cond:=X12
if not(cond(*alarm*))
then return;end_if;
for i (*index*):=1 to max_ite

end_if;end_for;
(*no effect if alarm*)
```

Same ST program with high readability

```
(*imax: number of iterations*)
(*i: FOR statement of index*)
(*cond: process validity*)

 imax:=max_ite;
 cond:=X12
 if not(cond(*alarm*))
 then return;
 end_if

(*process loop*)
for i (*index*):=1 to max_ite
do if i<>2 then
SPcall();
end_if;
end_for;
```

Figure 8-6. ST Program Readability Example

type of the operands must be the same. The following are examples of valid and invalid expressions:

1. Expression: (boo_var1 AND boo_var2); this expression is valid because we have two Boolean variables (boo_var1 and boo_var2) with a Boolean AND operand so that all items in the expression are the same type.
2. Expression: NOT(boo_var1); this is a valid expression because we have a Boolean variable (boo_var1) and a Boolean NOT operand.
3. Expression: (1s23 + 1.78); this is not a valid expression because there is a mixture of types used.

Parentheses are used to isolate parts of the expression and to explicitly order the priority of the operations. If no parentheses are used in a complex expression, the operational sequence is implicitly given by the default priority between ST operators. For example, 2+3*6 equals 2+18=20 because the multiplication operator has a higher priority than the add operator. However, if parentheses are used on the same equation, we have (2+3)*6 equals 5*6=30 because priority was given by the parenthesis.

Instruction List Programming

Instruction List (IL) is a textual programming language that can be used to create the code for a PLC control program. Its syntax for statements is similar to microprocessor assembly language and consists of instructions

EXAMPLE 8-2

Problem: Write an ST expression to multiply the sum of the numbers five and fifteen by the sum of the numbers four and six.

Solution: The valid ST expression is (4+6)*(5+15).

followed by addresses on which the instructions act. The IL language contains a comprehensive range of instructions for creating a complete user program. For example, in the Siemens S7 programming software package, there are over 130 different basic IL instructions and a wide range of addresses available depending on the model PLC used.

In this section, we will discuss the IL language used to program Siemens S7-300 and S7-400 programmable controllers. Most other PLC manufacturers use a similar version of IL. This section will provide the basics of Siemens IL programming with some typical application. However, the Siemens reference manual for Simatic S7 IL programming should be consulted for detailed information on the use of the software.

IL Statement Structure

IL instruction statements have two basic structures. The first is a statement made up of an instruction alone (e.g., NOT) and the second is a structure where the statement is made up of an instruction and an address. This second case is the most common structure.

The address of an instruction statement indicates a constant or a memory location where the instruction finds a value on which to perform an operation. The address can have a symbolic name or an absolute memory designation. The address can point to a number of items, such as a constant, a bit in a status, a symbolic name, a memory data block, and so on.

Table 8-1 shows the use of a constant value (+77) and a character string (END) as the address of an instruction. The first instruction loads the constant value or integer 77 into accumulator 1 in the CPU of the PLC. The Siemens S7 series programmable controllers have two 32-bit accumulators. These accumulations are general-purpose registers that are used to process bytes, words, and double words. Constants or values are loaded from PLC memory as addresses into the accumulators and logic operations are performed on them. The control program can also transfer the results of an operation from the first accumulator (accumulator 1) to a PLC memory location. The structure of accumulator 1 or 2 is given in Figure 8-6. Bits 0 through 7 are the lower byte of the lower word, and bits 8 through 15 are the upper byte (High Byte) of the lower word (Low Word). The High Word is bits 16 through 31 and has the same Low and High bytes as shown in Figure 8-7. The second instruction loads the ASCII character "END" into accumulator 1. At the same time, the CPU moves the integer 77 into accumulator 2.

IL Instruction	Description
L +77	Load the integer 77 into accumulator 1
L "END"	Load the ASCII character "END" into accumulator 1

Table 8-1. Examples of Constant Values as an Address

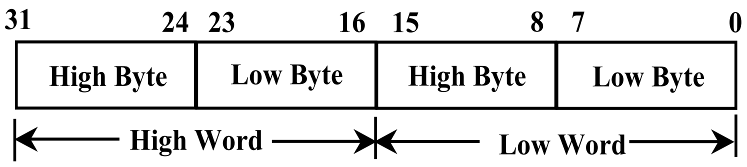


Figure 8-7. Word and Byte Structure for Accumulators

The address of an IL instruction can also refer to one or more bits in the status word of the programmable controller. The two instructions listed in Table 8-1 are two examples of an instruction operating on status word bits.

The "Status" word structure is shown in Figure 8-8. Bit cell 0 of the status word contains the "First Check" (FC) bit. The bar over the FC in bit 0 indicates that the bit is negated.

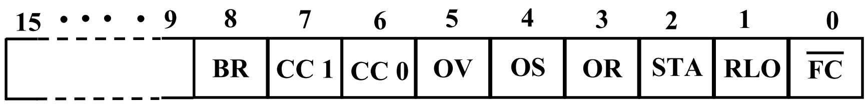


Figure 8-8. Status Word Structure

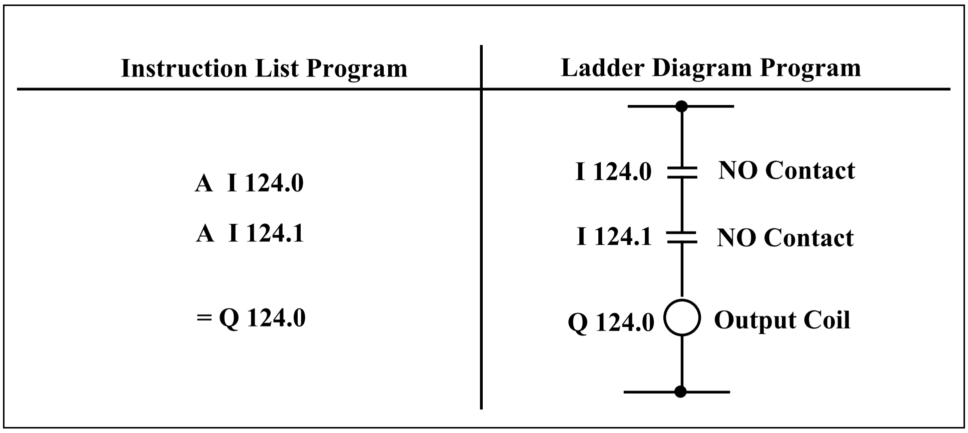
Bit cell 1 contains the "result of logic operation" (RLO) bit. This bit cell stores the result of a bit logic instruction or math comparison. Bit cell 2 contains the Status (STA) bit. This cell stores the value of a bit that is referenced in a program. Bit cell 3 stores the "Or Operation" status bit. The overflow (OV) bit indicates a fault encountered during the execution of a math instruction or a floating-point comparison instruction. The stored overflow (OS) bit is set together with the OV bit when a fault occurs. The "CC 1" and "CC 0" bits (condition codes) provide information on the following: 1) results of a math operation, 2) results of a comparison, 3) results of a digital operation, or 4) that bits have been shifted out by a shift or rotate command. The binary result (BR) bit forms a link between the processing of bits and words.

Bit Logic Instructions

The basic type of IL instructions is the Bit Logic Instructions. These instructions perform logic operations on single bits in PLC memory. The basic Bit Logic Instructions are as follows: "AND" (A) and its negated form "And Not" (AN); "OR" (O) and its negated form (ON); and "Exclusive Or " (XO) and its negated form "Exclusive Or Not" (XN). These instructions check the signal state of a bit address to establish whether the bit is activated (logic 1) or not activated (logic 0). These instructions can also be used to check the status of count or time value in timer or a counter instruction. If the time or count value is greater then 0, the result of the logic operation (RLO) bit is set to 1. If the time or count value is 0, the RLO bit is set to 0.

AND Logic Operation

The result of an *AND Logic Operation* is logic 1 only if all inputs are logic 1. Figure 8-9 is an example of "AND" logic operation with two inputs. The IL program is listed on the left side and the LD program is shown on the right side for comparison. The IL program uses an AND (A) on the two binary input bits at I124.0 and I124.1 and if both bits are 1 then the output bit Q124.0 is set to 1. The LD program is shown for comparison, with two normally open instructions in series, when the logic state of both the input bits is 1, the output coil is 1. This is the same AND function as the IL program on the left side of Figure 8-8.



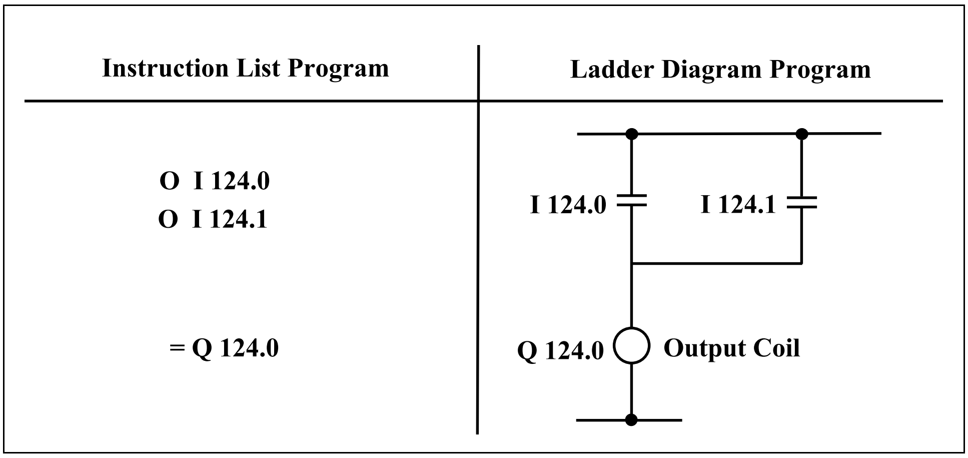
```
A  I 124.0
A  I 124.1

= Q 124.0
```

Figure 8-9. Comparison of IL and LD "AND" Function Programs

Or Logic Operation

The result of an *Or Logic Operation* is logic 1, if any or all inputs are logic 1. Figure 8-10 is an example of Or logic operation on two inputs, the IL program is listed on the left side and the LD program is shown on the right side for comparison of the same program in the two different programming languages. In this example, the IL program uses two Or (O) instructions to perform a logical Or operation on inputs I124.0 and I124.1. The LD program uses two normally open (NO) contact instructions in parallel, connected to an output coil to perform an Or logic operation. In both programs, when the logic state of one or both of the inputs is 1, the output bit, Q124.0 is set to 1. If both inputs are 0, the output is set to 0.



Instruction List Program

```
O  I 124.0
O  I 124.1

 = Q 124.0
```

Figure 8-10. Comparison of IL and LD "Or" Function Programs

Exclusive Or Logic Operation

The result of a two *Exclusive Logic Operation* is logic 1 only if one of the inputs is 1, but the result is 0 if both inputs are logic 0 or 1. The use of an "Exclusive Or" instruction in an IL program is shown in the left side of Figure 8-11. The corresponding circuit in a LD program is shown on the right side of Figure 8-11. In these example programs, the output Q124.0 is logic 1 only if inputs I124.0 and I124.1 have different logic values. The output is 0 if the two inputs have the same logic value.

The equivalent two-input Exclusive Or LD program is shown on the right side of Figure 8-11 for comparison.

And Not Logic Operation

The *And Not Logic Operation* negates the logic input before performing an AND operation with another logic input. The use of the IL And Not (AN) instruction is shown in Figure 8-12. The IL program is listed on the left

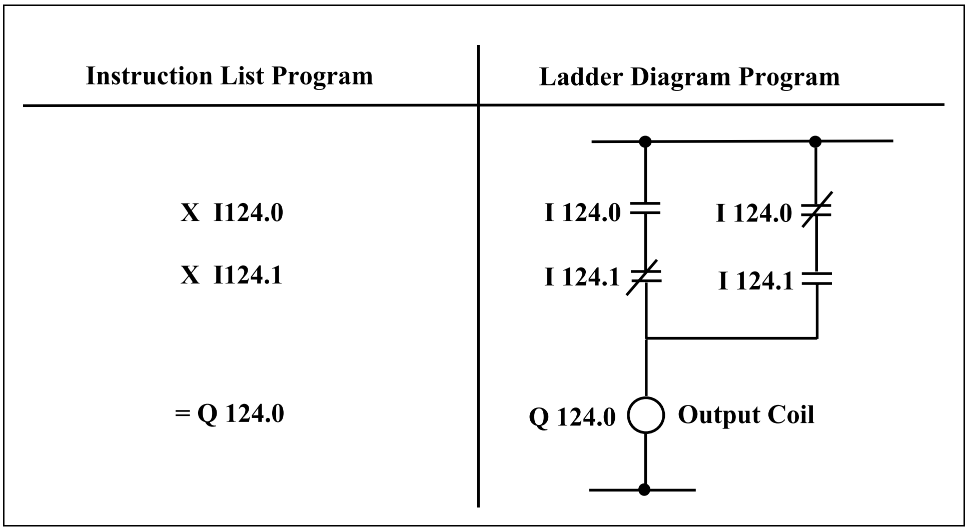


Figure 8-11. Examples Exclusive Or Instructions in IL and LD Programs

side of the figure and the equivalent LD program is shown on the right side for comparison. In this example, the IL program uses AND instructions on input bits I124.0 and I124.1. The output is set to logic 1 only if both input bits are logic 0. This is the same logic operation as performing an AND operation on two normally closed relay contacts in a relay ladder logic circuit as shown in Figure 8-12.

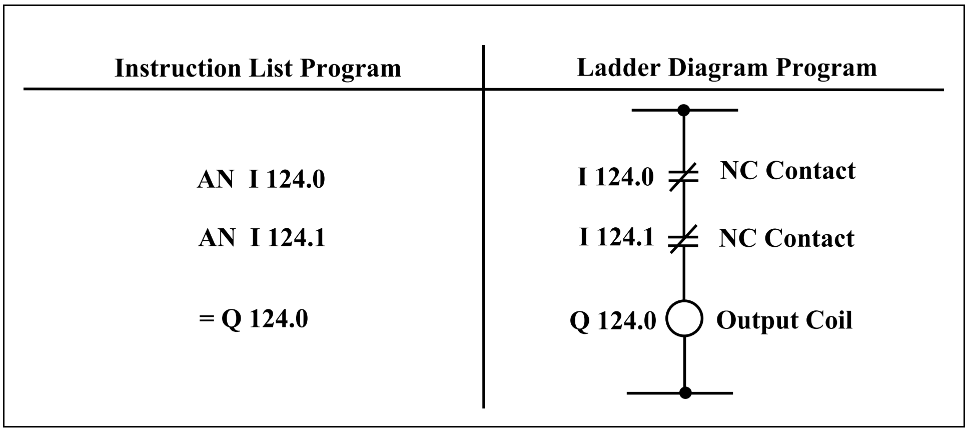


Figure 8-12. Example of the Use of IL "AN" Instructions

Or Not Logic Operation

The *Or Not Logic Operation* negates its logic variable before performing an OR operation on another logic variable. Figure 8-13 is an example of the use of a statement list Or Not (ON) logic instruction. The IL program is listed on the left side and the equivalent LD program is shown on the right side for comparison of the two programming languages. The IL program uses two Or Not (ON) instructions to perform a logical operation on inputs I124.0 and I124.1 to activate the output Q124.0. The LD program uses two NC contact instructions in parallel connected to an output coil to perform the logic operation. In both programs, when the logic state of one or both of the two input bits is 0, is the value of output bit, Q124.0 set to 1.

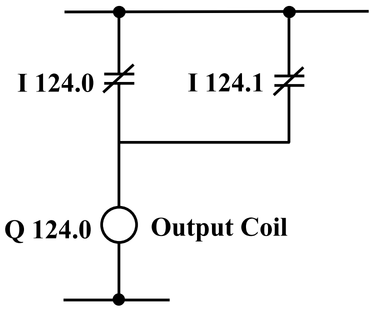
Instruction List Program	Ladder Diagram Program
ON I 124.0 ON I 124.1 = Q 124.0	

Figure 8-13. Use of the Or Not (ON) IL Instruction

Boolean logic operations can be performed on portions of a logic string that are enclosed in parentheses. These logic operations are called nesting expressions. Parentheses around a portion of a logic string means the program will perform the operations inside the parentheses before performing the logic operation indicated by the instruction that precedes the nesting expression.

AND Or statements can also be combined in a Boolean logic string without using parentheses. By convention, the AND statements are evaluated first and the results are then combined according to the OR truth table.

Timer Instructions

Software timers provide the same functions as hardware timers in process control applications. A typical application for a software timer instruction is to delay an operation for a fixed time interval. For example, the starting of a pump might be delayed for several seconds until a valve on the discharge line of the pump is completely opened so that high pressure cannot develop at the discharge of the pump. Another example would be to have a timer determine the amount of time a vessel takes to fill with fluid and display the time to the plant operator on a Graphic Display.

Timer Word Structure

Timer instructions have an area reserved for them in PLC memory. This memory area reserves one 16-bit word for each timer address. The Siemens Simatic S7 Instruction List language instruction set supports up to 256 timers. However, the exact number of timers supported depends on the model number of the CPU used in the application.

The IL Timer instruction in a Siemens S7 PLC uses bits 0 through 11 of accumulator 1 lower word to hold the time value. The programmer can load a time value into the lower word of accumulator 1 in binary, hexadecimal or binary coded decimal (BCD). The BCD format for the time value is shown in Figure 8-14. The left-most two bits (bits 14 and 15) in the timer word are irrelevant and ignored when the timer is started. Figure 8-14 shows the timer word loaded with a timer value of 349 with a time base of 1 second. The 12 bits in cell locations 0 through 11 are divided into 3 groups of 4 bits each to produce a 3-digit BCD number. Bits 12 and 13 hold the time base in binary code.

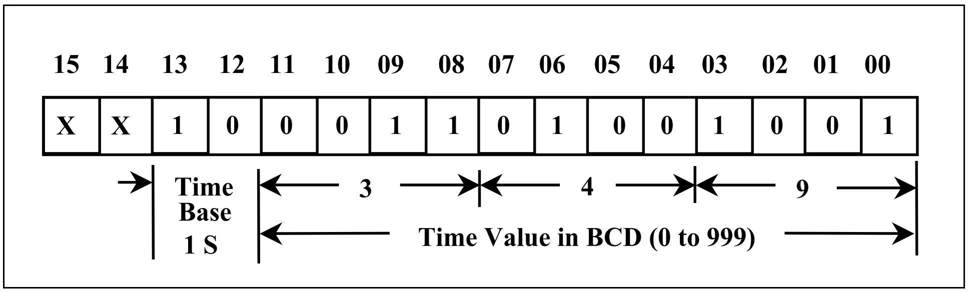


Figure 8-14. Timer Memory Word Structure in Accumulator 1

There are four time bases available: 10 ms, 100 ms, 1 s, and 10 s for Simatic S7 timers. Table 8-2 lists the time bases and their corresponding binary code. The time range is from 0 to 999 s.

Time Base	Binary Code for Time Base
10 ms	00
100 ms	01
1 s	10
10 s	11

Table 8-2. Time Base and Its Binary Code

Because time values are stored in memory with only one time interval, time values that are not exact multiples of a time interval are truncated. Time values that have too much resolution for the desired range are rounded down to achieve the desired range but not the desired resolution. Table 8-3 lists the possible resolutions and their corresponding time ranges.

Resolution	Range
0.01 s	10ms to 9s_990ms
0.1 s	100ms to 1m_39s_900ms
1 s	1s to 16m_39s
10 s	10s to 2hr_46m_30s

Table 8-3. Time Base Resolution and Ranges

The time base and the time value are loaded in an IL program using a Load (L) statement. Two syntax formats can be used to load a time value. The first format is L W#16#wxyz, where *w* is the time base and *xyz* is the time value in BCD. The other format is S5T#aaH_bbM_ccS_dddMS where "aa" is time value in hours, "bb" is a time value in minutes, "cc" is a time value in seconds and "ddd" is a time value in milliseconds. In this second method, the software selects the time base automatically and the value is rounded to the next lower number with that time base. For example, if we program a timer for 10 s, we would type in the IL instruction "L S5T#10S0MS" and the software would select a time base of 0.1 s according to Table 8-3. This time base resolution is selected because the time interval of 10 s is between the range of 100 ms and 1 minute, 39 seconds and 900 ms that give the lower or best resolution possible.

Programming of an IL Timer

To program for timer function in a Siemens S7 PLC using IL programming language, a minimum of three instructions must be used. First, an instruction to check the status of a timer start bit must be listed in the program (e.g., "A I124.1" as shown in the example program in Figure 8-15). Next, an instruction to load a time value must be included in the program (e.g., "L S5T#00H02M10S00S" as shown in Figure 8-15). Finally, an instruction for the type of timer to be used in the application must be listed in the program. There are five different types of timers available: Pulse timer (SP), Extended pulse timer (SE), On-delay timer (SD), Retentive on-delay timer (SS), and Off-delay timer (SF) in the IL language set.

In a IL program, a change in the logic value of the timer start bit prior to a start instruction statement starts a timer. A change in the start bit from 1 to 0 starts an off-delay timer (SF) and a logic change of 0 to 1 in the start bit starts any of the other timers. The programmed time and the start timer statements must follow the logic operation directly that provides the condition for starting the timer, as shown in Figure 8-15.

Because a timer runs down to zero from a set time, an IL program must provide the timer with a starting time. For example, a starting time of 1 minute and 10 s is used in the IL timer program shown in Figure 8-15.

IL Program	Description
A I124.1	Check the logic state of "Start" bit I124.1
L S5T#1M10S	Load starting time of 1 minute & 10 seconds
SP T1	Start timer T1 as a pulse timer

Figure 8-15. Example IL Timer Program

An IL timer is reset by using a Reset (R) instruction in the program. The CPU resets an IL timer when the result of a logic operation is 1 just before the R instruction in the program. Resetting a timer stops the current timing function and resets the time value of the timer to zero.

An Enable instruction (FR) can be used to restart the timing interval of an IL timer. A change in the state of the logic bit just before the FR instruction from logic 0 to logic 1 will restart a running timer. A timer enable is not required to start a timer, nor is it required for normal operation. An example of the use of an Enable instruction will be given in the Pulse timer to be discussed next.

IL Timer Application

An IL timer application program using a pulse timer is shown in Figure 8-16. The program uses enable, start and reset bit inputs to control the operation of a pulse timer.

When the Start bit shown in the timing diagram of Figure 8-16 changes from a logic state of 0 to logic of 1 at time t_0, the pulse timer will start. The programmed time then runs down to zero, at time t_1. At time t_2, the Start bit goes from logic 0 to logic 1, starting the timer. At time t_3, the Reset bit changes from logic 0 to 1 and the timer is reset to 0. The timer Start bit is returned to 0 at time t_4 , then at time t_5, the Start bit returns to 1, starting the timer. At time t_6, the Enable bit goes from logic 0 to logic 1 with the timer running, this restarts the timer. The timer runs until time t_7 with the time extended by the amount of the programmed time interval. At time t_8, the Enable bit again goes from logic 0 to logic 1 until time t_9, but there is no effect on the timer because the Start bit is off and the timer is not running.

IL Program	Description
A I124.0	Check the logic state of “Enable” bit I124.0
FR T1	Restart timer
A I124.1	Check the logic state of “Start” bit I124.1
L S5T#1M10S	Load starting time of 1 minute & 10 seconds
SP T1	Start timer T1 as a pulse timer
A I124.2	Reset timer bit, I124.2
R T1	Reset timer T1
A T1	
= Q124.0	Check the logic state of Timer T1

Timing Diagram

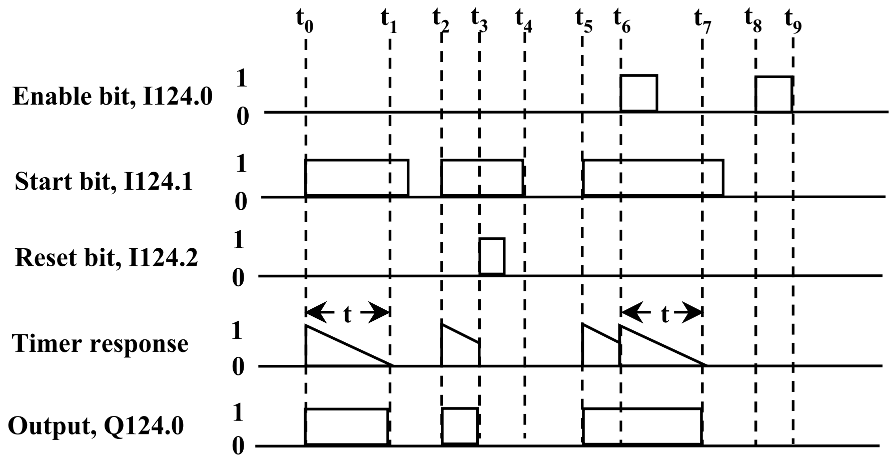


Figure 8-16. IL Pulse Timer Application

If an application requires that the timer output be delayed by a fixed amount of time, an On-delay (SD) timer instruction can be used. A typical application of an SD timer instruction is given in Example 8-3.

Counter Instructions

Counters are instructions that provide the same functions as hardware counters in process control applications. In some applications, they are used to activate or deactivate a control device after set count has been reached. For example, an IL control program might be used to count the number of parts produced on an assembly line and then stop the production line after a given number of parts have been manufactured.

EXAMPLE 8-3

Problem: Write an IL program for Siemens Simatic S7-300 PLC that delays the starting of a process pump for 10 seconds to allow a valve in the discharge line of the pump to fully open. Assume that the Pump starter relay is wired to PLC output point Q124.2 and a normally open switch connected to input point I124.0 is used by an operator to start the pump.

Solution: The IL program to operate the process pump is listed below with an explanation of each instruction.

IL Instruction	Explanation
A I124.0	Check logic of Pump Run input.
L S5T#10S	Load 10 second time delay if Pump Run bit is 1.
SD T2	Start timer T2 as an "On-delay" timer.
A T2	Check timer output status.
=Q124.2	Start process pump after 10 second delay.

There are two counter instructions in the IL programming language, Count Up (CU) and Count Down (CD). We will discuss an application using both types of counter instruction, but first we will cover the structure of the IL counter word.

Counter Word Structure

Counter instructions have a memory area reserved for them in a PLC. This memory area reserves one 16-bit word for each counter. The Siemens Simatic S7 Instruction List language instruction set supports up to 256 counters, but the exact number of counters supported depends on the model number of the CPU used in the application.

The IL counter word in a Siemens S7 PLC uses bits 0 through 9 to hold the count value in binary format. A Load (L) instruction can be used to read the binary count value out of a counter word and load the count value into the low word of accumulator. For example, the IL instruction, "L C2" will load accumulator 1 low directly with the count value of counter number 2 in binary format. A Transfer (T) instruction can be used to transfer the count value to another location in memory for use in a control application. An LC instruction can be used to read the binary count value out of a counter word and load the count value into the low word of accumulator 1 in binary coded decimal (BCD) format. For example, the IL instruction

"LC C3" will load accumulator 1 low directly with the count value in BCD format from counter number 3, as shown in Figure 8-17.

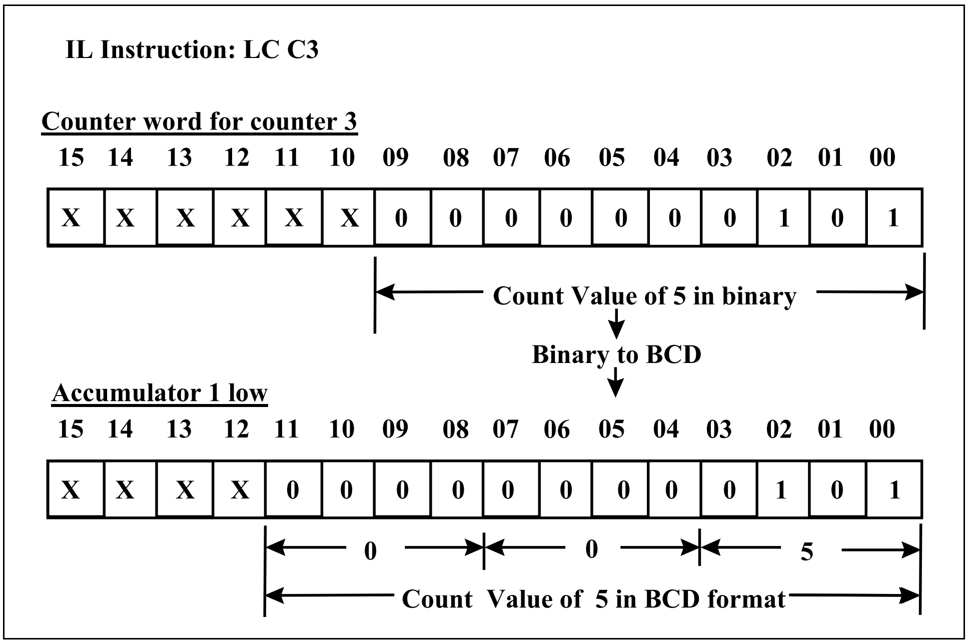


Figure 8-17. LC Instruction Used to Load Accumulator 1 with BCD Count Value

Counter Application

The IL programming application shown in Figure 8-18 provides an example of counting up, counting down, setting and resetting a counter, checking the logic state of a counter, and loading a fixed count value into a counter.

In the first line of the IL program (A I124.6), the logic state of the Count Up bit I124.6 is checked. If this bit is set to logic 1, the counter will be incremented by one count by the IL instruction, CU C1 in line 2. However, if the reset counter bit, I124.3 is one at the same time, the count value will not be incremented by 1. The third line of the program (A I124.7) examines the logic state of the Down Count bit, I124.7. If this bit is set to 1, the counter will be decremented by one count by the instruction DU C1 in line 4 of the program. The fifth line of the program (A I124.2) examines the logic state of the "Set Counter" bit I124.2. If this bit is 1, a count value of 3 is loaded into counter 1 by the instructions, L C#3 and S C1 in lines 6 and 7, respectively, in the program.

In the counter operation diagram shown at the bottom of Figure 8-18, the set counter bit is logic 1 and a count value of 3 is loaded into counter 1. Later, the down count bit is changed from 0 to 1 and then back to 0, three

IL	Program	Description
A	I124.6	Check the logic state of "Count Up" bit I124.6
CU	C1	Increment count value by 1
A	I124.7	Check the logic state of "Count Down" bit I124.7
CD	C1	Decrement count value by 1
A	I124.2	Check the logic state of "Set Counter" bit I124.2
L	C#3	Load count value of 3
S	C1	Set count value to 3 in counter number 1.
A	I124.3	Check logic state of "Reset Counter" bit I124.3
R	C1	Reset count value to 0
A	C1	Check the logic state of Counter C1
=	Q124.1	Output the logic state of Counter C1

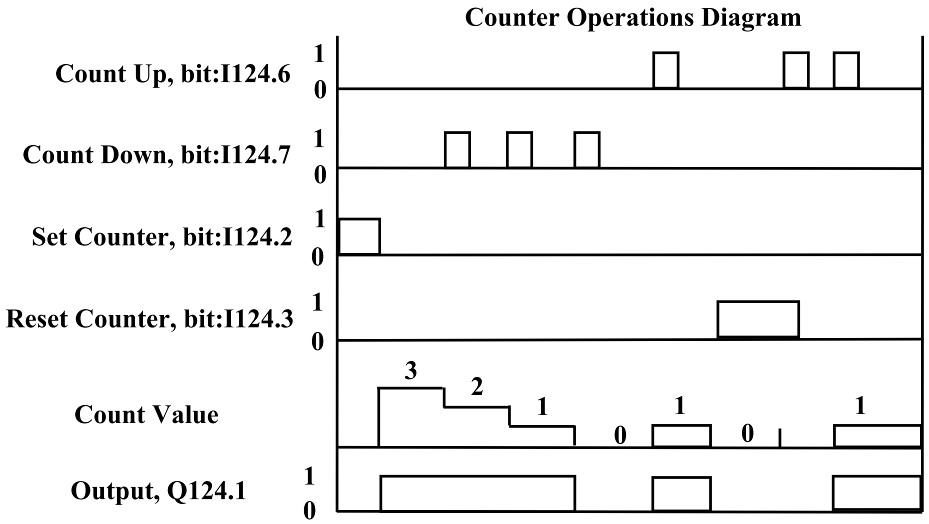


Figure 8-18. Up-Down Counter Programming Application

consecutive times. Each time, the count down bit goes from 0 to 1; the counter is decremented by one. This returns the count value to zero. Next, the count up bit is changed from 0 to 1 then back to 0, and the counter is incremented by one on the logic change of 0 to 1.

The last two IL instructions (A C1 and = Q124.0) are used to check the status of the count value in the counter. If the count value is zero, the output logic bit Q124.1 is set to 1. In all other cases, the bit is set to 0.

In the counter application shown in Figure 8-18, a second up count pulse occurred at the same time as the count reset bit changed to 1. In this case, the counter value is momentarily incremented by 1, but it is reset immediately because of the reset instruction that follows directly in the program. A pulse line in the counter operations diagram in Figure 8-18 indicates this momentary increase.

A typical control application using an IL counter instruction is given in Example 8-4.

EXAMPLE 8-4

Problem: Write an IL program for Siemens Simatic S7-300 PLC to turn off a conveyor belt on a production line after 150 parts have been produced. Assume the following: 1) Output bit Q124.3 = 0, turns off the conveyor belt; 2) Input bit I124.7 changes from 0 to 1 and then back to 0 each time a new part is produced; 3) A normally open (NO) pushbutton connected to input I124.2 is used to set the production count to 150, and 4) A NO pushbutton connected to input I124.3 is used to reset the counter to zero and stop the conveyor belt.

Solution: The IL program to control the operation of the conveyor belt is listed below with an explanation for each instruction.

IL Instruction	Explanation
A I124.7	Check "Part Produced" input for logic 1.
CD C2	Decrement count by 1, if a part is produced.
A I124.2	Test "Set part count" bit I124.2 for logic 1.
L C#150	Load count of 150.
S C2	Set count value to 150 for counter number 2.
A I124.3	Check logic state of "Reset Counter" input.
R C2	Reset count to 0, if I124.3 = 1.
A C2	Check logic state of counter C2.
= Q124.3	Run conveyor, if count value is not 0.

Integer Math Instructions

In this section, we will discuss IL instructions used to perform mathematical operations on positive and negative whole numbers 1, 2, 3, etc., or zero (i.e., integers). The basic IL math instructions used to add, subtract, multiply, and divide single word (16 bits) integers and double word (32 bits) integers are listed in Table 8-4.

IL Instruction	Description
+I	Add Single Word Integer
+D	Add Double Word Integer
-I	Subtract Single Word Integer
-D	Subtract Double Word Integer
*I	Multiply Single Word Integer
*D	Multiply Double Word Integer
/I	Divide Single Word Integer
/D	Divide Double Word Integer

Table 8-4. IL Integer Math Instructions

Integer Add Instructions

The "add" single integer (+I) instruction adds the contents of the low words of accumulators 1 and 2 and stores the result in the low word of accumulator 1. This operation overwrites the old contents of the low word of accumulator 1. The old contents of accumulator 2 and the high word of accumulator 1 remain unchanged as shown in Figure 8-19.

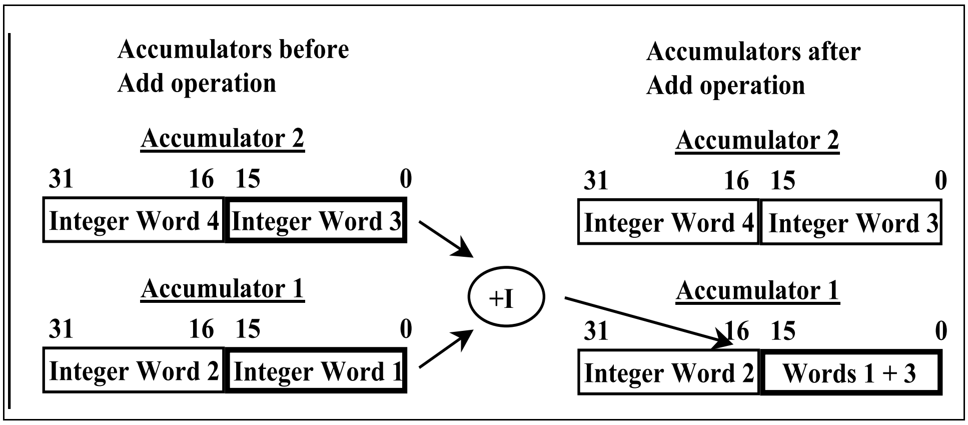


Figure 8-19. Accumulator Operation for an Integer Add Instruction

An example IL program using an Add Integer instruction is shown in Figure 8-20. The first instruction, "L MW10", loads the integer value in memory word MW10 into accumulator 1. The second instruction, "L MW12", loads the integer value in data word MW12 into accumulator 1. The CPU shifts the old contents of accumulator 1 into accumulator 2. The third instruction, "+I" tells the CPU to add the low words in accumulator 1 and 2 and store the result in the low word of accumulator 1. The last instruction, "T MW14", instructs the CPU to transfer the result of the addition that is in the low word of accumulator 1 into memory word DW14.

IL Instructions	**Description**
L MW10	Load the integer value in MW10 into accumulator 1
L MW12	Load the integer value in MW12 into accumulator 1
+I	Add single word MW12 to single word MW10
T MW14	Transfer integer result to data word MW14

Figure 8-20. Typical IL Program Using Integer Add Instruction

Constant single integers can be added to another integer by first loading the integer constant into the low word of accumulator 1. This moves the old value in the lower word of accumulator 1 into the low word of accumulator 2. Then an Add instruction can be used to add the low words of accumulators 1 and 2. For example, to add the whole number 5 to the value in accumulator 1, we use the instruction "L 5". This instruction, loads whole number 5 into the low word of accumulator 1. Then an add (+I) instruction can be performed. The result of this "add" operation is stored in accumulator 1 overwriting the old contents of the low word in accumulator 1. The content of accumulator 2 does not change after the add operation is performed.

Constant double word integers can be added to the contents of accumulator 1 by using the load instruction followed by an "add" double integer (+D) instruction. For example, to add the number 155 to the contents of accumulator 1, we first use the instruction "+L#155". The number 155 is loaded into accumulator 1 and the old contents of accumulator 1 are moved to accumulator 2. Then, an "add" double integer (+D) instruction would add the contents of accumulators 1 and 2 and store the result in accumulator 1. This operation overwrites the old contents of accumulator 1. The old contents of accumulator 2 remain unchanged.

Integer Subtract Instructions

The subtract single integer (-I) instruction subtracts the contents of the low word of accumulator 1 from the low word of accumulator 2 and stores the result in the low word of accumulator 1. This operation overwrites the old contents of the low word of accumulator 1. The old contents of accumulator 2 and the high word of accumulator 1 remain unchanged.

The subtract double integer (-D) instruction subtracts the contents of accumulator 1 from the contents of accumulator 2 and stores the result in accumulator 1. This operation overwrites the old contents of accumulator 1, but the contents of accumulator 2 remain the same.

Integer Multiply Instructions

The multiply single integer (*I) instruction multiplies the contents of the low word of accumulator 1 by the contents of the low word of accumulator 2 and stores the result in the low word of accumulator 1. This operation overwrites the old contents of the low word of accumulator 1. The old contents of accumulator 2 and the high word of accumulator 1 remain unchanged.

The multiply double integer (*D) instruction multiplies the contents of accumulator 1 by the contents of accumulator 2 and stores the result in accumulator 1. This operation overwrites the old contents of accumulator 1, but the contents of accumulator 2 remain the same.

The following example is a typical IL program using integer math instructions that might be encountered in a PLC control applications.

EXAMPLE 8-5

Problem: Write a Siemens Simatic S7 IL program to add the integer data in word MW22 to the integer data in word MW40 and then multiply the result by 5. Store the result in word MW50.

Solution: The IL program to perform the math operation is as follows:

IL Instruction	Explanation
L MW22	Load the integer value in MW22 into accumulator 1.
L MW40	Load the integer value in MW40 into accumulator 1.
+I	Add value in MW22 to value in MW40.
L 5	Load integer 5 into accumulator 1.
*I	Multiply result of addition (MW22 + MW40) by 5.
T MW50	Store result in MW50.

Integer Divide Instructions

The divide single integer (/I) instruction divides the contents of the low word of accumulator 2 by the contents of the low word of accumulator 1 and stores the result in the low word of accumulator 1. Any whole remainder is stored in the high word of accumulator 1. This operation overwrites the old contents of accumulator 1, but the contents of accumulator 2 remain unchanged.

The divide double integer (/D) instruction divides the contents of accumulator 2 by the contents of accumulator 1 and stores the result in accumulator 1. This operation overwrites the old contents of accumulator 1, but the contents of accumulator 2 remain the same.

Floating-Point Math Instructions

In this section, we will discuss the most common IL instructions used to perform mathematical operations on real numbers. The basic floating-point math instructions: add, subtract, multiply, and divide, are listed in Table 8-5. The Simatic reference-programming manual should be consulted for a complete listing of floating-point math instructions available in the S7 PLCs.

IL Instruction	Description
+R	Add 32-bit floating-point numbers
-R	Subtract 32-bit floating-point numbers
*R	Multiply 32-bit floating-point numbers
/R	Divide 32-bit floating-point numbers

Table 8-5. IL Real Math Instructions

Add Real Instruction

The add read (+R) instruction adds the 32-bit floating-point numbers in accumulators 1 and 2 and stores the result in accumulator 1. This operation overwrites the old contents of accumulator 1. The old contents of accumulator 2 remain unchanged as shown in Figure 8-21.

An example IL program using an “add real” instruction is shown in Figure 8-22. The first instruction, “L MD100”, loads the real number in memory double word MD100 into accumulator 1. The second instruction, “L MD104”, loads the real number in double data word MW104 into accumulator 1. The old contents of accumulator 1 are shifted into accumulator 2. The third instruction, “+R” tells the CPU to add the 32-bit floating-point numbers in accumulation 1 and 2 and store the result in accumulator 1. The last instruction, “T MD108”, instructs the CPU to transfer the result in accumulator 1 to double data word MD108.

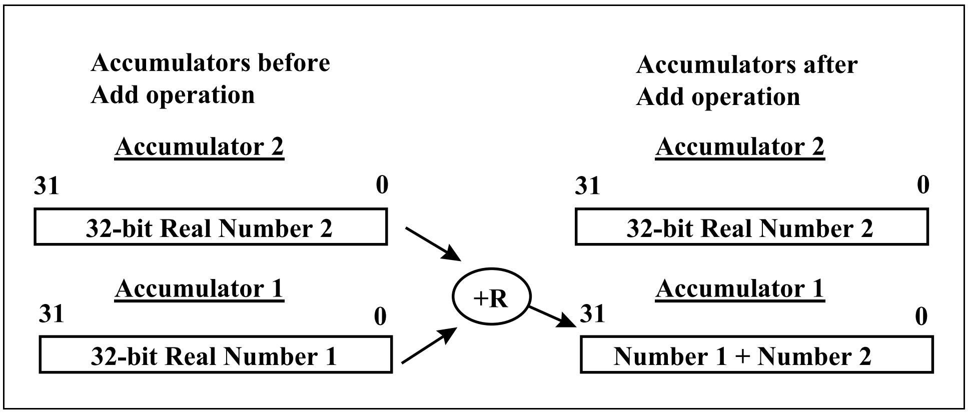


Figure 8-21. Accumulator Operation for an "Add Real" Instruction

IL Instruction	Description
L MD100	Load the real number in MD100 into accumulator 1
L MD104	Load the real number in MD104 into accumulator 1
+R	Add real number in MD100 to real number in MD104
T MD108	Transfer the result to double data word MD108

Figure 8-22. Typical IL Program Using an "Add Real" Instruction

Subtract Real Instruction

The subtract real (-R) instruction subtracts the contents of accumulator 1 from the contents of accumulator 2 and stores the 32-bit result in accumulator 1. This operation overwrites the old contents of accumulator 1. The old contents of accumulator 2 remain unchanged.

Multiply Real Instruction

The multiply real (*R) instruction multiplies the contents of accumulator 1 by the contents of accumulator 2 and stores the result in accumulator 1. This operation overwrites the old contents of accumulator 1. The old contents of accumulator 2 remain unchanged.

Divide Real Instructions

The divide real (/R) instruction divides the contents of accumulator 2 by the contents of accumulator 1 and stores the result in accumulator 1. This operation overwrites the old contents of accumulator 1, but the contents of accumulator 2 remain unchanged.

The following example is a typical IL program using integer math instructions that might be encountered in a PLC control applications.

EXAMPLE 8-6

Problem: Write a Siemens Simatic S7 IL program to add a 32-bit floating-point number in word MD50 to a 32-bit floating-point number in word MD54 and then divide the result by 100. Store the final result in word MD58.

Solution: The IL program to perform the math operation is as follows:

IL Instruction	Description of Instruction
L MD50	Load value in MD50 into accumulator 1.
L MD54	Load value in MD54 into accumulator 1.
	CPU transfers MD50 value to accumulator 2.
+R	Add MD50 to MD54.
L 1.000E+02	Load 100 into accumulator 1.
/R	Divide 100 into result of adding MD50 to MD54.
T MD58	Transfer result to MD58.

Comparison Instructions

In this section, we will discuss the three types Siemens S7 IL Language Comparison Instructions: compare single integer (I), compare double integer (D), and compare real (R). The type of comparisons made are listed Table 8-6 with the relational operator used in the IL instruction.

Compare Single Integer

The *Compare Single Integer (I)* instruction compares the two 16-bit whole numbers loaded in accumulators 1 and 2. This instruction compares the numeric value according to the type of comparison that is selected from the list of relational operators in Table 8-6.

If the comparison is true, the result of logic operation (RLO) bit in the status word is set to 1. If the comparison is false, the RLO bit in the status word is set to 0. The status of the RLO bit can be used in other parts of your IL program. The CPU executes the Compare instruction without regard to the result of a logic operation.

Types of Comparisons	Relational Operator
Accumulator 2 numeric value is equal to accumulator 1 numeric value	==
Accumulator 2 numeric value is not equal to accumulator 1 numeric value	<>
Accumulator 2 numeric value is greater than accumulator 1 numeric value	>
Accumulator 2 numeric value is less than accumulator 1 numeric value	<
Accumulator 2 numeric value is greater than or equal to accumulator 1 numeric value	>=
Accumulator 2 numeric value is less than or equal to accumulator 1 numeric value	<=

Table 8-6. Types of IL Program Language Comparisons

The IL program shown in Figure 8-23 uses three different types of integer compare instructions to illustrate their operation. The first instruction in the program, "L 5", loads the number 5 into accumulator 1. The second instruction, L MW10, loads the integer value in memory word MW10 into the low word accumulator 1 and the number 5 is shifted to the low word in accumulator 2. The third line in the program contains a compare equal ("==I") instruction. It compares the number 5 in the low word of accumulator 2 to the integer value in the low word of accumulator 1 to see if they are equal. If value in word MW10 is equal to 5, then the output bit Q124.0 is set to 1 by the assign instruction, "=Q124.0" in the fourth line of the program.

IL Instructions	Description
L 5	Load the number 5 into accumulator 1
L MW10	Load the integer value in MW10 into accumulator 1
= = I	Determine if the integer value of MW10 is equal to 5.
= Q124.0	Set Q124.0 to 1, if MW10 = 5
>I	Determine if MW10 is greater than 5
= Q124.1	Set Q124.1 to 1, if MW10 > 5
<I	Determine if MW10 is less than 5
= Q124.2	Set Q124.2 to 1, if MW10 < 5

Figure 8-23. Example IL Program Using Compare Single Integer Instructions

The fifth line contains a compare greater than (">I") instruction. It compares the number 5 in the low word of accumulator 2 to see if it is greater than the value in the low word of accumulator 1. If the number 5 is greater than the integer value in word MW10, then the output bit Q124.1 is set to 1 by the assign instruction, "=Q124.1" in the sixth line of the program. The seventh line contains a compare less than ("<I") instruction. It compares the number 5 in the low word of accumulator 2 to see if it is less than the value in the low word of accumulator 1. If the number 5 is less than the integer value in word MW10, then the output bit Q124.2 is set to 1 by the assign instruction, "=Q124.2" in the last line of the program.

Compare Double Integer

The *Compare Double Integer* instruction compares the two 32-bit whole numbers loaded in accumulators 1 and 2. The instruction compares the numeric value according to the type of comparison that is selected from the list of relational operators in Table 8-6.

If the comparison is true, the RLO bit in the status word is set to 1. If the comparison is false, the RLO bit in the status word is set to 0. The status of the RLO bit can be used in other parts of the IL program. The CPU executes the Compare instructions without regard to the result of a logic operation.

The IL program shown in Figure 8-24 uses three different double integer compare instructions to illustrate their operation. The first instruction in the program, "L MD2", loads the integer value in double word MD2 into accumulator 1. The second instruction, L MD10, loads the integer value in memory word MD10 into accumulator 1 and the old value is shifted into accumulator 2. The third line in the program contains a compare equal ("==D") instruction. It compares MD2 in accumulator 2 to the double integer value (MD10) in accumulator 1 to see if they are equal. If values are equal, then the output bit Q124.3 is set to 1 by the assign instruction, "=Q124.3" in the fourth line of the program.

The fifth line of the program contains a double integer compare greater than (">D") instruction. It compares the integer value from double word MD2 in accumulator 2 to see if it is greater than the integer value in accumulator 1 from double word MD10. If the integer value from double word MD2 is greater than the integer value from word MD10, then the output bit Q124.4 is set to logic 1 by the assign instruction, "=Q124.4" in the sixth line of the program. The seventh line contains a compare less than ("<D") instruction. It compares the integer value from double word MD2 in accumulator 2 to test to see if it is less than the integer value from word MD10 in accumulator 1. If the integer value in word MD2 is less

than the integer value in word MD10, then the output bit Q124.5 is set to true by the assign instruction, "=Q124.5" in the last line of the program.

IL Instructions	**Description**
L MD2	Load the value in double word MD2 into accumulator 1
L MD10	Load the integer value in MW10 into accumulator 1
= = D	Determine if the values in MD2 and MD10 are equal
= Q124.3	Set Q124.3 to 1, if MD2=MD10
>D	Determine if MD10 is greater than MD2
= Q124.4	Set Q124.4 to 1, if MD10 >MD2
<D	Determine if MD10 is less than MD2
= Q124.5	Set Q124.5 to 1, if MD10 < MD2

Figure 8-24. Example IL Program Using Compare Double Integer Instructions

Compare Real

The *Compare Real* instruction compares the two 32-bit real numbers loaded in accumulators 1 and 2. This instruction compares the numeric values according to the type of comparison that is selected from the list of relational operators in Table 8-6.

If the comparison is true, the RLO bit in the status word is set to 1. If the comparison is false, the RLO bit in the status word is set to 0. The status of the RLO bit can be used in other parts of your IL program. The CPU executes the Compare instruction without regard to the result of a logic operation.

The IL program shown in Figure 8-25 uses "less than" and "greater than" compare instructions to illustrate their operation on real numbers. The first instruction in the program, "L MD10", loads the value in double word MD10 into accumulator 1. The second instruction, L +1.75E01, loads the number 17.5 into the accumulator 1 and the MD10 value is shifted into accumulator 2. The third line in the program contains a compare greater than real (">R") instruction. It compares the value from double word MD10 in accumulator 2 to see if it is greater than the value of 17.5 in accumulator 1. If the number 17.5 is greater than the integer value from word MD10, then the output bit Q124.6 is set to logic 1 by the assign instruction, "=Q124.6" in the forth line of the program. The fifth line contains a compare less than real ("<R") instruction. It compares the value from double word MD10 in accumulator 2 to test to see if it is less than the

real number 17.5 in accumulator 1. If the integer value in word MD10 is less than 17.5, then the output bit Q124.7 is set to true by the assign instruction, "=Q124.7" in the last line of the program.

IL Instructions	Description
L MD10	Load the value in double word MD10 into accumulator 1
L 1.75E+01	Load the number 17.5 into accumulator 1
>R	Determine if the value in MD10 is greater than 17.5
= Q124.6	Set Q124.6 to true, if MD10 > 17.5
<R	Determine if value in MD10 is less than 17.5
= Q124.4	Set Q124.7 to true, if MD10 < 17.5

Figure 8-25. Example IL Program Using a Compare Real Instruction

EXERCISES

8.1 List the five PLC programming languages defined in the IEC 1131-3 standard.

8.2 What are the two text-based PLC languages and provide a brief description of each type?

8.3 What are the three graphical PLC languages and provide a brief description of each language?

8.4 Describe the "initial" step instruction in a Sequential Function Chart (SFC) program.

8.5 Explain the purpose of a "transition" instruction in a SFC program.

8.6 What is the purpose of the extra processor scan in a SFC program?

8.7 Determine if the following ST expressions are valid or invalid: a) 5*(6-4), b) (5 AND boo-var5), c) (boo_var2 OR boo_var2).

8.8 List the programming tips followed when using inactive separators to improve the readability of a Structured Text program.

8.9 What are the two basic structures of a IL instruction statement?

8.10 Write an IL program to turn on a pump starter connected to output point Q124.0, if control valve FV-1 is opened (input bit I124.0 = 1) and process tank level high (input bit I124.2 = 1).

8.11 List the five timer instructions available in the Siemens S7 IL programming instruction set.

8.12 Write an IL program for a Siemens Simatic S7-300 PLC to open a fill valve on a process tank to allow an ingredient to be added to the tank for 30 seconds. Assume that the fill valve is wired to PLC output point Q124.4 and a momentary normally open (NO) pushbutton connected to PLC input point I124.6 is used by an operator to open the fill valve. Use timer T3 and input bit I124.2 to reset the timer in the program.

8.13 Write an IL program for Siemens Simatic S7-300 PLC to turn off a conveyor belt on a production line after 10 parts have been produced. Assume the following: 1) Output bit Q124.5 = 0, turns off the conveyor belt; 2) Input I124.6 changes from 0 to 1 and then back to 0 each time a production part is rejected; 3) Input I124.7 changes from 0 to 1 and then back to 0 each time a new part is produced; 4) A NO pushbutton connected to input I124.2 is used to set the production count to 10, and 5) A NO pushbutton connected to input I124.3 is used to reset the counter to zero and to stop the conveyor belt. Use counter number 4 in the IL program.

8.14 Write a Siemens Simatic S7 IL program to add the integer in word MW60 to the integer data in word MW62 and then divide the result by 10. Store the result in word MW64.

8.15 Write a Siemens Simatic S7 IL program to subtract a 32-bit floating-point number in word MD70 from a 32-bit floating-point number in word MD74 and then divide the result by 10. Store the final result in word MD78.

BIBLIOGRAPHY

1. *Simatic S7 Statement List for S7-300 and S7-400 Programming Reference Manual*, Siemens AG, October 1998.
2. Lloyd, M., *Grafcet Graphical Functional Chart Programming for Programmable Controllers, Measurement & Control Magazine*, September 1987.
3. Christensen, J. H., *Programmable Controller Users and Makers to Go Global with IEC1131-3, I&CS Magazine*, October 1993.

9 Function Block Diagram Programming

Introduction

The Functional Block Diagram (FBD) is a graphical programming language that uses logic blocks similar to those used in Boolean algebra to represent basic logic. It also uses more complex function blocks to perform operations such as timing, counting, math, loading data, transferring data, and data comparison. The programmer is able to build complex control schemes by using functions from the FBD library and then interconnecting them in a graphical diagram area.

In this chapter, we will discuss the basics of FBD programming, and we will cover some typical applications. We will cover the FBD programming language used in Siemens S7-300 and S7-400 programmable controllers. Most of the other PLC manufacturers use a similar version of the standard FBD language. However, since we will only cover the basics of Siemens FBD programming, the Siemens reference manuals should be consulted for more detailed information on the use of their software.

Elements and Box Structure

FBD instructions consist of elements and boxes that are connected graphically to form networks. A FBD box describes a relationship or function between input and output variables. A function is described as a set of elementary function blocks as shown in Figure 9-1. Input and output variables are connected to blocks by connection lines.

An entire FBD program can be built with standard elementary function blocks from the FBD library, and the programmer can also develop custom function blocks. Each elementary function block has a given number of input connection points and a given number of output connection points. For example, the Boolean "AND" function block shown in Figure 9-1 has two inputs and only one output. The number of inputs can be increased to a higher value but the output is limited to one for this "AND" function box instruction. However, the output can be branched off to any number of other function blocks. The inputs are connected on its left border. The outputs are connected on its right border. An elementary box performs a single function between its inputs and its outputs. For example, the elementary function block shown in Figure 9-1 performs the Boolean "AND" operation on its two inputs and produces a result at the output. The name of the function to be performed by the block is written in its symbol. In the case of the "AND" function, the standard symbol "&" is used for the word "and" in the function block as shown in Figure 9-1.

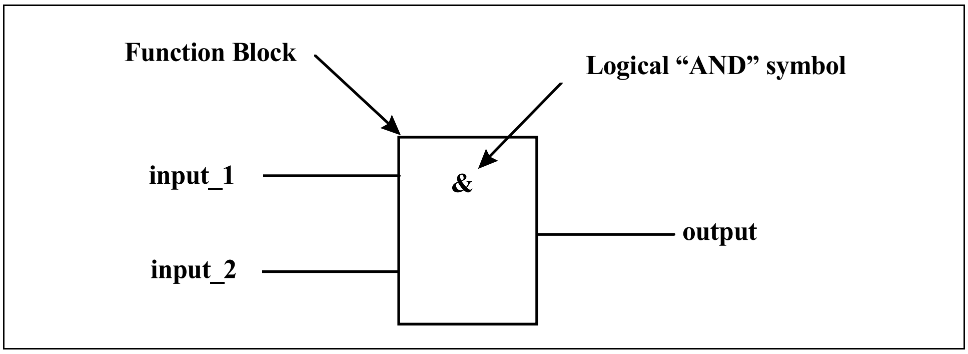


Figure 9-1. Typical Function Block Diagram Instruction

Elements and boxes that are connected graphically to form logic networks can be classified in the following types:

1. Instruction as elements
2. Instruction as a box with an address
3. Instruction as a box with an address and value
4. Instruction as a box with parameters

Table 9-1 provides examples of the four types of elements and boxes used in FBD programming.

Bit Logic Instructions

The most basic types of FBD instructions are the Boolean Bit Logic Instructions. These instructions perform logic operations on single bits in PLC memory. The basic Bit Logic instructions are: 1) "AND" (A), 2) "Or"

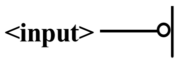
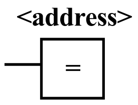
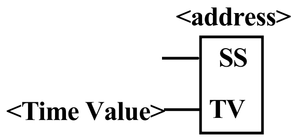
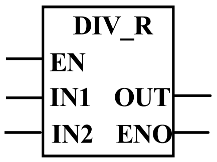
Type	Example	Element or Box
1. Instruction as elements	Negate binary input	
2. Instruction as a box with an address	Assign	
3. Instruction as a box with an address and value	Retentive on-delay timer	
4. Instruction as a box with parameters	Divide real	

Table 9-1. List of Typical FBD Instruction Types

(O), and 3) "Exclusive Or " (XOR). These instruction boxes check the logic state of an input bit address to establish whether the bit is activated "1" or not activated "0".

AND Logic Operation

In an "AND" logic operation, the logic state (1 or 0) of two or more specified input addresses is tested and if all inputs are 1, the output of the "AND" (&) function block is set to 1. If any input is 0, the output of the "AND" function block is set to 0. Figure 9-2 shows an example of a two input "AND" function block instruction with the output from the function box assigned to output bit location Q124.0. If the state of the two inputs, I124.0 and I124.1 are set to 1 then the bit in the assigned output Q124.0 is set to 1. If the "AND" instruction is the first instruction in a string of logic operations, it saves the result of its logic state test in the RLO bit of the status word of the processor.

The two input bits (I124 and I124.1) and the output bit are bit addresses for a Siemens Simatic programmable controller model number CPU314IFM. This is a compact PLC with 20 digital inputs, 16 digital outputs, 4 analog inputs, and 1 analog output. This compact PLC is designed for small control applications. Figure 9-3 is an input/output (I/O) wiring diagram for the Siemens CPU314IFM showing the 8 digital inputs with addresses

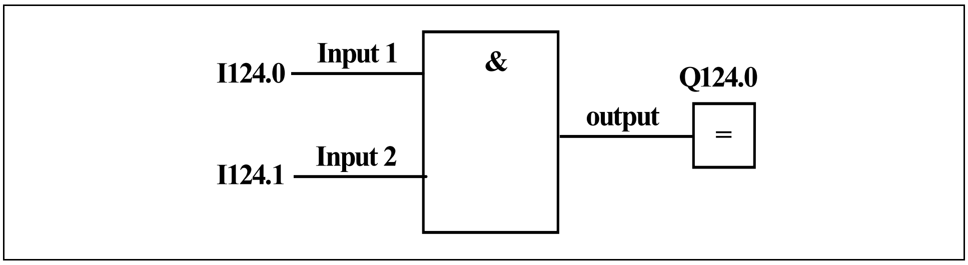


Figure 9-2. Example "AND" Function Block Instruction

of I124.0 through I124.7 and 8 digital outputs with addresses Q124.0 through Q124.7.

The wiring diagram also shows 8 test switches and pushbuttons connected to the 8 digital inputs. The 16 I/O points will be used in the example and application programs in this chapter and the input switches and pushbuttons can be used to test the control program. The 16 digital I/O points have light-emitting diode (LED) light to indicate the status of each I/O point. If the LED is on, the I/O bit is set to logic 1. If one of the normally opened (NO) hand switches (HS-1 though HS-4) is closed, +24 VDC (direct current) is applied to the input point, the associated logic bit (I124.0 though I124.3) will be set to 1, and the LED status light will be turned on.

Pushbuttons PB1 and PB2 are normally closed (NC), so +24 VDC is applied to inputs I124.4 and I124.5 when the pushbuttons are not depressed, the LED for inputs I124.4 and I124.5 will be turned on, and the logic bits will be set to 1. If the NC pushbuttons are pressed, the DC voltage is removed from the input points and the input bits will be set to 0.

The LEDs on the digital output points (Q124.0 through Q124.7) are turned on if the control program sets the associated bits to logic 1.

The PLC shown in Figure 9-3 with its test switches and pushbuttons can be used to test the operation of a new PLC program. The input switches and pushbuttons can be operated (opened or closed) and the LED lights on the output points can be monitored to verify that a control program is operating properly before the program is installed in an industrial application.

Or Logic Operation

In an "Or" logic operation, the logic state of two or more specified bit addresses is tested. If the logic state of one of the input bit addresses is 1, the output of the "Or" function block is 1. If the logic state of all the input bit addresses is 0, the output is set to 0. Figure 9-4 shows a typical example

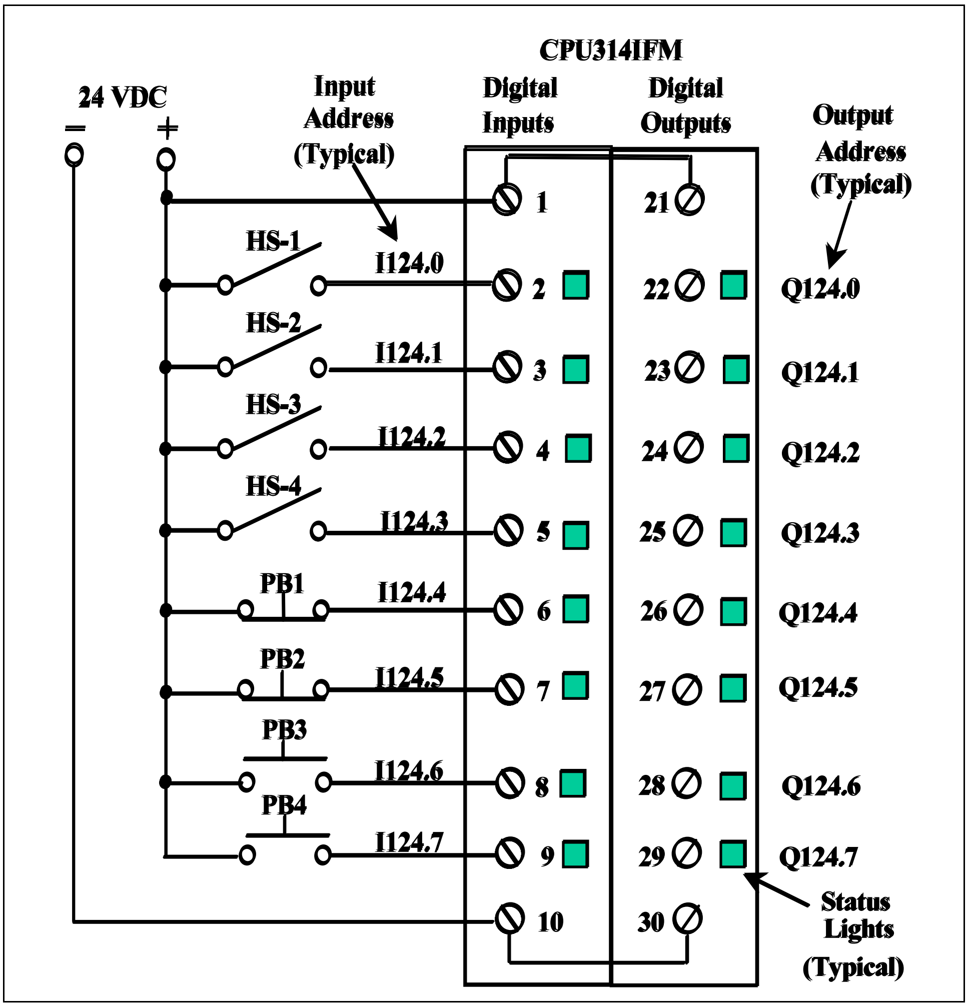


Figure 9-3. Wiring Diagram for the 16 digital I/O on a Siemens CPU314IFM

of an "Or" function block instruction with inputs at I124.2 and I124.3 and the output of the function box assigned to output bit address, Q124.1. The output is set to 1, if the logic state of input I124.2 or input I124.3 is set to 1.

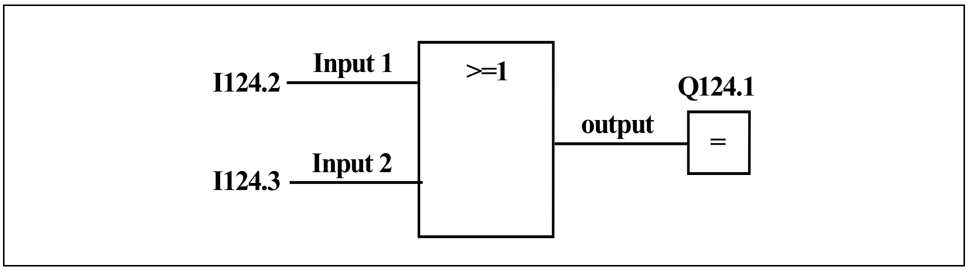


Figure 9-4. Example "Or" Function Block Instruction

An example application will help to illustrate FBD programming using "AND" and "Or" instructions.

EXAMPLE 9-1

Problem: Write a FBD program to start and stop a pump. In this application, the normally open (NO) contacts of a start pushbutton are wired to input bit address I124.6 and the normally closed (NC) contacts of a stop pushbutton are connected to input bit address I124.4 of a Siemens CPU314IFM (see Figure 9-3). The pump starter relay is connected to PLC output address Q124.0 and the auxiliary (aux.) motor start contacts (NO) are connected to PLC input bit address I124.0.

Solution: The application can be implemented using the program of Figure 9-5.

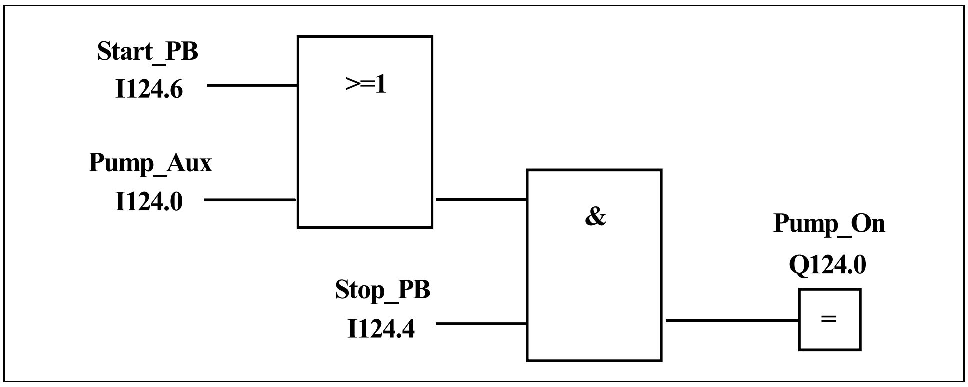


Figure 9-5. Answer for Example Problem 9-1

When the NO start pushbutton is depressed, input I:1/1 is true so the output of the "Or" function block instruction is 1 and since the NC stop pushbutton is not depressed, input I:1/0 is also true so that both inputs to the "AND" box are true. Since both inputs to the "AND" instruction block are 1, the output of the "AND" block is set to 1. This sets the bit address Q124.0 to 1. Since this output bit is wired to the pump start relay, the pump start relay is turned on causing the pump auxiliary (aux.) contacts to close. This, in turn, sets the Pump_Aux input bit I124 to 1 to keep the output of the "Or" box at 1 even after the start PB is released. If the stop PB is depressed, the Stop_PB bit, I124.4: is set to zero and Pump_On output bit is deenergized or set to 0 so the pump will be turned off and the auxiliary contacts on the pump starter will open and set the Pump_Aux input bit I124.0 to zero.

Exclusive Or Logic Operation

In an Exclusive Or logic operation, the logic state of two or more specified bit addresses is tested. If the logic state of one and only one of the input bit addresses is 1, the output of the "Or" function block is 1. If the logic state of all the input bit addresses is 0 or 1, the output is set to 0. Figure 9-6 shows a typical example of an exclusive Or function block instruction with inputs at I124.4 and I124.5 and the output of the function box assigned to output bit address, Q124.2. The output is set to 1 if the logic state of input I124.4 or input I124.5 is set to 1, but the output is set to 0 if both inputs are 1 or 0.

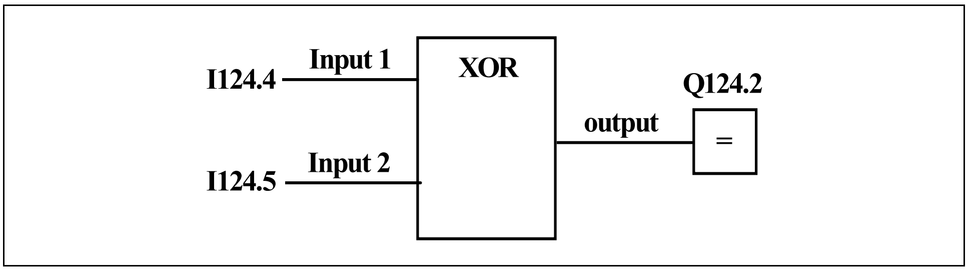


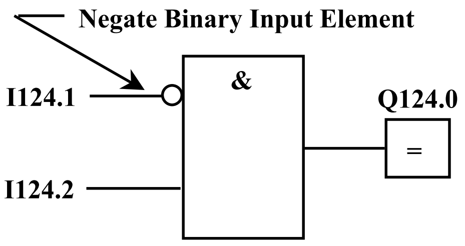
Figure 9-6. Example of an "Exclusive Or" Instruction

Negate Binary Input

The *Negate Binary Input* instruction is represented by a small circle at the input to a function box. This FBD element type instruction inverts or negates a bit present at the input to the element. For example in Figure 9-7, if the bit at input address I124.1 is logic 0, it is negated or inverted to a logic 1 by the negate element before it is operated on by the AND instruction. On the other hand, if the same input (I124.1) is logic 1, it is negated to a logic 0 before the AND function block uses its value. The truth table in Figure 9.7 shows the effect of the Negate Binary Input element on input bit I124.1 and its effect on the operation of the "AND" instruction. The first column of the Truth Table lists the three possible values for I124.1. The second column lists the negated value of I124.1. The third column lists the logic states of the second input bit, I124.2. The last column provides the "AND" logic result for the second and third columns.

Assign

The *Assign* instruction is represented by an equal sign (=) in a function box as shown in Figure 9-7. This FBD logic operation assigns the logic state of the bit at the input to the box to the address listed above the function box. For example in Figure 9-7, the result of the AND operation is assigned to the address Q124.0. The bit address types that can be used with the Assign



Truth Table

I124.1	$\overline{\text{I124.1}}$	I124.2	Q124.0
0	1	0	0
1	0	0	0
0	1	1	1
1	0	1	0

Figure 9-7. Example of a Negate Binary Input Element Instruction

instruction are I (input), Q (output), M (memory), D (data), and L (local data).

Set Output

The *Set Output* instruction is represented by the letter "S" in a function block. The Set Output instruction is only executed when the input to the box is logic 1. If the input bit is 1, this instruction sets the specified bit address to 1. If the input to the block is 0, the instruction does not affect the specified bit address and the logic state remains unchanged. The memory address types that can be used with the Set Output instruction are as follows: I (input), Q (output), M (memory), T (Timer), C (Counter), D (data), and L (local data) bit addresses.

Reset Output

The *Reset Output* instruction is represented by the letter "R" in a function block. The Reset Output instruction is only executed when the input to the box is logic 1. If the input bit is 1, this instruction resets the specified bit address to 0. If the input to the Reset instruction block is 0, the instruction does not affect the specified bit address and the logic state remains unchanged. The memory address types that can be used with the *Set Output* instruction are as follows: I (input), Q (output), M (memory), T (Timer), C (Counter), D (data), and L (local data) bit addresses.

Application Using Bit Instructions

To illustrate the use of the basic logic bit instructions, the control of the conveyor belt shown in Figure 9-8 will be discussed. In this application there are two sets of start and stop pushbuttons, one at the beginning of the conveyor and one at the end of the belt. These pushbuttons (PB-1 through PB-4) are wired to input points on a Siemens CPU314IFM programmable controller as shown in Figure 9-9. The two start buttons use NO contacts, and they are labeled PB-1 and PB-2. The two stop buttons use NC contacts, and they are labeled PB-2 and PB-4. The control program allows the operator to start and stop the conveyor belt from either end. There is a position detection switch (ZS-1) at the end of the conveyor to sense when a production part reaches the end of the conveyor. This input signal is used by the PLC control program to automatically stop the conveyor when a part reaches the end of the conveyor.

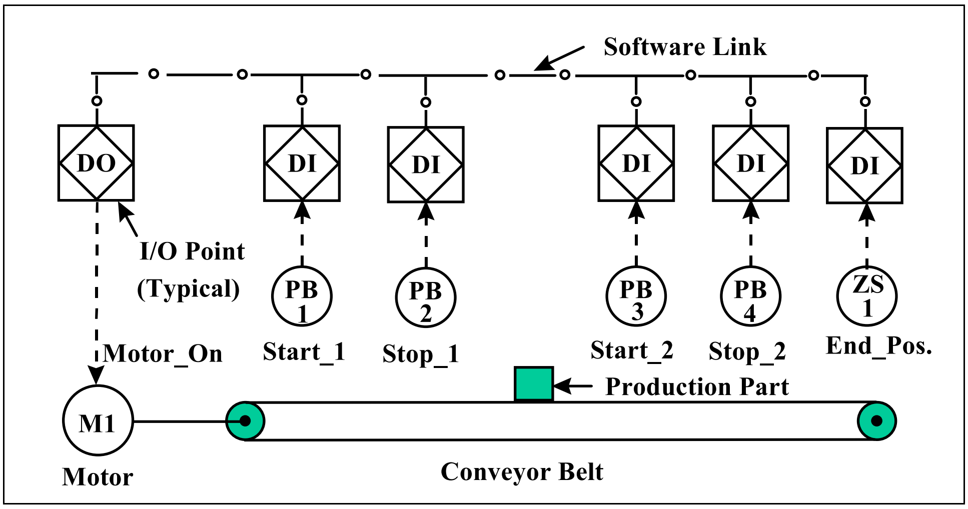


Figure 9-8. Mechanical Process Diagram for Conveyor Belt Application

The square box with a diamond inside is the ISA standard symbol used on process and mechanical diagrams to indicate a PLC input or output point. The dash line with circles between the dashes is the ISA standard symbol for a computer software link.

The function block program for the conveyor belt application is given in Figure 9-10. In this program the symbolic addresses for the I/O points are used instead of the absolute addresses. Table 9-2 provides the cross reference between the absolute I/O addresses and the symbolic addresses. Most PLC programming software packages provide for symbolic representation of absolute I/O to aid in programming and testing of control programs. It is much easier to understand and troubleshoot a control program that uses symbolic I/O memory addresses than a program that uses absolute addresses.

The first column of the table contains a list of the system instruments and components and the second column gives the absolute PLC I/O address for each component. The third column provides the symbol used to

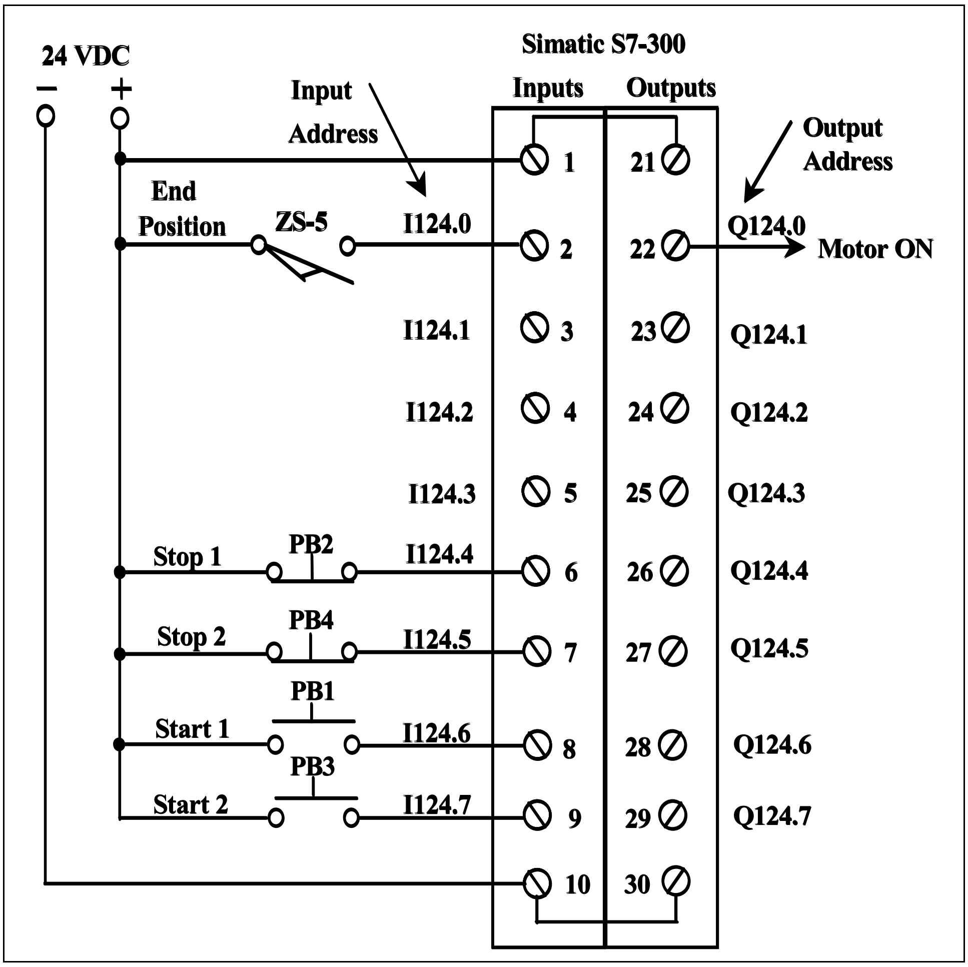


Figure 9-9. Input/Output Wiring Diagram for Conveyor Belt Application

replace the I/O address and the last column is the Symbol Table used by the PLC program to make the program easier to understand.

System Component	Absolute Address	Symbol	Symbol Table
Stop 1 NC PB	I124.4	Stop_1	I124.4 Stop_1
Stop 2 NC PB	I124.5	Stop_2	I124.5 Stop_2
Start 1 NO PB	I124.6	Start_1	I124.6 Start_1
Start 2 NO PB	I124.7	Start_2	I124.7 Start_2
Pos. Sw. ZS1	I124.0	End_Pos	I124.0 End_Pos
Motor Starter	Q124.0	Motor_On	Q124.0 Motor_On

Table 9-2. Elements for Symbolic Programming of a Conveyor Belt System

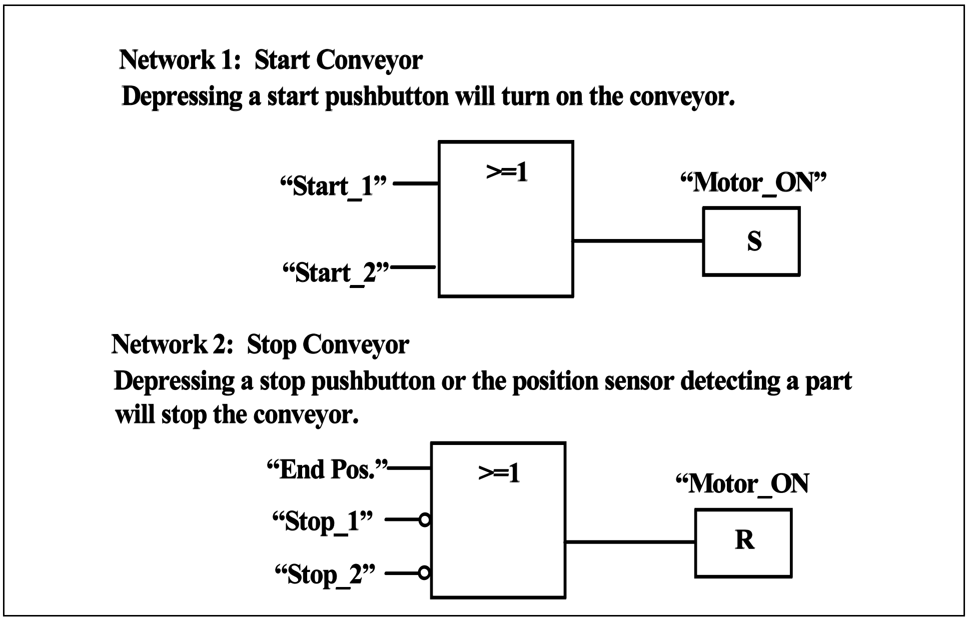

Network 1: Start Conveyor
Depressing a start pushbutton will turn on the conveyor.
"Start_1"
"Start_2"
>=1
"Motor_ON"
S
Network 2: Stop Conveyor
Depressing a stop pushbutton or the position sensor detecting a part will stop the conveyor.
"End Pos."
"Stop_1"
"Stop_2"
>=1
"Motor_ON
R

Figure 9-10. FBD Program for the Conveyor Belt Application

Timer Instructions

Software timers provide the same functions as hardware timers in process control applications. A typical application for a timer instruction is to delay an operation for a fixed time interval. For example, the starting of a pump might be delayed for several seconds until a valve on the discharge line of the pump is completely opened so that high pressure cannot develop on the discharge of the pump. Another example would be to have a timer determine the amount of time a vessel takes to fill with fluid and display the time to the plant operator on a graphics display.

Timer Memory Word Structure

Timer instructions in a Siemens S7 PLC require a 16-bit word in memory as shown in Figure 9-11. Bits 0 through 11 of the timer word contain the time value in binary coded decimal (BCD). Bits 12 and 13 of the timer contain the time base in binary code. The left-most two bits (bits 14 and 15) in the timer word are irrelevant and ignored when the timer is started. Figure 9-11 shows the timer word loaded with a timer value of 349 with a time base of 1 second. The 12 bits in cell locations 0 through 11 are divided into 3 groups of 4 bits each to produce a 3-digit BCD number.

Timer Function Block Configuration

The programmer selects a time value (TV) or timing interval for a timer by using the following configuration syntax: S5T#aaH_bbM_ccS_dddMS

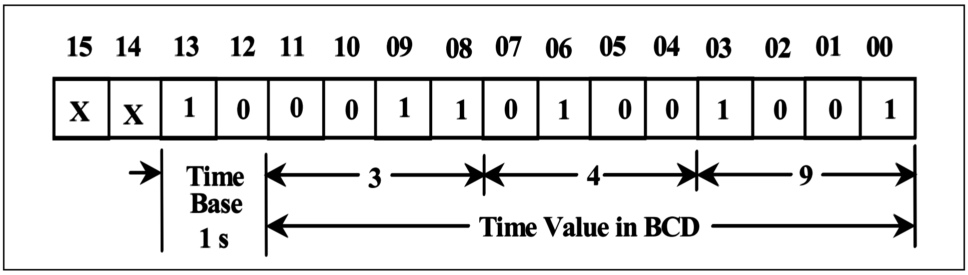


Figure 9-11. Timer Memory Word Structure

where "aa" is a time value in hours, "bb" is a time value in minutes, "cc" is a time value in seconds, and "ddd" is a time value in milliseconds. For example, to program a timer for 10 seconds, the programmer would type in "S5T#10S when configuring the time value input to the timer function block. If a time value of 1 hour and 30 minutes was required, the programmer would input "S5T#1H30M into the timer function block time value input during configuration.

There are four time bases available: 10 ms, 100 ms, 1 second, and 10 seconds. Table 9-3 lists the time bases and their corresponding binary code. The time base is selected automatically by the PLC when the function block is configured for a time value.

Time Base	Binary Code for Time Base
10 ms	00
100 ms	01
1 s	10
10 s	11

Table 9-3. Time Base and Its Binary Code

Pulse S5 Timer

The *Pulse S5 Timer* starts timing when there is a change in the logic input state from 0 to 1 at the start (S) input. A change in logic state is always required to start a Siemens S5 timer. The Pulse S5 Timer will continue to run for the time interval specified at the TV input until the programmed time elapses, as long as the logic state of the S input is 1 and the reset (R) is not set to logic 1 during the timing interval. If the S input changes from 1 to 0 before the timing interval is complete, the timer is stopped and the output Q goes to zero.

If the signal at the R input changes from 0 to 1 while a timer is running, the timer is stopped and the time is reset to zero. A logic state of 1 at the R input of the timer has no effect if the timer is not running. Figure 9-12 illustrates the operation of a Pulse S5 Timer instruction. In this application, input bit I124.1 is connected to the set input and input bit I124.6 is connected to the R input. The timing diagram shows the operation of the timer for various states of the R and S inputs. The timer output is assigned to output bit Q124.0 and the programmed time (*t*) interval is 10 seconds.

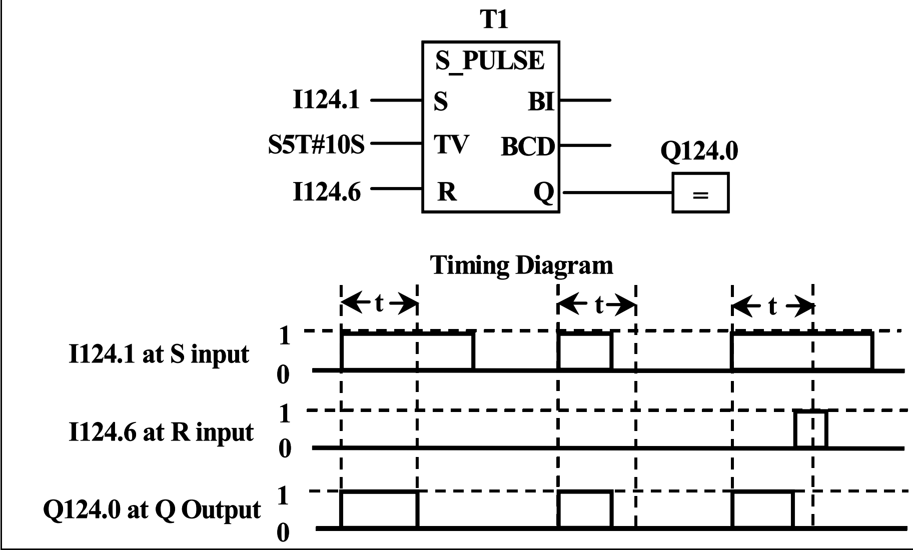


Figure 9-12. Pulse S5 Timer Application

The timer has two word outputs, BI and BCD, to indicate the time remaining. The time value in the BI output word is in binary format and the value in the BCD output word is in binary coded decimal format. The programmer configures the timer instruction for the memory word address for BI and BCD words if this timer information is needed in an application. In the example shown in Figure 9-12, the BI and BCD output words are not assigned to any particular memory address.

EXAMPLE 9-2

Problem: Write a FBD program to activate an ingredient feed pump on the inlet of a process tank for 5 seconds each time the batch process internal software bit B3/1 request the ingredient. In this application assume that the pump starter relay is connected to PLC output address Q124.1 and assume that internal bit B3/2 is used to stop the adding of the ingredient at any time.

Solution: The application can be implemented using the program of Figure 9-13.

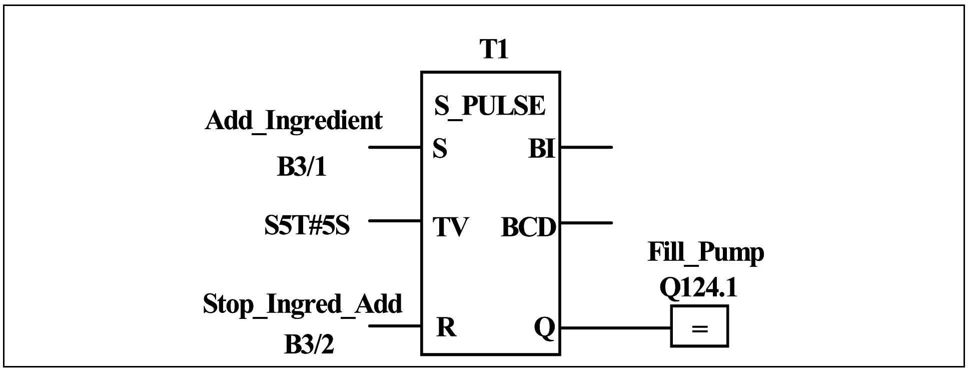


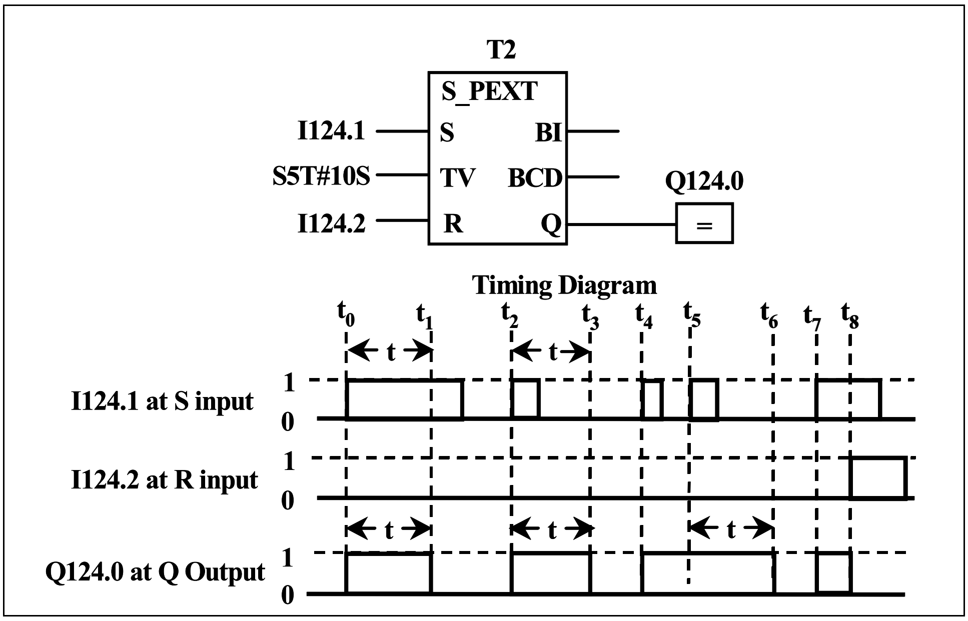
Figure 9-13. FBD Programming Answer for Example 9-2

Extended Pulse S5 Timer

The *Extended Pulse S5 Timer* starts timing when there is a change in the logic input state from 0 to 1 at the Start (S) input. A change in logic state at the S input is always required to start a Siemens S5 timer. The Extended Pulse S5 Timer will continue to run for the time interval specified at the time value (TV) word input until the programmed time elapses even if the logic state of the S input is not 1 and the reset (R) input is not changed from 0 to 1 during the timing interval. However, if the S input changes from 1 to 0 before the timing interval is complete, the timer will still continue to run to the end of the programmed time value.

If the signal at the R input changes from 0 to 1 while a timer is running, the timer is stopped and the time is reset to zero. A logic state of 1 at the R input of the timer will prevent the timer from starting on a 0 to 1 transition at the S input. The Extended Pulse timer has two word outputs, BI and BCD, to indicate the time remaining in a run cycle. The time value in the BI output word is in binary format and the value in the BCD output word is in binary coded decimal format. The programmer configures the timer instruction for the memory word address for BI and BCD words if this timer information is needed in an application.

Figure 9-14 illustrates the operation of an Extended Pulse S5 timer instruction. In this application, input bit I124.1 is connected to the set input and input bit I124.2 is connected to the R input. The programmed TV is 10 seconds (i.e., TV = S5T10s). The timing diagram shows the operation of the timer for various states of the R and S inputs. The timer output is assigned to output bit Q124.0 and the programmed TV of 10 seconds is represented by the letter *t* on the timing diagram. As can be seen in the timing diagram of Figure 9-14, the Extended Pulse timer is more complex than a Pulse timer. There are four different conditions shown on the timing diagram. In the first condition, the start input goes from 0 to 1 at time t_0 and remains at 1 for more than 10 seconds. The timer runs for the set time value of 10 seconds and the Q output is 1 for 10 seconds. In the second condition, the S input signal goes from logic 0 to logic 1 at time t_2, but the S input returns to 0 after a short time. However, the timer continues to run and the Q output is logic 1 while the timer is running, then after 10 seconds at time t_3, the timer stops and the Q output returns to logic 0. The third condition starts at t_4 when the S input goes to logic 1 for a few seconds and then returns to logic 0. The timer runs normally, but a few seconds later, a second S input signal is applied. The timer continues to run, but the time interval is *extended* by an additional 10 seconds until time t_6; hence the timer is called Extended Pulse. The final condition shown in Figure 9-14 is a standard timer reset operation. The timer is started at time t_7, but several seconds later at time t_8, the Reset input at I124.2 goes from 0 to 1 and the timer is reset and the Q output goes to 0.



9-14. Extended Pulse S5 Timer Application

On-Delay S5 Timer

The *On-Delay S5 Timer* instruction starts timing when there is a change in the logic input state from 0 to 1 at the Start (S) input. This type of timer counts down the time interval specified at the TV input as long as the S input is not logic 1. The Q output is changed from 0 to 1 only after the programmed time elapses. In other words, the output is turned on only after a *delay* in time interval programmed in the timer instruction; hence, the timer is named *On-Delay*.

If the signal at the Reset (R) input changes from 0 to 1 while a timer is running, the timer is stopped and the time is reset to zero. The On-Delay S5 Timer has two word outputs, BI and BCD, to indicate the time remaining in a timing interval. The time value in the BI output word is in binary format and the value in the BCD output word is in binary coded decimal format. The programmer configures the timer instruction for the memory word address for BI and BCD words if this timer information is needed in an application. In the application On-Delay S5 Timer shown in Figure 9-15, the BI and BCD outputs are assigned to memory word 10.

In this application, a normally open "Run" switch is wired to input address I124.3 on a Simatic S7 PLC. This input address is connected via programming to the set input of On-Delay timer T3. A normally closed pushbutton is wired to input address I124.4 on a Simatic S7 PLC. This input address point is connected via programming to the negated reset input of On-Delay timer. The timer is programmed for a 30-second time value (i.e., TV = S5T30s). The timing diagram shows the operation of the timer for various states of the R and S inputs. The timer output is assigned to output bit Q124.2 and the programmed TV of 30 seconds is represented by the letter *t* on the timing diagram.

There are three different conditions shown on the timing diagram. In the first condition, the "Run" switch is closed and the start input goes from 0 to 1 at time t_0 and remains at 1 for more than 30 seconds. The timer runs for the set time value of 30 seconds and then sets the Q output to 1 at time t_1. This output is connected to output Q124.2 on the PLC that in turn is wired to the pump start relay shown in Figure 9-15. The motor start relay is energized and remains on until time t_2, when the "Run" switch is opened and the start input to the Timer goes from 1 to 0. In the second condition, the S input signal goes from logic 0 to logic 1 at time t_3, but the S input returns to 0 before 30 seconds has elapsed at time t_4 and the Q output remains at 0.

The final condition shown in Figure 9-15, is a reset of the timer with the start input still at logic 1 and after the timer output has been activated. The timer is started at time t_5 and the output Q is activated after a delay of 30 seconds at time t_6. At time t_7, the Stop pushbutton is depressed so input I124.4 changes from 1 to 0 and the negated R input goes from 0 to 1 resetting the timer. The Q output goes to 0 and turns off the motor start relay.

Retentive On-Delay S5 Timer

The *Retentive On-Delay S5 Timer* instruction starts timing when there is a change in the logic input state from 0 to 1 at the Start (S) input. This timer continues to run for the time specified at the Time Value (TV) input even if

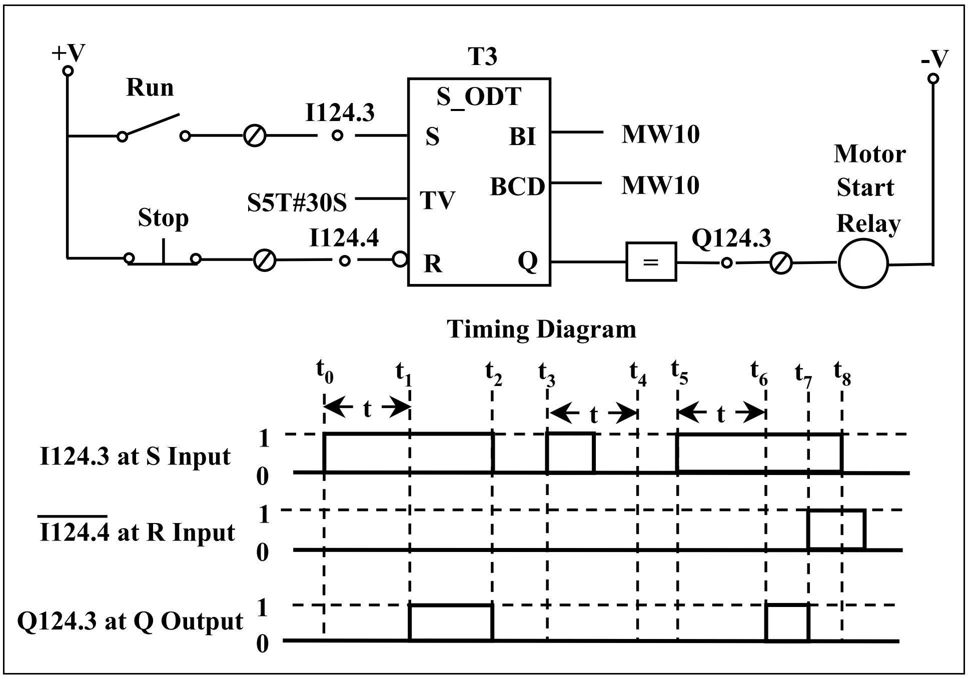


Figure 9-15. On-Delay S5 Timer Application

EXAMPLE 9-3

Problem: Write a FBD program to turn on a process pump, 2 seconds after the outlet valve on the pump as been opened by PLC output Q124.3. In this application, assume that the pump starter relay is connected to PLC output address Q124.4 and assume that internal bit B3/4 is used to stop the pump.

Solution: The application can be implemented using the FBD program of Figure 9-16.

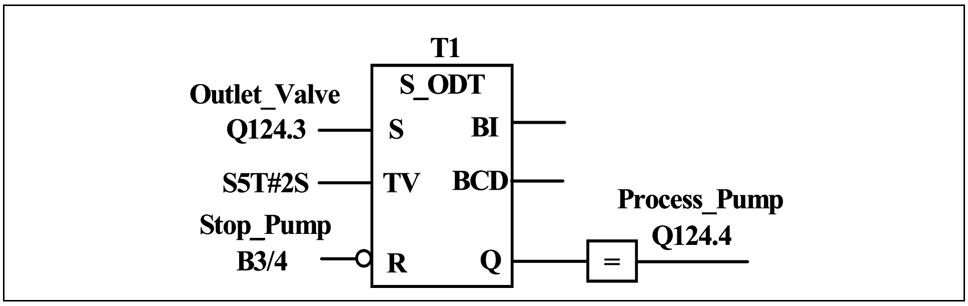


Figure 9-16. FBD Program for Example 9-3

the signal state at input S changes 1 to 0 before the time has expired. In other words, the Q output is retained at 1 after the programmed time elapses even if the S input is not kept positive.

If the signal at the Reset (R) input changes from 0 to 1 while a timer is running, the timer is reset to zero. The *Retentive On-Delay S5 Timer* has two word outputs, BI and BCD, to indicate the time remaining in a timing interval. The time value in the BI output word is in binary format and the value in the BCD output word is in binary coded decimal format. The programmer configures the timer instruction for the memory word address for BI and BCD words if this timer information is needed in another part of an application program. In the Retentive On-Delay S5 Timer instruction shown in Figure 9-17, the BI and BCD outputs are assigned to memory word 14.

In this application, a NO "start" switch is wired to input address I124.0 on a Simatic S7 PLC. This input address is connected via internal software to the set input of timer instruction. A NO "reset" switch is wired to input address I124.1. This input address point is connected via internal software to the reset input of timer instruction. The timer is programmed for a 30-second time value (i.e., TV = S5T30s).

The timing diagram shows the operation of the timer for various states of the R and S inputs. The timer output is assigned to output bit Q124.2 and the programmed TV of 30 seconds is represented by the letter *t* on the timing diagram.

There are three different operations shown on the timing diagram. In the first operation, the "start" switch is closed and the start input goes from 0 to 1 at time t_0 and remains at 1 for more than 30 seconds. The timer runs for the set time value of 30 seconds and then sets the Q output to 1 at time t_1. This output is connected to output Q124.2 on the PLC that in turn is wired to a relay as shown in Figure 9-17. The relay is energized and remains on until time t_3 when the "reset" switch is closed by an operator, the reset input to the Timer instruction goes from 0 to 1, and the timer Q output is set to 0.

In the second condition, the S input signal goes from logic 0 to logic 1 at time t_4 with the R input set to logic 0. Before the time interval has reached 30 seconds, the S input signal is returned to 0, but the timer continues to run until time t_5 when the timer output Q is set to 1. At time t_6, the timer is reset and the output Q is deactivated.

In the final operation, at time t_7, the S input is changed from 0 to 1 and the timer instruction starts. However, before the programmed time has elapsed, the S input is returned to 0 and then back to 1. This restarts the timer at time t_8, the timer runs until time t_{10}, and then set the Q output to 1. It will remain at 1 until the timer is reset.

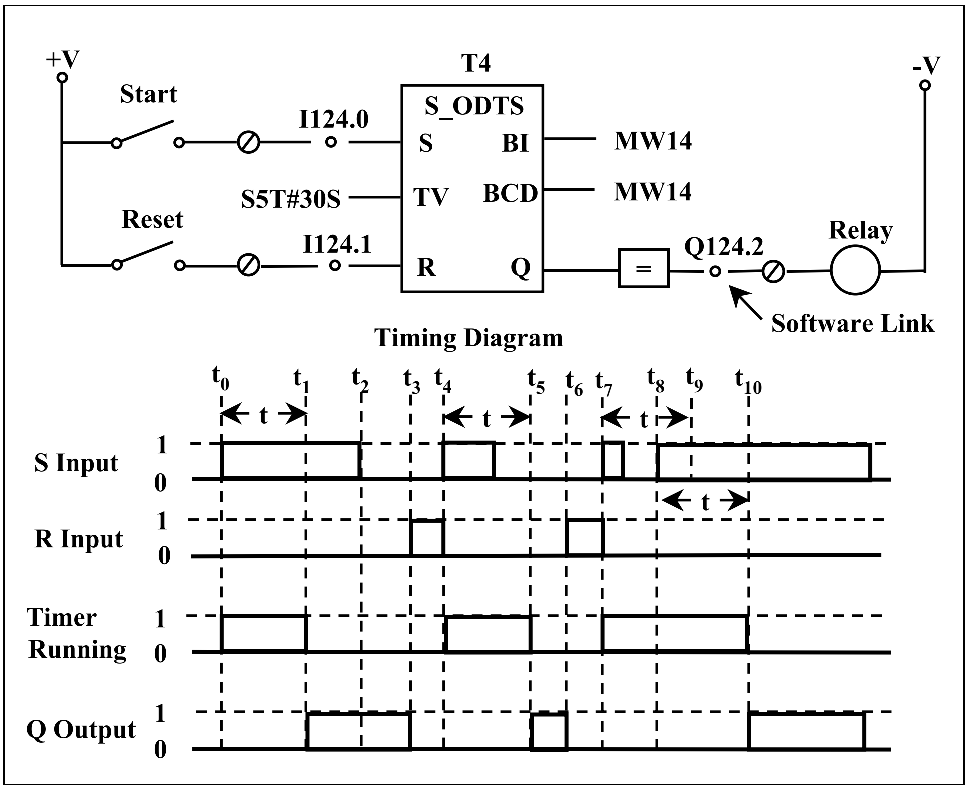


Figure 9-17. Retentive On-Delay S5 Timer Application

Off-Delay S5 Timer

The *Off-Delay S5 Timer* instruction starts timing when there is a falling edge (change in signal state from 1 to 0) at the Start (S) input. The output Q is logic 1 when the logic state of the S input is 1 or when the timer is running.

If the signal at the Reset (R) input changes from 0 to 1, the timer is reset to zero and stops running. The Off-Delay S5 Timer has two word outputs, BI and BCD, to indicate the time remaining in a timing interval. The time value in the BI output word is in binary format and the value in the BCD output word is in binary coded decimal format. In the Off-Delay S5 Timer instruction application shown in Figure 9-18, the BI and BCD outputs are assigned to memory word 10.

In this application, a normally open "start" switch is wired to input address I124.0 on a Simatic S7 PLC. This input address is connected via internal software to the set input of timer instruction. A NO "reset" switch is wired to input address I124.1. This input address point is connected via internal software to the reset input of timer instruction. The timer is programmed for a 30-second time value (i.e., TV = S5T30s).

The timing diagram shows the operation of the timer for various states of the R and S inputs. The timer output is assigned to output bit Q124.2 and the programmed time value of 30 seconds is represented by the letter *t* on the timing diagram.

There are three different operations shown on the timing diagram. In the first operation, the "start" switch is closed and the S input goes from 0 to 1 at time t_0, the start switch is opened at t_1 and the timer runs for 30 seconds until time t_2. The Q output is set to 1 from time t_o to t_2. The output of the timer is connected to output bit Q124.2 on the PLC that in turn is wired to a relay as shown in Figure 9-18.

In the second condition, the S input signal is changed from 0 to 1 and 1 to 0 several times. Starting at time t_3 when the "start" switch is closed, changing the S input signal from 0 to 1. This sets output Q to 1. At time t_4, the S signal goes from logic 1 to logic 0 starting the timer so the output remains at 1. Before the time interval has reached 30 seconds, the S signal is again returned to 1 at time t_5 so the timer stops but the timer output Q remains set to 1. At time t_7, the start input goes from 1 to 0 and the timer runs again keeping the output Q active until time t_8, when the timer completes its programmed time interval.

In the final operation, the timer is reset during a timing operation as follows: at time t_9, the start input is changed from 0 to 1 and the timer output Q is activated. At time t_{10}, the S input is changed from 1 to 0 and the timer instruction starts. However, before the programmed time interval has elapsed, the reset input is changed from 0 to 1. This stops the timer at time t_{11} and sets the Q output to 0. It will remain at 0 until the timer is started or the S input is set to 1.

Counter Instructions

Software counters provide the same types of functions as hardware counters used in process control applications—they are used to activate or deactivate a device after a selected count value has been reached. A typical application for a counter instruction is to stop a production assembly operation after a fixed number of parts have been produced. For example,

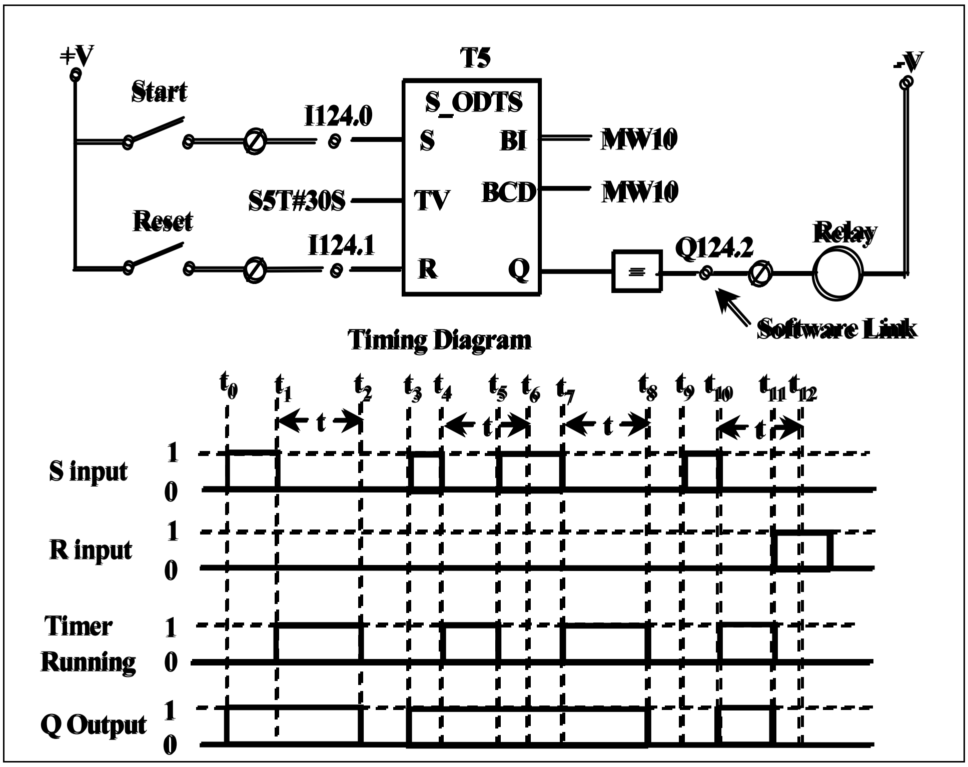


Figure 9-18. Off-Delay S5 Timer Application

a conveyor line might be stopped after a given number of parts have passed a certain point on the conveyor belt.

Counter Memory Word Structure

Counter instructions in a Siemens S7 PLC require a 16-bit word in memory as shown in Figure 9-19. Bits 0 through 9 of the Siemens counter word contain the count value in binary code as shown in the bottom of Figure 9-19. The top memory word in Figure 9-19 shows the counter word loaded with a counter value of 349. The 12 bits in the top cell locations 0 through 11 are divided into 3 groups of 4 bits each to produce a 3-digit BCD number for the counter.

Counter Function Block Configuration

The programmer selects a preset value (PV) for a counter by using the configuration syntax: C#ccc, where "ccc" is the count value in BCD format. For example, to program a counter for PV of 130 counts, the programmer would type in "C#130 when configuring the PV input to a counter function block instruction. The range of the preset count value is 0 to 999. The programmer can be configured to increment or decrement the count

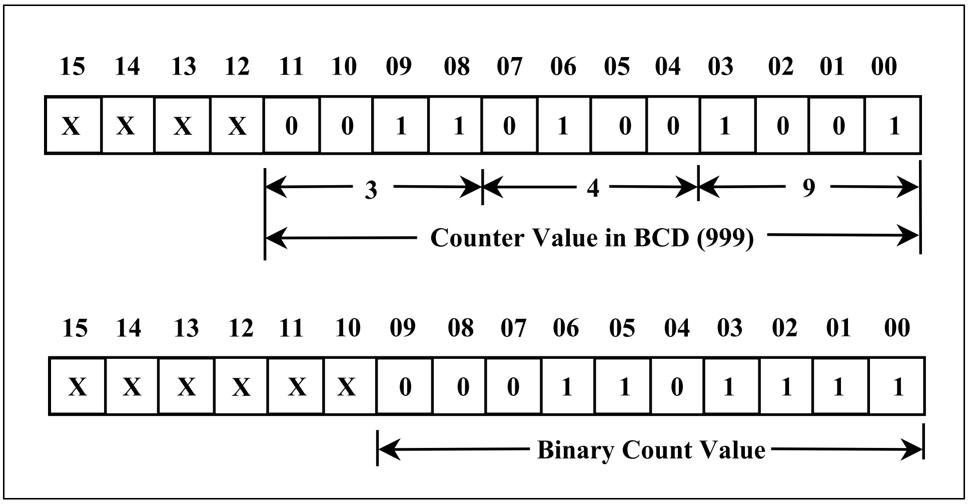


Figure 9-19. Counter Memory Word Structure

value (CV) within this range using the Up-Down Counter, Up Counter, and Down Counter instructions.

Up-Down Counter

The *Up-Down Counter* instruction counts when there is a rising edge change in the logic input state from 0 to 1 at the Count Up (CU) input or at the Count Down (CD) input. The CV is incremented by 1 if the CU input changes from 0 to 1 (rising edge) and the value of the counter is less than 999. The counter is decremented by 1 if the CD input changes from 0 to 1 and the value of the counter is greater than 0. If there is a rising edge signal at both count inputs, both operations are executed and the count remains the same. The counter is reset to a CV of 0 if there is a 0 to 1 transition at the Reset (R) input.

A typical application is shown in Figure 9-20 to illustrate the operation of Up-Down counter instructions. There are four NO switches connected to four input points I124.0 through I124 on a Siemens S7-300 PLC to test the operation of the Up/Down Counter. If the "Set" switch connected to input I124.2 is closed, the signal at the S input to the counter instruction is changed from 0 to 1. This operation sets the CV to 44. If the "Up" switch connected to input point I124.0 is closed, the signal at the CU input to the counter instruction is changed from 0 to 1 and the value of counter C10 is incremented by 1, except when the value of the counter is already 999. If the "Down" switch connected to input point I124.1 is closed, the signal at the CD input to the counter instruction is changed from 0 to 1 and the value of counter C10 is decremented by 1, except when the value of the counter is already 0.

The counter status bit Q is assigned to Q124.7 in the application shown in Figure 9-20. This status bit is 1, when the CV is not equal to 0.

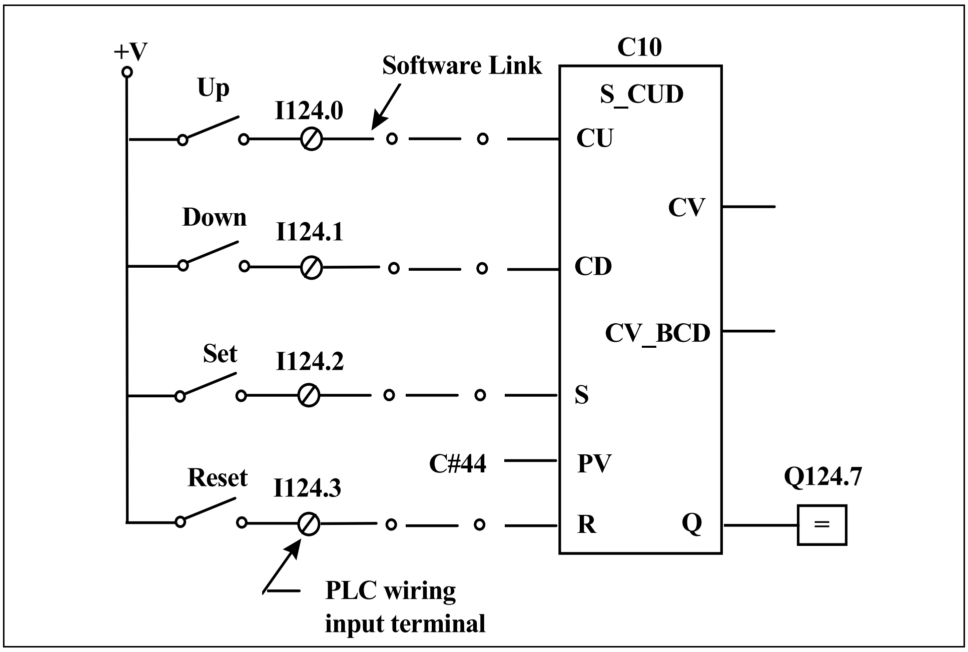


Figure 9-20. Typical Up-Down Counter Application

Up Counter

The *Up Counter* instruction counts when there is a rising edge change in the logic input state from 0 to 1 at the Count Up (CU) input. The counter value (CV) is incremented by 1, if the CU input changes from 0 to 1 (rising edge) and the value of the counter is less than 999. The counter is reset to a CV of 0 if there is a 0 to 1 transition at the Reset (R) input.

A typical application is shown in Figure 9-21 to illustrate the operation of an Up Counter instruction. There is a production part detection switch connected to input point I124.0 on a Siemens S7-300 PLC to count the number of parts produced in a given production run. There is a "Start" switch connected to input I124.2. If this "Start" switch is closed, the signal at the S input to the counter instruction is changed from 0 to 1. This operation sets the CV to 900. If the switch connected to input point I124.0 is closed, the signal at the CU input to the counter instruction is changed from 0 to 1 and the value of counter C11 is incremented by 1, except when the value of the counter is already 999.

The counter status bit Q is assigned to Q124.7 in the application shown in Figure 9-21. This output bit is 1, when the counter value is not equal to 0.

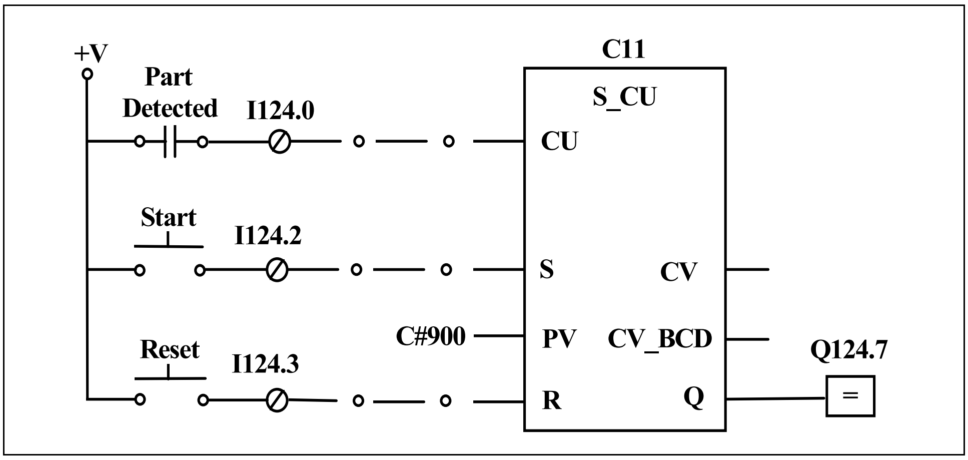


Figure 9-21. Typical Up Counter Application

The counter can be reset to 0 by depressing the "Reset" switch connected to input I124.3.

Down Counter

The *Down Counter* instruction counts down when there is a rising edge change in the logic input state from 0 to 1 at the CD (Count Down) input. The counter value (CV) is decremented by 1, if the CD input changes from 0 to 1 (rising edge) and the value of the counter is greater than 0. The counter is reset to a CV of 0 if there is a 0 to 1 transition at the Reset (R) input.

A typical application is shown in Figure 9-22 to illustrate the operation of a Down Counter instruction. A production part detection switch is connected to input point I124.0 on a Siemens S7 PLC to count the number of parts produced in a given production run. A "Start" switch is connected to input I124.2. When this "Start" switch is closed, the signal at the S input to the Down Counter instruction is changed from 0 to 1. This operation sets the CV to 100. If the switch connected to input point I124.0 is closed, the signal at the CD input to the counter instruction is changed from 0 to 1 and the value of counter C12 is decremented by 1, except when the value of the counter is already 0.

The negated value of the counter output Q bit assigned to output bit Q124.7 and this output bit is connected to a Stop Conveyor relay as shown in Figure 9-22. The Q status bit is 1 when the CV is not equal to 0, so its negated bit will be 1 when the CV is 0.

The purpose of the application is to count parts produced on a conveyor line and then stop the conveyor after 100 parts have been produced.

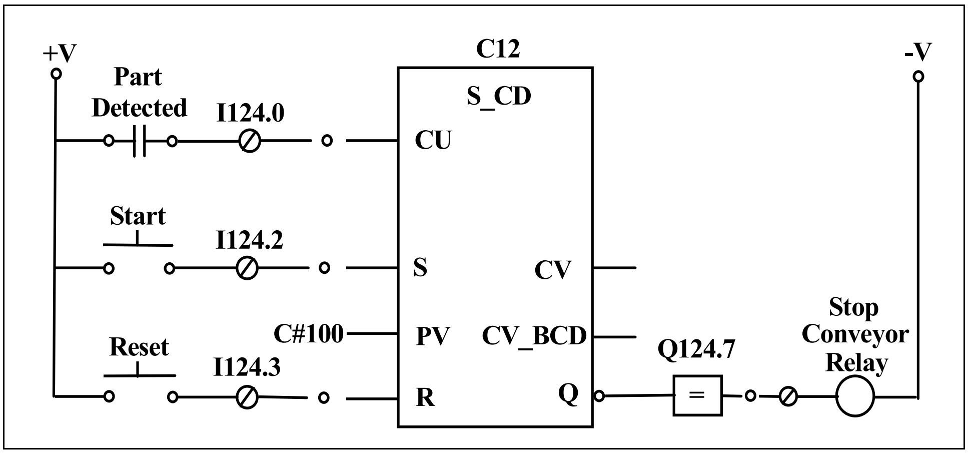


Figure 9-22. Typical Down Counter Application

The counter can be reset to 0 by depressing the Reset switch connected to input I124.3.

EXAMPLE 9-4

Problem: Write a FBD program to count the number of parts rejected during a production operation and activate an alarm beacon if the number of rejected parts reaches 20 parts, during any production run. Assume that the part rejection signal is connected to input I124.1, a production line running signal is given by internal bit B3/2, a counter reset pushbutton is connected to input I124.3 and the alarm beacon is connected to output Q124.4.

Solution: The application can be implemented using the FBD program of Figure 9-23.

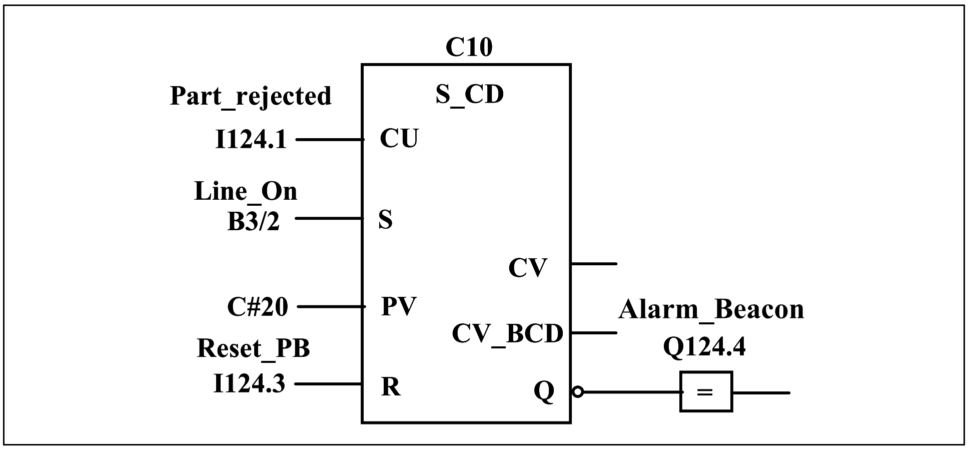


Figure 9-23. FBD Program for Example 9-4

Integer Math Instructions

In this section, we will discuss FBD instructions used to perform mathematical operations on positive and negative whole numbers 1, 2, 3, etc., or zero. The basic math instructions add, subtract, multiply, and divide for single and double integer words are listed in Table 9-4.

The integer math FBD instruction for an Add operation is shown in Figure 9-24. All the other integer math instructions have the same input and output functions and format. A signal state of 1 at the Enable (EN) input activates the math instruction. The instruction performs the math operation on the integer values at IN1 (input 1) and IN2 (input 2). If the result is outside the permissible range for an integer, the OV and OS bits of the status word are set to 1 and the Enable Output (ENO) bit of the instruction is set to 0. The result of the math operation is stored in the memory location programmed for the output (OUT) of the instruction.

Instruction Symbol	Description
ADD_I	Add Integer
ADD_DI	Add Double Integer
SUB_I	Subtract Integer
SUB_DI	Subtract Double Integer
MUL_I	Multiply Integer
MUL_DI	Multiply Double Integer
DIV_I	Divide Integer
DIV_DI	Divide Double Integer

Table 9-4. Integer Math Instructions

In the Add Integer instruction shown in Figure 9-24, a logic bit of 1 at input I124.0 activates the ADD_I instruction. The result of the addition of word MW0 and word MW2 is entered into memory word MW10. If the result is outside the permitted range for an integer or the logic state of the EN input I124.0 is 0, the ENO will be set to 0. However, if the result of the

Add operation is valid and the EN input bit I124.0 is set to 1, the output bit Q124.7 will be set to 1.

EXAMPLE 9-5

Problem: Write a Siemens S7 FBD program to add the integer data in word MW5 to the integer data in word MW7 and then multiply the result by 100, if the input enable bit, I124.0 is true. Store the result in word MW100.

Solution: The FBD program to perform the math is shown in Figure 9-25.

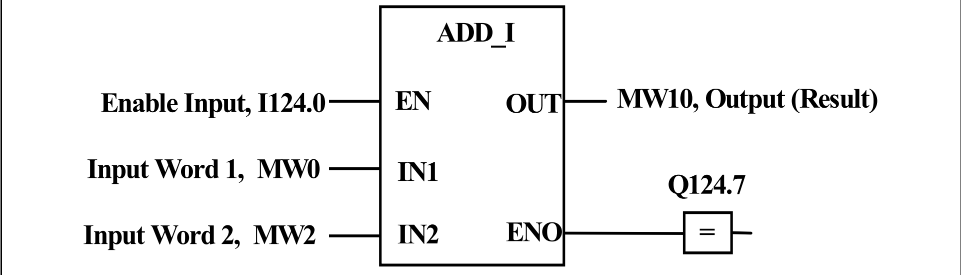


Figure 9-24. Example of an ADD Integer Instruction

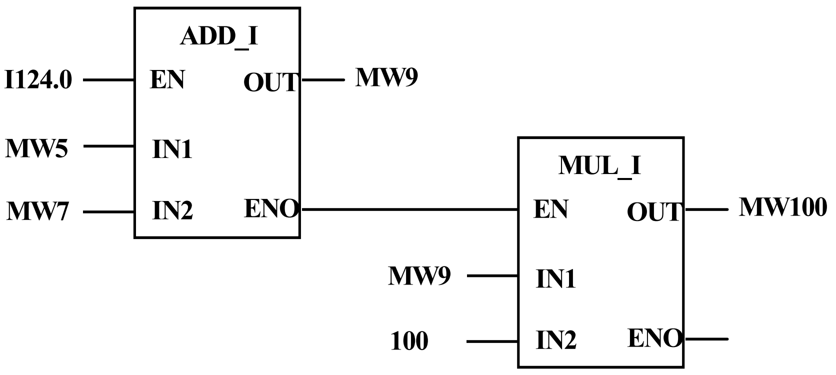


Figure 9-25. FBD Program for Math Problem of Example 9-5

Floating-Point Math Instructions

In this section, we will discuss the most common FBD instructions used to perform mathematical operations on real numbers. The basic floating-point math instructions add, subtract, multiply, and divide are listed in Table 9-5. The Siemens S7-300 PLC reference programming manual should be consulted for a complete listing of floating-point math instructions available in the S7 PLCs.

Instruction Symbol	Description
ADD_R	Add Real
SUB_R	Subtract Real
MUL_R	Multiply Real
DIV_I	Divide Integer

Table 9-5. Floating-Point Math Instructions

The floating-point math FBD instruction for an Add Real operation is shown in Figure 9-26. All the other floating-point math instructions have the same input and output functions and format. A signal state of 1 at the Enable (EN) input activates the math instruction. The instruction performs the math operation on the values at IN1 (input 1) and IN2 (input 2). If either of the inputs or the result are not floating-point numbers, the OV and OS bits of the status word are set to 1 and the Enable Output (ENO) bit of the instruction are set to 0. The result of the math operation is stored in the memory location programmed for the output (OUT) of the instruction.

In the Add Real instruction shown in Figure 9-26, a logic bit of 1 at input I124.0 activates the ADD_I instruction box. The result of the addition of double word MD0 and double word MD4 is entered into memory double word MD10. If either of the inputs or the result are not a floating-point number or if the logic state of the EN input I124.0 is 0, the ENO will be set to 0. However, if the result of the Add operation is valid and the input bit I124.0 is set to 1, the output bit Q124.0 will be set to 1.

EXAMPLE 9-6

Problem: Write a Siemens S7-300 FBD program to add the floating-point number in word MD0 to the floating-point in word MD4 and then divide the result by 1.5 if the input enable bit, I124.0 is true. Store the result in word MD20.

Solution: The FBD program to perform the math is shown in Figure 9-27.

Comparison Instructions

In this section we will discuss the three Siemens S7 Comparison Instructions: compare integer, compare double integer, and compare real.

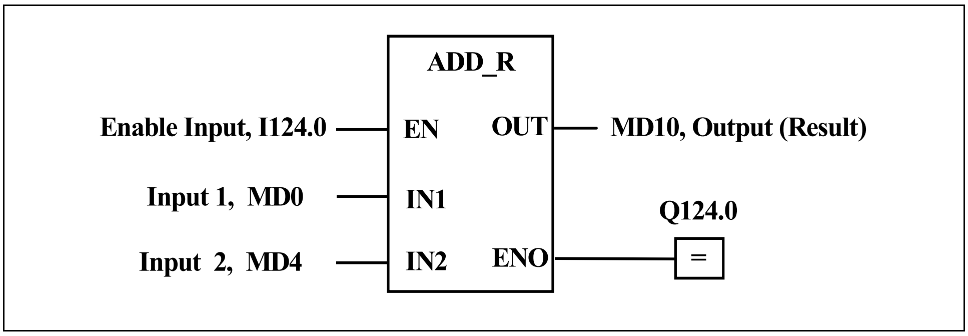


Figure 9-26. Example of an Add Real Instruction

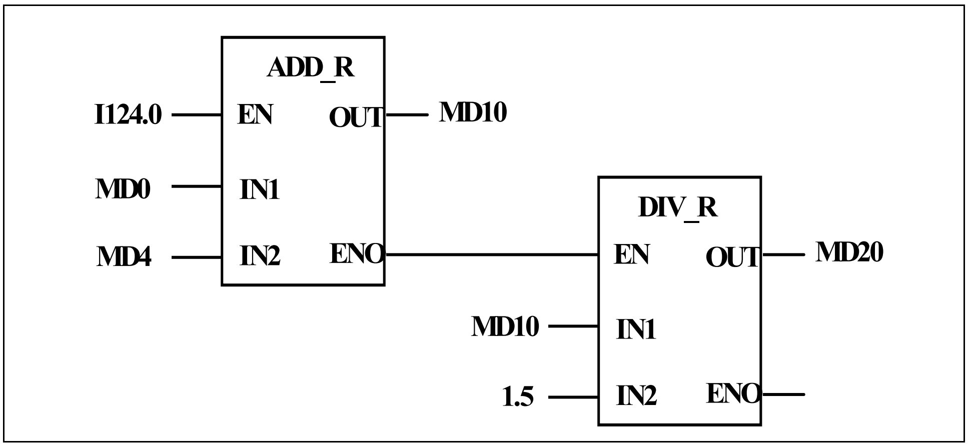


Figure 9-27. FBD Program for Math Problem of Example 9-6

The types of comparisons made in FBD programs are listed in Table 9-6 with their relational operator used in the compare instruction box.

Types of FBD Comparisons	Relational Operator
Input 1 is equal to Input 2	==
Input 1 is not equal to Input 2	<>
Input 1 is greater than Input 2	>
Input 1 is less than Input 2	<
Input 1 is greater than or equal to Input 2	>=
Input 1 is less than or equal to Input 2	<=

Table 9-6. Types of FBD Comparisons

Compare Integer

The *Compare Integer* instruction compares two values on the basis of 16-bit whole numbers. This instruction compares inputs IN1 and IN2 according to the type of comparison that is selected from the list of relational operators in Table 9-6.

If the comparison is true, the result of the operation is 1. The comparison result cannot be negated, but the same effect as negation can be obtained by using the opposite compare function. An example compare integer is shown in Figure 9-28. If the value in word MW0 is equal to the value in word MW2 and the logic state of input I124.0 is 1 then output bit Q124.0 is set to 1.

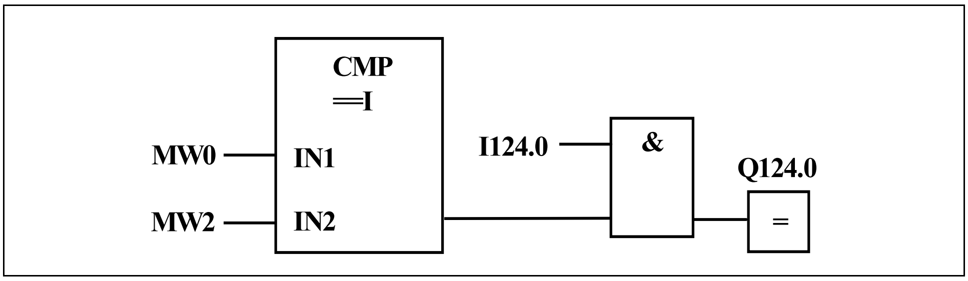


Figure 9-28. Example FBD Compare Integer Instruction

Compare Double Integer

The *Compare Double Integer* instruction compares two values on the basis of 32-bit whole numbers. This instruction compares inputs IN1 and IN2 according to the type of comparison that is selected from the list of relational operators in Table 9-6.

If the comparison is true, the result of the operation is 1. The comparison result cannot be negated, but the same effect as negation can be obtained by using the opposite compare function. An example compare double integer is shown in Figure 9-29. If the value in word MW0 is not equal to the value in word MW4 and the logic state of input I124.0 is 1, then output bit Q124.0 is set to 1.

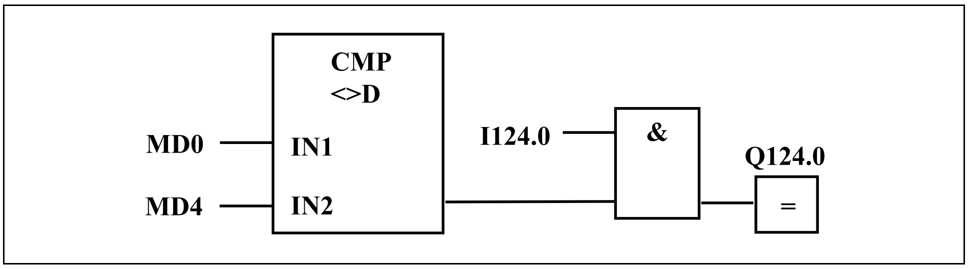


Figure 9-29. Example FBD Compare Double Integer Instruction

Compare Real

The *Compare Real* instruction compares two values on the basis of floating-point numbers. This instruction compares inputs IN1 and IN2 according to the type of comparison that is selected from the list of relational operators in Table 9-6.

If the comparison is true, the result of the operation is 1. The comparison result cannot be negated, but the same effect as negation can be obtained by using the opposite compare function. An example compare real is shown in Figure 9-30. If the value in word MD0 is greater than the value in word MD4 and the logic state of input I124.0 is 1, then output bit Q124.0 is set to 1.

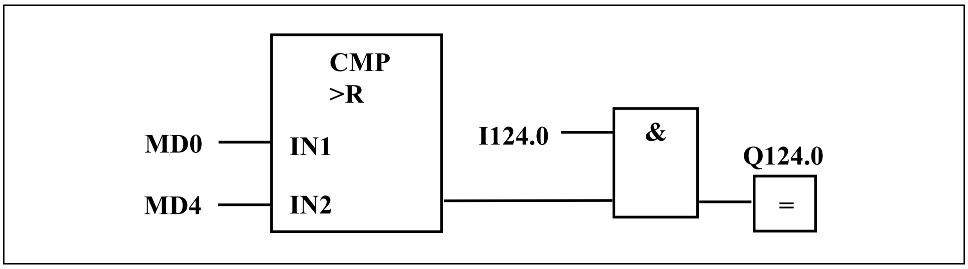


Figure 9-30. Example FBD Compare Real Instruction

EXERCISES

9.1 What are the four types of elements and boxes used in FBD programming?

9.2 Write a FBD program to start a pump that will fill a process tank with fluid until a high level is reached in the tank. Assume that a normally opened fill tank pushbutton is wired to a discrete input module at I124.4 and a normally closed tank high-level switch is connected to input point I124.5 on a Siemens CPU314IFM. Also assume that the pump start relay is connected to output Q124.0 on the same Siemens PLC.

9.3 Write a FBD program to control an electric motor. Assume that a NO start pushbutton is wired to an input point I124.0 and a NC stop pushbutton is wired to input I124.5 of a Siemens CPU314IFM. Also assume that output Q124.1 of the Siemens PLC will control a motor start relay, and there is an auxiliary motor start contact (NO) connected to the PLC input I124.1.

9.4 What is a typical application for a timer instruction?

9.5 List the five timer instructions available in the Siemens S7-300 FBD programming instruction set.

9.6 Write a FBD program to control the temperature of the fluid in a process tank close to 400°C. Assume that the heater contactor is connected to PLC output Q124.1, a temperature low switch set is connected to PLC input I124.0, and a temperature high switch is connected to PLC input I124.1 on a Siemens CPU314IFM. The temperature low switch is closed if the fluid temperature is below 395°C and opens at a temperature of 395°C or higher. The temperature high switch is closed if the fluid temperature is below 405°C and opens at 405°C or higher temperatures.

9.7 Write a FBD program for a Siemens CPU314IFM to activate a pump starter relay connected to output point Q124.0, if the inlet control valve FV-1 on a process tank is opened (input bit I124.0 = 1) and the process tank is at a high level (input bit I124.2 = 1).

9.8 Write a FBD program for a Siemens CPU314IFM to open a fill valve on a process tank to allow an ingredient to be added to the tank for 10 seconds each time the Tank Level High Switch LSH-100 is activated (logic = 1). Assume that switch LSH-100 is connected to input I124.0 and the fill valve is wired to PLC output point Q124.0.

9.9 Write a FBD program for a Siemens S7-300 PLC to turn off a conveyor belt on a production line after 50 parts have been produced. Assume the following for the control program: 1) Output bit Q124.5 = 0, turns off the conveyor belt; 2) Input I124.1 is set to 1 each time a production part is rejected; 3) Input I124.2 is set to 1 each time a new part is produced; 4) A normally open (NO) pushbutton connected to input I124.6 is used to set the production count to 50, and 5) A NO pushbutton connected to input I124.7 is used to reset the counter to zero and to stop the conveyor belt.

9.10 What is the purpose of FBD integer math functions?

9.11 Write a FBD program for a Siemens S7 to subtract the integer data in word MW20 from the integer data in word MW18 and store the result in word MW22 if the input bit I:124.0 is true. Then divide the result by 2 and store the final result in word MW24.

9.12 Write a FBD program for a Siemens S7 to subtract a 32-bit floating-point number in word MD70 from a 32-bit floating-point number in word MD74 and store the result in word MD78 if input I124.0 is set to 1. Then divide the result by 4.5 and store the final result in word MD82.

BIBLIOGRAPHY

1. *Simatic S7 Function Block Diagram (FBD) for S7-300 and S7-400 Programming Reference Manual*, Siemens AG, October 1998.
2. Hughes, T. A., *Basics of Measurement and Control*, ISA – The Instrumentation, Systems, and Automation Society, 2nd edition, 1995.

10 Data Communication Systems

Introduction

The purpose of communications is to transfer information from one point to another or from one system to another. In process control, this information is called *process data* or simply, *data*.

An understanding of data communications is essential for the proper application of programmable controllers to process control and data collection. This chapter will provide a basic understanding of data communications terminology and concepts and their application to programmable controller systems.

Basic Communications

Data is transmitted through two types of signals: baseband and broadband. In a *baseband system*, a data transmission consists of a range of signals sent on the transmission medium without being translated in frequency. A telephone call is an example of a baseband transmission. A human voice signal in the 300- to 3000-Hz range is transmitted over the phone line in the same frequency range. In a *baseband system*, there is only one set of signals on the medium at a time.

A broadband transmission consists of multiple sets of signals. Each set of signals is converted to a frequency range that will not interfere with other signals on the medium. Cable television is an example of broadband transmission.

As shown in Figure 10-1, three basic components are required in any data communication system: the *transmitter* to generate the information, the *medium* to carry the data, and the *receiver* to detect the data.

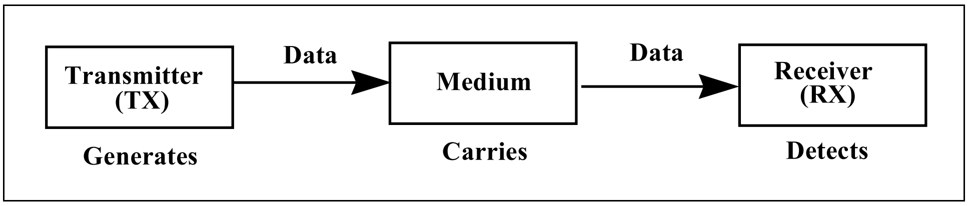


Figure 10-1. Basic Communication System Components

The medium can be divided into more than one channel. A channel is defined as a path through the medium that can carry information in only one direction at a time.

Figure 10-2 shows an example of a multi-channel medium with *n* channels. Probably the most common example of multi-channel communication systems is cable television systems.

Physical transmission media fall into four general categories: multiconductor cable, twisted-pair cable, coaxial cable, and fiber optic cable. Multiconductors consist of two or more insulated electrical conductors inside a plastic-covered cable. A typical example is the parallel printer cable used between a personal computer and a printer. Twisted-pair cables consist of two electrical conductors, each covered with insulation. The two wires are twisted together to ensure that they are both equally exposed to electrical interference signals in the environment. Since the wires are carrying current in opposite directions, the electrical interference will tend to cancel out in the cable. Twisted pair is the most common cable used in programmable controller systems. It is the least expensive transmission medium and provides adequate electromagnetic interference (EMI) immunity.

A coaxial cable consists of an electrical conductor surrounded first by insulating material and then by a tube-shaped metal braid conductor. In most cases, an insulator covers the entire cable. The round conductor in the center of the cable and the circular outer tube conductor are coaxial in that they share the same central axis.

Coaxial cables are used in both baseband and broadband communications systems. A variety of cable types are used in process control applications, ranging from flexible types similar to coaxial cables used on home TV sets to heavy, rigid cables that require special installation procedures.

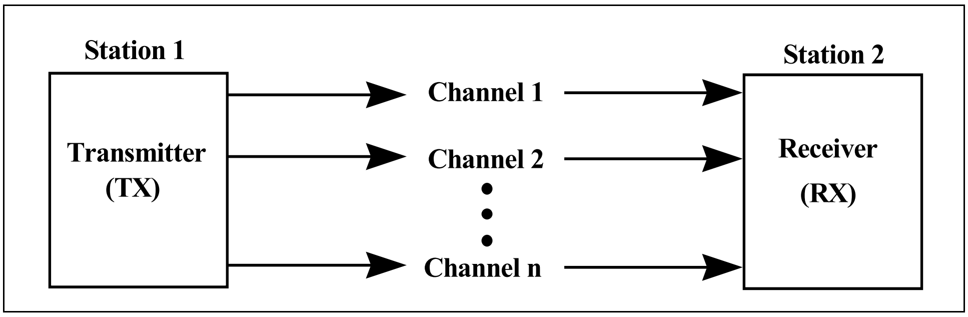


Figure 10-2. *n*-Communication Channels

Coaxial cables are generally used in process automation applications where long communications runs of over 1000 feet are involved and improved EMI immunity is required. They find very extensive use in plant-wide communication networks.

The fiber optic cable consists of fine glass or plastic fibers. At one end, electrical pulses are converted into light by a photo diode and sent down the fiber optic cable. At the other end of the cable, a light detector converts the light pulses back to electrical signals. The light signals can travel only in one direction, so two-way transmission requires two separate fiber cables. A fiber optic cable is normally the same size as a twisted-pair cable, and it is immune to electrical noise.

The cost of the fiber optic cables is about the same as coaxial cables. However, the use of fiber optics in programmable controller applications has been limited by the high cost of connectors and lack of industrial standards.

The number of channels used to effect the information flow can also describe communication. The three common methods of data transmission are unidirectional, half-duplex, and duplex.

Unidirectional Communications

In unidirectional communications, a single channel is used and communication is only one way (i.e., from transmitter to receiver), so the receiver can never respond.

In unidirectional communications, it is not possible to send errors or control signals from the receiver station because the transmitter (TX) and the receiver (RX) are each dedicated to performing one function. A typical example of unidirectional communications in process control would be a weigh scale in the field sending data to a programmable controller in a central control room.

Half-Duplex Communications

Two-way communication allows the receiver to verify that the data was received. One kind of two-way communication is called *half-duplex*. In half-duplex, a single channel is used and communication is two-way, but communication can occur only in one direction at a time (see Figure 10-3). In this configuration, the receiver and transmitter alternate functions so communication occurs in one direction at a time on the single channel. A circuit in the station communication module acts as an OR gate to place the unit in transmit or receive mode.

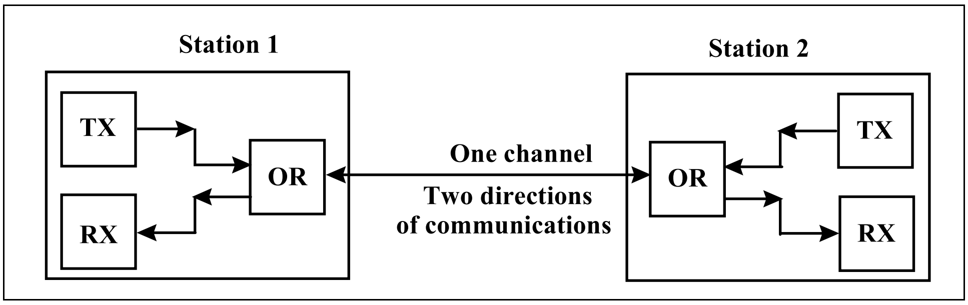


Figure 10-3. Half-Duplex Communications

Full-Duplex Communication

Two-way communications where data can flow in both directions at the same time is called *full-duplex communications*. In this case, two physical paths are required so information can flow from both directions simultaneously as shown in Figure 10-4.

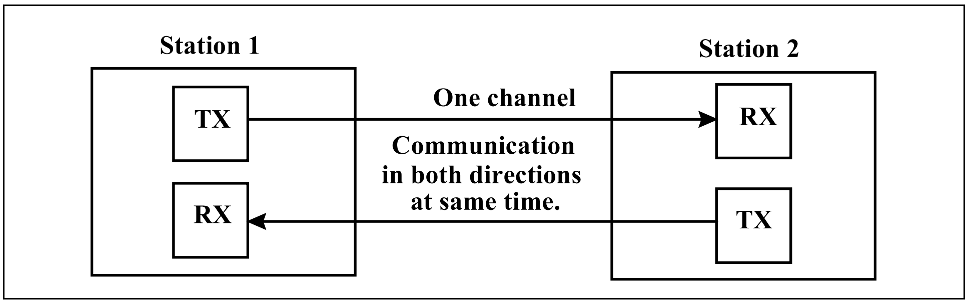


Figure 10-4. Full-Duplex Communication

Transmission Methods

Transmission involves moving data between a receiver and a transmitter. The two methods of transmission are *parallel* and *serial*.

Parallel Transmission

In parallel transmission, all bits are transmitted at the same time, and each bit of information requires a unique channel. For example, if we transmit an eight-bit ASCII character, eight channels are required. Figure 10-5 demonstrates the transmission of the ASCII character using odd parity. The term "parallel" refers to the position of the bits of the character and the fact that the characters are transferred as a group of bits in parallel. The use of a group of channels results in a high data transfer rate.

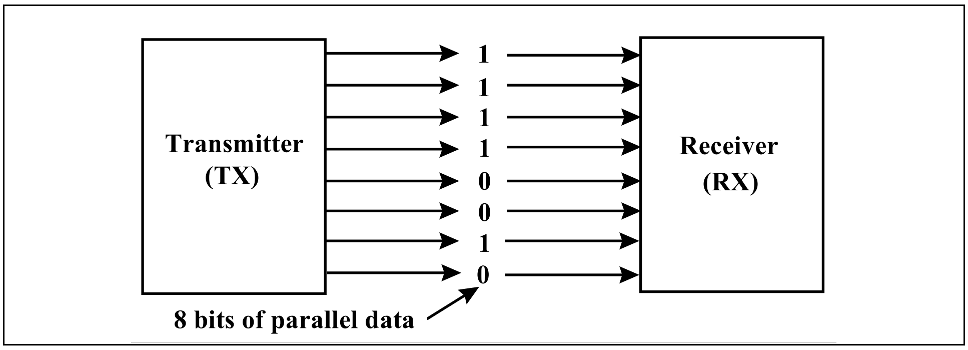


Figure 10-5. Parallel Data Transmission

There is a problem with bit synchronization delay in parallel data transfer. Synchronization is the process of making two or more activities happen at the same rate and time. If we send a string of characters on a long parallel cable, the difference in impedance in the cable wires can cause bit synchronization decay, as shown in Figure 10-6.

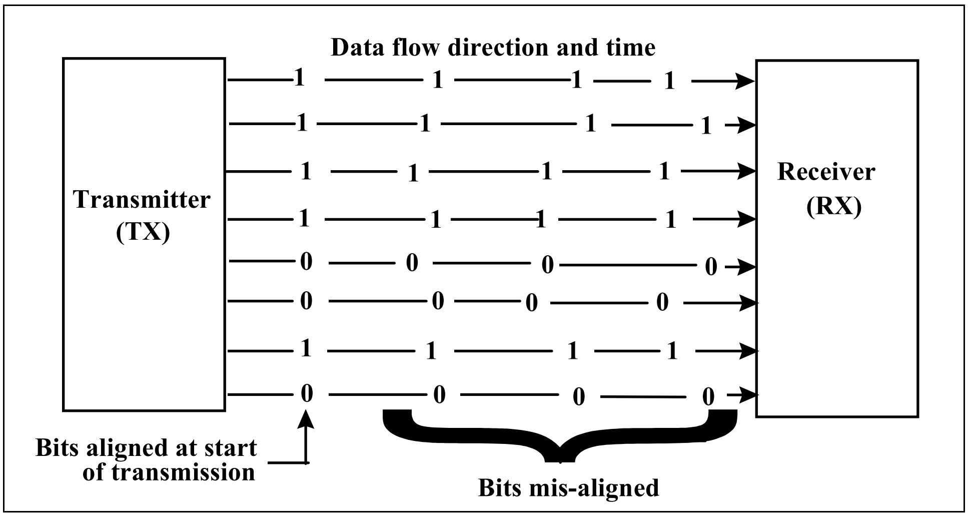


Figure 10-6. Bit Synchronization Decay

To overcome this problem, the distance between the receiver and the transmitter must be kept short to avoid transfer errors caused by bit synchronization decay. For this reason, parallel transmission is used only in closely connected data systems where high speed is required. For example, data transfer on data and address buses in a microprocessor is parallel transmission. However, in process control systems, the data transfer between systems such as PCs and programmable controllers is normally serial. For this reason, we will limit our discussion in the remaining parts of this chapter to serial data transmission.

Serial Transmission

In this section, we will discuss serial interfacing between microprocessor-based devices. Serial interfacing is probably the least complex electrical interface to implement because it requires only one signal wire to carry all data flow in one direction and only two wires for bi-directional flow. Serial interfaces do, however, require additional logic circuitry to convert between parallel and serial data because most computers process data in parallel. This extra circuitry is not required in parallel transmission of information. Because serial links are low in cost and easy to install, they have been standardized into a few widely used protocols (transmission control standards) where several low-cost integrated circuit (IC) interface chips are available.

In serial transmission, the bits of the encoded character are transmitted one after another in a single channel (see Figure 10-7). The transmission takes the form of a bit stream that the receiver must assemble into characters (normally 8 bits) using specially designed ICs.

The main advantages of serial communications are lower cost and the elimination of byte synchronization problems.

Controlling the speed of transmission is critical in serial data transfer. A common measurement unit used to describe serial data speed is *bits per second (bps)*.

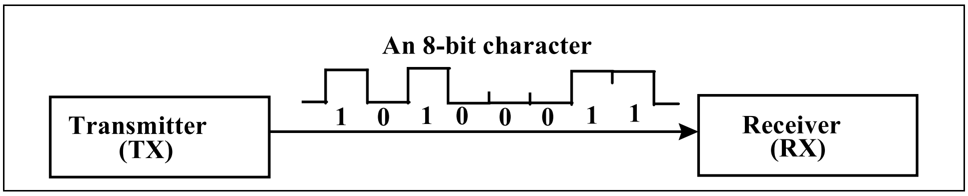


Figure 10-7. Serial 8-Bit Character Transmission

The process of bit synchronization is required for serial data transfer. A timing circuit at the transmitter places transmitted bits on the channel at fixed intervals determined by the selected communication rates. Some typical communication rates used are 2.4, 4.8, 9.6, 14.4, and 28.8 Kbps.

The transmitter must send bits at the same rate the receiver is set to accept bits. For example, if a PC is transmitting serial data at 9600 bps to a programmable controller system, the communications module on the programmable controller must be set at a communications rate of 9600 bps.

Character synchronization is used to determine which eight consecutive bits in a data stream represent a character. The receiver must recognize the first data bit and count bits until the character or byte is complete. Various forms of character synchronization are used in data communications. Sometimes each byte is framed with special bits to indicate start and stop.

Signal Multiplexing

A single line can be used to carry more than one signal by using signal multiplexing. To understand multiplexing, we must first define the term "bandwidth. " Bandwidth describes the signal-carrying capability of a communications channel or line. Bandwidth is defined as the difference in cycles per second between the lowest and highest frequencies a channel can handle without a certain amount of signal loss.

Multiplexing allows one line to serve more than one receiver by creating slots or divisions in the line. The equipment used to obtain multiplexing is called a *multiplexer* or *MUX* (see Figure 10-8).

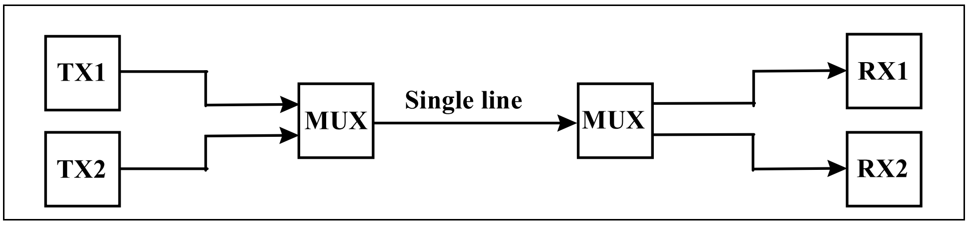


Figure 10-8. Signal Multiplexing

The commonly used signal multiplexing methods are frequency division, time division, and statistical multiplexing.

Frequency Division Multiplexing

In frequency division multiplexing, each channel is assigned its own individual frequency range. Frequency division is normally used to combine a group of low-speed sources onto a single voice line. A typical application of frequency division communications is shown in Figure 10-9.

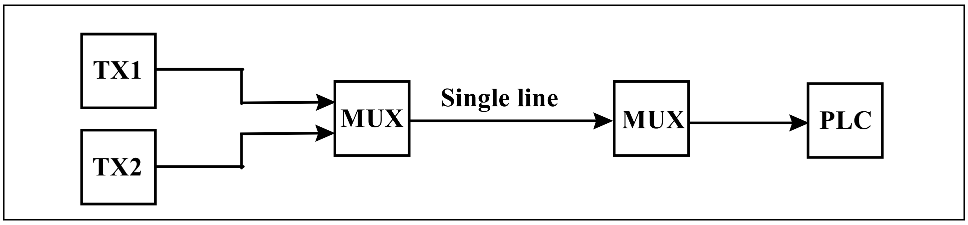


Figure 10-9. Frequency Division Multiplexing

Time Division Multiplexing

Time division multiplexing uses time periods to assign channel space and allot the available bandwidth. In Figure 10-10, two time periods (Period 1 and Period 2) are used to assign each channel its own individual time slot in the data stream. In this type of multiplexing, no other channel can use the time slot so there is some wasted bandwidth when a station is not transmitting data in its time slot. However, one transmission line can support many data streams. It is normally used where there is a need to combine a number of relatively low-speed data transmissions onto a single high-speed line.

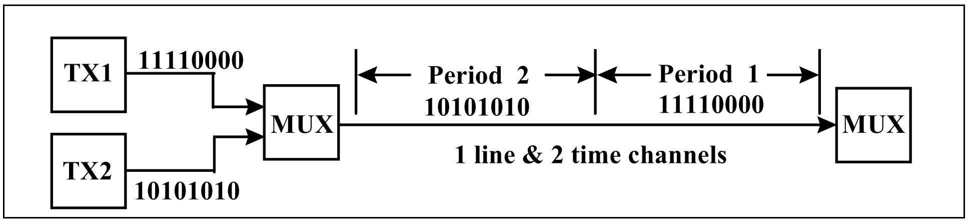


Figure 10-10. Time Division Multiplexing

Statistical Multiplexing

Statistical multiplexing is an enhancement of time division multiplexing designed to reduce the wasted bandwidth when a station is transmitting data. This enhancement process is sometimes called *data concentration*. Statistical multiplexing is generally used where a large number of data entry terminals require only a brief or occasional data transfer. If the use of the terminals becomes intensive, the operators will experience long time delays, so this method must be applied with caution.

Error Control and Checking

Error control and checking is the process used to detect any discrepancy between transmitted data and received data in a communication system. Errors detected at the receiver are either corrected or retransmitted. The common error control methods used in programmable controller systems are (1) echo check, (2) vertical redundancy check (VRC), (3) longitudinal redundancy check (LRC), and (4) cyclical redundancy check (CRC).

Echo Check

Echo check is used in two-way communications systems to check the accuracy of transmitted data (see Figure 10-11).

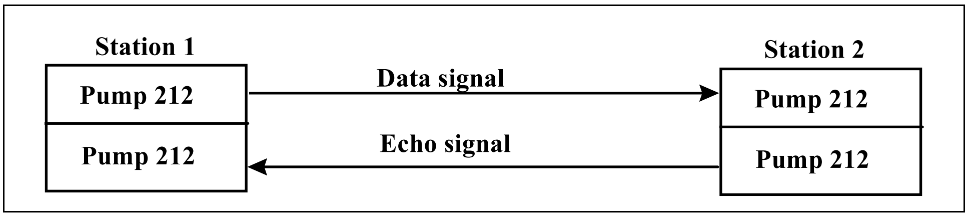


Figure 10-11. Echo Check

The receiver (RX) in station 2 sends every character received back to the originating terminal (station 1). Station 1 compares the echoed data with the information it sent. The comparison is performed by electronic circuitry in the transmitter/receiver and if a transmission error is detected, the information is retransmitted a fixed number of times to obtain an error free transmission.

Vertical Redundancy Check (VRC)

In the vertical redundancy check process, parity bit checking is used to detect the change of a single bit. In this method, a single bit is added to a character string to create either an odd or an even number of 1 bits. Parity bits are often called “redundant” because they can be removed without loss of data. If even parity is used, a 1 bit is added to a character string to make the total of 1 bits even. For example, the ASCII letter “A” is given by 1000000 (see Table 10-1); if even parity is used, a 1 bit is added to the binary string before transmission (i.e., A = 1000 0001). If odd parity is employed, a 0 bit is added to the binary string to made the 8-bit binary string A = 1000 0000.

Bits	**7 6 5**	**0 0 0**	**0 0 1**	**0 1 0**	**0 1 1**	**1 0 0**	**1 0 1**	**1 1 0**	**1 1 1**
4321	HEX	0	1	2	3	4	5	6	7
0000	0	NUL	DLE	SP	0	@	P	‘	p
0001	1	SOH	DC1	!	1	A	Q	a	q
0010	2	STX	DC2	“‘	2	B	R	b	r
0011	3	ETX	DC3	#	3	C	S	c	s
0100	4	EOT	DC4	$	4	D	T	d	t
0101	5	ENQ	NAK	%	5	E	U	e	u
0110	6	ACK	SYN	&	6	F	V	f	v
0111	7	BEL	ETB	’	7	G	W	g	w
1000	8	BS	CAN	(	8	H	X	h	x
1001	9	HT	EM	)	9	I	Y	i	y
1010	A	LF	SUB	*	:	J	Z	j	z
1011	B	VT	ESC	+	;	K	[	k	{
1100	C	FF	FS	’	<	L	\	l	|
1101	D	CR	GS	-	=	M	]	m	}
1110	E	SO	RS	.	>	N	^	n	~
1111	F	SI	US	/	?	O	-	o	DEL

Table 10-1. ASCII Code

In even parity VRC transmission, the receiver would detect an error when it receives a character that contains an odd number of 1 bits. In this method, after a parity error is detected, the receiver would request retransmission of the data.

An example problem will help to explain the concept of parity.

EXAMPLE 10-1

Problem: Generate the ASCII code with even parity for the word STOP.

Solution: Using the ASCII code in Table 10-1, we first obtain the binary code for each character as follows:

Bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**Character**
	1	0	1	0	0	1	0	S
	1	0	1	0	1	0	0	T
	1	0	0	1	1	1	1	O
	1	0	1	0	0	0	0	P

EXAMPLE 10-1 continued

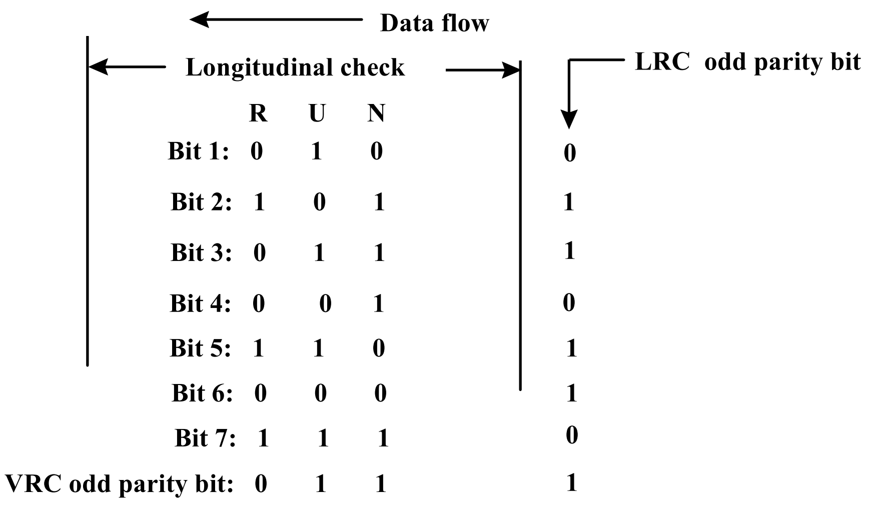
Next, we count the number of 1 bits for each character. If the number is even, we set the parity (P) bit to 0 to obtain an even number of 1 bits in the binary string. On the other hand, if the number of 1 bits is odd, we set the parity bit to 1 to obtain an even number of 1 bits in each string as shown:

Bits	P	7	6	5	4	3	2	1	Character
	1	1	0	1	0	0	1	0	S
	1	1	0	1	0	1	0	0	T
	1	1	0	0	1	1	1	1	O
	0	1	0	1	0	0	0	0	P

Longitudinal Redundancy Check

Longitudinal redundancy check (LRC), or block check character (BCC), is a procedure that checks an entire horizontal line within a block of data for odd or even parity. This process is used in combination with VRC. While VRC can detect a single error, the only way to obtain corrected data is retransmission. When LRC is used with VRC, any single bit error in an entire data block is not only detected but can also be corrected at the transmitter without a retransmission. This increases the overall data transmission speed for a system.

To illustrate the LRC process, let us assume we are transmitting the data word RUN in ASCII code using odd parity as shown in Figure 10-12.



	R	U	N	LRC odd parity bit
Bit 1:	0	1	0	0
Bit 2:	1	0	1	1
Bit 3:	0	1	1	1
Bit 4:	0	0	1	0
Bit 5:	1	1	0	1
Bit 6:	0	0	0	1
Bit 7:	1	1	1	0
VRC odd parity bit:	0	1	1	1

Figure 10-12. Longitudinal Redundancy Check Example

In Figure 10-12, an LRC odd parity character is sent by the transmitter. This character is generated by adding a parity bit at the end of the entire transmitted block so that an odd number of 1 bits is created for each longitudinal row of bits.

At the receiver, the LRC is calculated for the data bytes in the block, and it is compared with the transmitted LRC character. If they are not equal, the vertical parity bit reveals which byte is in error and the LRC reveals which of the 8 bits is in error. Logic circuitry or software in the receiver is used to change the error bit to its opposite state, correcting the error. Only if there is more than one error in the block transfer must retransmission be requested by the receiver.

EXAMPLE 10-2

Problem: Calculate the LRC/VRC odd parity characters for the word PUMP.

Solution: Using the ASCII code in Table 10-1, we first obtain the binary code for each character as shown in the table. Next, we count the number of 1 bits for each character. If the number is odd, we set the vertical parity bit (bit 8) to 0 to maintain an odd number of 1 bits in the binary string. On the other hand, if the number of 1 bits is even, we set the parity bit to 1 to obtain an odd number of 1 bits in each vertical string as shown in Table 10-2.

Word	P	U	M	P	LRC Parity
Bit 1	0	1	1	0	1
Bit 2	0	0	0	0	1
Bit 3	0	1	1	0	1
Bit 4	0	0	1	0	0
Bit 5	1	1	0	1	0
Bit 6	0	0	0	0	1
Bit 7	1	1	1	1	1
Vertical Parity Bit	1	1	1	1	0

Table 10-2. VRC/LRC Odd Parity Example

Finally, we count the number of 1 bits in each row to generate the longitudinal redundancy check (LRC) odd parity bit. If the number of one's is odd, we set the LRC parity bit to 0 to maintain an odd number of 1 bits in the longitudinal row. On the other hand, if the number of 1 bits is even, we set the LRC parity bit to 1 to obtain an odd number of 1 bits in each row as shown in Table 10-2.

Cyclical Redundancy Check

Cyclical redundancy check (CRC) is a method for checking error on an entire data block. In this method, a check character is transmitted at the end of each block. The transmitter calculates the check character from the data transmitted.

The receiver compares the transmitted CRC to its own calculated CRC based on the received block of data. If they are not equal, the receiver requests retransmission of the previous message block. If they are equal, the transmitted data is assumed to be error-free.

CRC is a very accurate method for detecting data transmission errors. It is the most common method used in programmable controller–based systems.

Communication Protocols

Communications protocol is a set of rules specifying the format and control of transmission between two communication devices. The two main functions of protocol are *handshaking* and *line discipline*. Handshaking determines whether a circuit is available and makes sure the circuit is ready to transfer data. Line discipline performs the following functions: (1) receive and transmit information, (2) error control procedures, (3) sequencing of the message blocks, and (4) error checking.

To illustrate the concept of communications protocol, a typical line discipline sequence between a transmitter and receiver is shown in Figure 10-13.

The two basic transmission methods used to obtain line discipline are *asynchronous* and *synchronous*. In *asynchronous*, successive data appear in the data channel at arbitrary times with no specific clock control governing the time delays between information. In *synchronous*, each successive datum in a data stream is controlled by a master data clock and appears at a specific interval of time.

Most synchronous and asynchronous systems deliver serial data in 8-bit characters. The asynchronous systems treat each character as an individual message and the characters appear in the data stream at arbitrary relative times. However, within each character, the 8 bits are transmitted at a fixed predetermined clock rate such as 2400 bps, 9600 bps, 14.4, 28.8, and 33.6 Kbps. Hence, an asynchronous communications system is actually synchronous within a character and asynchronous between characters.

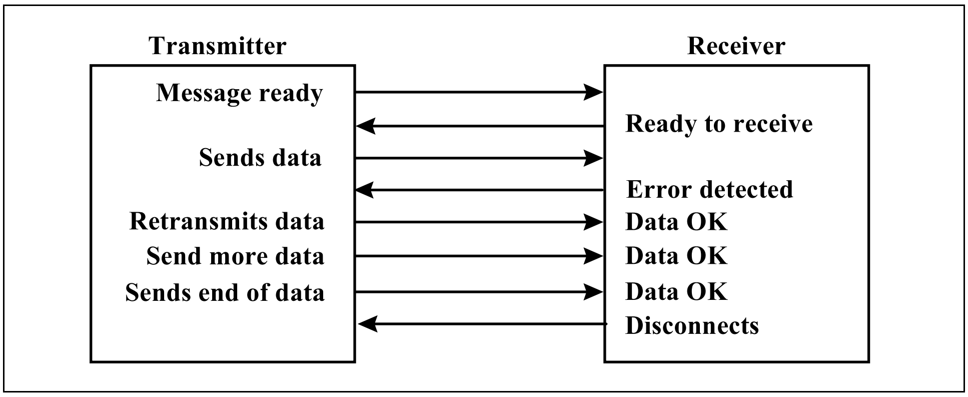


Figure 10-13. Typical Line Discipline Sequence

A typical bit stream for an asynchronous transmission is shown in Figure 10-14. Line discipline is achieved by framing every character transmitted with start/stop bits. The start bit is a 0 or "space" and the stop bits are a 1 or "mark."

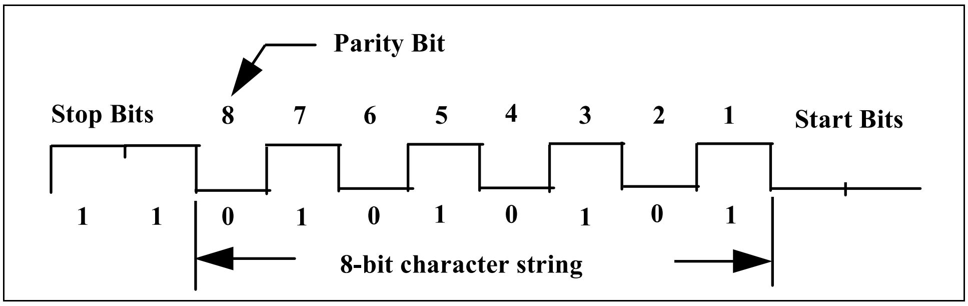


Figure 10-14. Typical Asynchronous Character String

Asynchronous transmission operates at random speeds and is almost always used with half-duplex protocols.

Synchronous transmission sends an entire message at one time and uses special character strings to achieve line discipline. It includes a predefined start and end sequence and normally operates at speeds of 2400 bps and higher. A typical sequence begins with two synchronous (syn) characters as shown in Figure 10-15.

In Figure 10-15, the start-of-message (SOM) character indicates the beginning of the data transfer and the control character gives the receiver control information on the data transfer. The actual data can vary in length and is followed by error checking which is normally CRC. The end-of-message (EOM) is used to signal the end of a data transmission.

The two transmission modes used for line discipline are half-duplex and full-duplex protocols. Half-duplex protocol allows for communications in only one direction. In a two-wire half-duplex circuit, the stations must be able to switch automatically from transmit to receive. In a four-wire full-duplex circuit, each station has a dedicated receiver and transmitter circuit, but data cannot be transmitted and received at the same time. Full-duplex protocol permits simultaneous transmissions in both directions. It falls back to half-duplex operation if the communications link cannot support full-duplex.

Serial Synchronous Transmission

Synchronous communications depend on the data and protocol control information being assembled into a structured and predefined package or format. The format tells the computer or other equipment what to do with the message and how to do it.

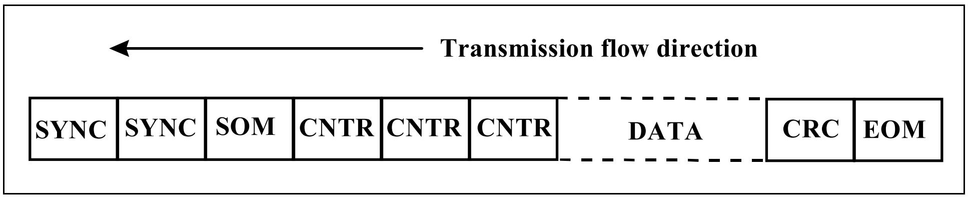


Figure 10-15. Typical Synchronous Transmission

A message will contain one or more fields of data. Each synchronous transmission in a message block format will contain three major parts: (1) a header, (2) text, and (3) the trace or trail block, as shown in Figure 10-16.

The *header* starts the message package and provides the capability for synchronization between the transmitter and receiver. It normally contains characters that identify the transmitting or receiving stations and provides communications routing data. The *text* contains the data characters to be transferred and consists of a single- or multiblock message. The *trace* or *tail* characters signal the end of a transmission block.

Serial Synchronous Protocols

Different control and data formats are used by different protocols. A communications protocol can be defined as a fixed set of rules governing the format and control of inputs and outputs between two data transfer devices. In a standard data transfer system, a protocol governs the following: line control, framing, error control, and sequence control.

Line control is used to list the station that will transmit and the station that will receive in half-duplex communications. It is also required in a multipoint circuit (a circuit connecting three or more points on a common line). The *framing* function normally determines which 8-bit groups are the characters and which groups of characters are the messages. *Error control* detects transmission errors using various redundancy checks and normally corrects faulty messages. *Sequence control* function numbers the messages to eliminate duplication and avoid data loss. It also identifies any retransmitted messages.

There are basically three serial data protocols in wide use today in process control applications:

1. BISYNC (Binary Synchronous Communications), an older protocol used in IBM equipment.
2. HDLC (High-level Data-Link Control), probably the most widely used protocol in programmable controllers.
3. TCP/IP (Transmission Control Protocol/Internet Protocol), originated with the UNIX operating system. It was selected for use on the Internet. It had a relatively low use in networking until the Internet became widely used in the early 1990s.

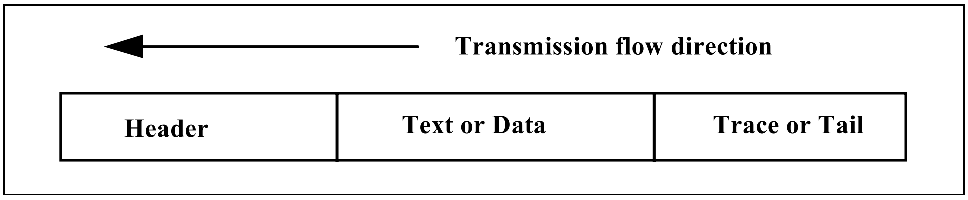


Figure 10-16. Three Elements of a Typical Synchronous Transmission

HDLC has evolved from two earlier standards, the advanced data communications control procedure (ADCCP) and the synchronous data link control (SDLC). Because of the close similarity of the three, we will cover only the HDLC protocol in this discussion. But first, we need to discuss the BISYNC protocol.

BISYNC Protocol

BISYNC stands for BInary SYNChronous communications; it is a half-duplex character-oriented protocol. The protocol has a very rigid format that uses special characters (ASCII or EBCDIC) to delineate the various fields of a message and to control the required protocol functions.

A typical BISYNC message, shown in Figure 10-17, consists of the following discrete parts: (1) two or more synchronizing characters (SYNC), (2) start of header (SOH), (3) header, (4) start of text (STX), (5) text, (6) an end of text (ETX), and (7) a trace or tail block (CRC).

The synchronizing (SYNC) characters are used to establish the correct timing between the transmitter and receiver. The number of SYNC characters varies with the different communication applications and networks. The start of header is a format control character that is transmitted just before the header character to identify the individual message control characters. The header is an optional character that normally contains routing or message priority information. The start of text (STX) is a special format control character that is transmitted before the first data characters; it indicates that the characters to follow are information. The text, of course, is the data that is being transmitted. The end of transmission block (ETB) is a format control character indicating the end of text and the beginning of the trace or tail (CRC). It is normally used to indicate the end of an intermediate text block.

The end of text (ETX) is also a special format control character that indicates the end of a text block and the start of the trace or tail (CRC). The trace or tail block (CRC) detects and corrects errors in transmission. It depends on the information code being used, such as ASCII or EBCDIC, and has a block check character or a combination of checks.

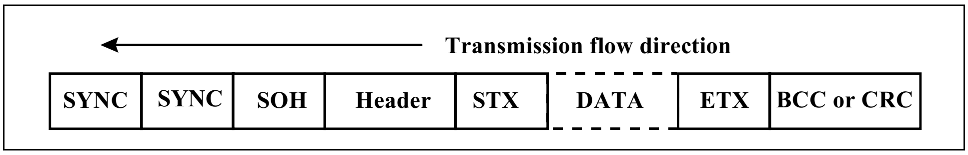


Figure 10-17. A BISYNC Transmission String

If the trace or tail block is in ASCII, a VRC/LRC message check is performed. In EBCDIC transmission, normally no VRC/LRC check is performed and the CRC is calculated on the entire message.

The BISYNC is a rather simple protocol, but several problems complicate it. The format of the protocol places special meaning on the ETX character. If the data block (or control information) contains this character among its data, the characters can be misinterpreted. For example, if the datum happens to be an 8-bit pattern identical to the ASCII pattern for ETX, this datum character could deceive the receiver into taking an end-of-block action when actually the message block has more characters to follow. To correct this problem, the protocol needs to distinguish specific patterns as data characters. The ability to treat control characters either as control information or as data is called *data transparency*.

BISYNC uses the control character DLE (data link escape) to obtain the required transparency. If a control symbol is to be treated as data, the DLE character precedes it. In other words, the receiver is warned by the receipt of a DLE to accept the next character as data and not to take any control action. The use of the DLE character is somewhat more complicated than described because of other special considerations. For example, to maintain transmission synchronization in the absence of data in the transmitter queue, the protocol provides for an automatic insertion of sync characters that are ignored by the receiver. But it is possible that a sync character might be inserted between a DLE and a control character that follows it. This condition forces the receiver to interpret the sync character as if it were data rather than accepting the next character as data. In this situation, the transmitter cannot simply send a sync character after a DLE. The transmitter must queue the DLE and then send sync characters until both the DLE and its corresponding data character are ready. If message buffering is not available, the transmitter has to send a DLE and then stay idle on the line. In this situation, it should transmit in the idle state the characters DLE and SYNC that the receiver can ignore as pairs of characters. Also note that since the DLE is a control character with special significance, the DLE character in the data transmission must be treated the same way the ETX is treated, and it must be preceded by a DLE character when transmitted over a communications link. An example problem will help to illustrate the BISYNC protocol.

EXAMPLE 10-3

Problem: Assemble the bit streams required to send the message M1 ON using 8-bit odd parity ASCII code and BISYNC protocol. Assume the header is given by 00000001 for transmitting station 1 and omit the check character (BCC or CRC) at the end of the transmission.

Solution: Use the ASCII code Table 10-1 to find the bits required as follows: SYNC = 00010110, SOH = 00000001, STX = 00000010, M = 11001101, 1 = 00110001, space = 00100000, 0 = 01001111, N = 11001110, ETX = 10000011.

The HDLC Protocol

The main feature of HDLC protocol is that it opens and closes each message block or frame with start-frame and stop-frame characters (flags). A typical HDLC message is shown in Figure 10-18 and consists of six discrete parts as follows: (1) open flag, (2) address byte, (3) control byte, (4) data field, (5) a check field, and (6) a close flag. The open flag always consists of the same 8 bits (01111110); it is used to indicate the start of a transmission frame. This 8-bit sequence is never repeated again throughout the entire message until the close flag.

The address byte is the address of the transmitter on a command message or the receiver on a response message. The address byte consists of 8 bits allowing for 256 addresses, but 16 addresses is the maximum normally used since some bits are used for other functions.

The control byte has 8 bits and contains command or control response information. The data field, on the other hand, can contain any number of bytes and is the user's total data transmission. The data field normally uses EBCDIC, ASCII, BCD, or straight binary code.

The check field follows the data and precedes the closing flag. It contains a cyclical redundancy check (CRC) character that detects and in some cases corrects errors during transmission. To review, CRC functions as follows: the transmitter sends its computation to the receiver, which compares the transmitted computation with its own calculation. If equal, the data is assumed to be error-free, and, if unequal, the receiver may not accept the transmission and normally requests retransmission.

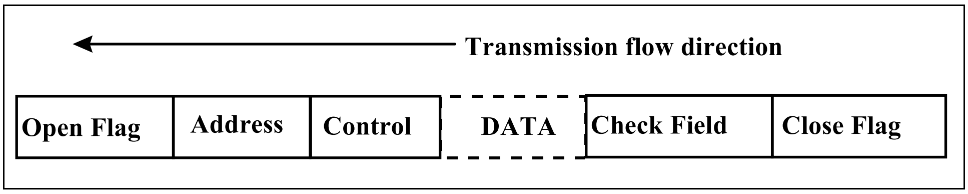


Figure 10-18. An HDLC Transmission String

The close flag is the final transmission byte and has the same bit configuration as the open flag. It terminates the transmission frame and begins the next if it is available.

The synchronous nature of this protocol forces the transmitter to have data ready in a buffer at the beginning of a transmission block. If it is not ready and the system fails to produce the data in time for transmission, the transmitter will run out of information to transmit. The HDLC protocol does not have an idle character within a block so the system must abort an entire transmission block when the transmitter runs out of data to send. The abort code is normally a sequence of eight 1s.

TCP/IP

Transmission Control Protocol/Internet Protocol (TCP/IP) originated with the UNIX operating system and it was selected for use on the Internet. Therefore, it has very high commercial use. There are several advantages of TCP/IP as it is an open standard and free. Almost all operating systems have TCP/IP protocol stack included in their system at no extra cost. It is the protocol language of the Internet and provides for a robust method of reliable data transmission and reception. When operating on a local area network, however, its disadvantages are that it requires considerable overhead, it has security holes, and it is not a real-time system model.

Local Area Networks

A local area network (LAN) is a user-owned and operated data transmission system operating within a building or set of buildings. LANs allow a great number and variety of machines and processes to exchange large amounts of information at high speed over a limited distance. LANs connect communicating devices, such as computers, programmable controllers, process controllers, terminals, printers, and mass storage units within a single process or manufacturing building or plant.

LANs allow neighboring computers to share data resources, hardware resources, and software. For example, in a typical manufacturing facility, the process control system computers and a central computer used by purchasing might be tied together to speed up the ordering of raw materials for the plant.

A typical LAN for an industrial facility is shown in Figure 10-19. It consists of three levels with three different types of networks. The highest level (Level I) is the information network used by groups like accounting and purchasing at the plant. This network has the highest speed communication (10 Mbps and higher) network because it must handle large amounts of data and information. The middle level (Level II) is used to perform the automation and control of the industrial plant. This network is generally slower (normally 100 Kbps to 5 Mbps) than the Level I network because the data rates required for process and machine control time periods are lower. The low-level bus (Level III) is used to connect programmable control and other controllers directly to the field devices, such as weight, flow, pressure, level transmitters, and switches.

LAN Topologies

There are three LAN topologies in common use: ring, bus, and star. The *ring* network topology shown in Figure 10-20 can operate using

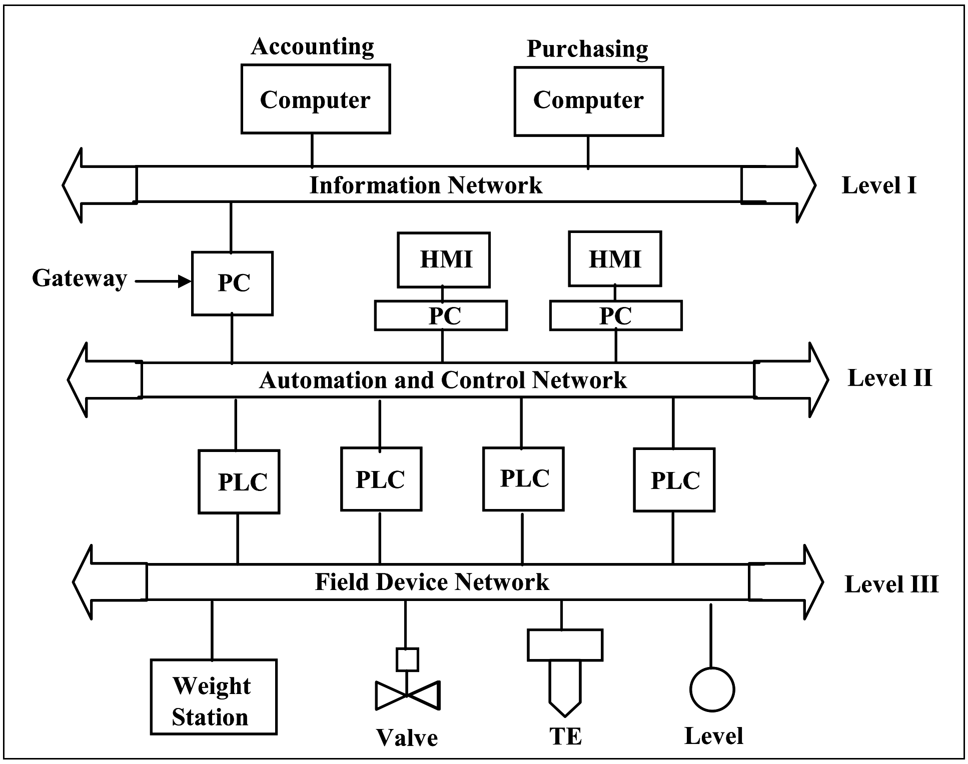


Figure 10-19. Typical Industrial Plant LANs

unidirectional transmission medium. However, most ring topology networks use bi-directional transmission allowing messages to flow in the most efficient manner. Each node decides whether to accept or pass on a message. This scheme is relatively easy to implement.

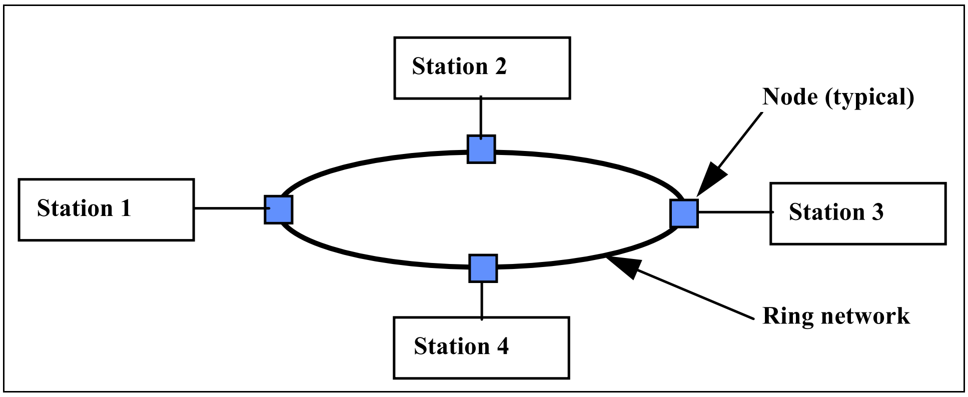


Figure 10-20. Ring-Type LAN

The *bus* network topology shown in Figure 10-21 requires a broadcast medium in which signals flow to all stations at all times. All the stations receive transmissions, even if they act only on some. The advantage of this scheme is that the stations connected to the bus perform no message routing because the bus is a broadcast type medium. This is the most common found in process control.

The *star* network shown in Figure 10-22 normally allows only one station to be in communication with the central station at one time. The central station may be allowed to transmit to several nodes at the same time. Routing messages is very easy because the central station has a unique hardware path to each node. System security is high since the central station controls access to the network. Another advantage is that priority status can be assigned to selected nodes in the network.

Communications Protocols

The software used to transfer messages over a network is known as network *protocols*. A communication *protocol* was defined earlier as the set of rules that govern data communications. Protocols ensure that devices communicating over data communications lines will be speaking the same language. One of the most important functions of a protocol is to govern access to the communications network. Failure to control access would result in chaos any time the traffic on the network rose above a certain minimum level.

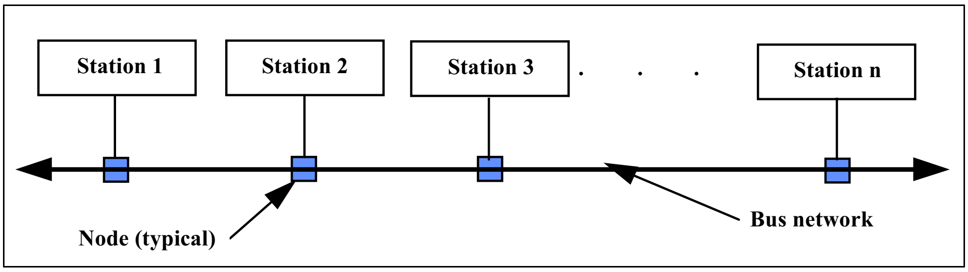


Figure 10-21. Bus-Type LAN

The standard LAN protocols are polling, token passing, and carrier sense multiple access/collision detect (CSMA/CD). In polling, a master station selects each of the other stations in turn and gives the station permission to communicate for a fixed period of time. The main disadvantage of polling is that each station must remain idle except when selected by the master. This type of protocol is best suited for bus- or star-type LAN topologies, but it is not normally used because stations must remain idle for a large percentage of the time.

The token passing protocol operates by passing a symbolic electronic token from one station to another in the network. Each station may hold

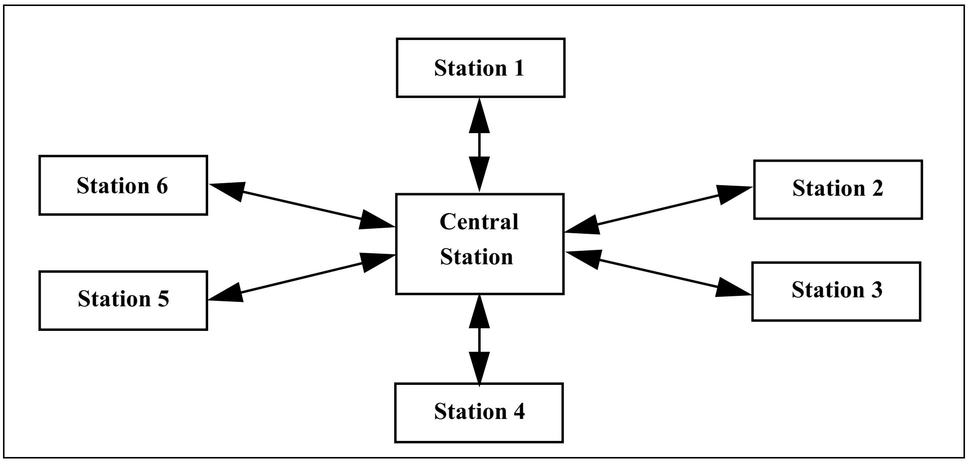


Figure 10-22. Star-Type LAN

the token for a predetermined length of time before passing it. The station that has the token controls the right to transmit to the network.

The CSMA/CD protocol allows stations to try to communicate whenever they need access. When a station has a message to send, it first listens to determine if anyone else is transmitting ("carrier sense"). When the station detects an idle channel, it transmits data. If two stations detect an idle channel and transmit simultaneously, a collision will occur. In this case, the "collision detect" part of the protocol informs both stations that the communication has failed; then both stations wait a random amount of time before attempting to retransmit.

ISO/OSI Communications Standard

Data communications between computer systems and networks is possible only if they adhere to some common set of rules for both hardware and software. A standard approach to network design or architecture that defines the relations between network services and functions is required in computer system design.

The International Standards Organization (ISO) recognized the need for standards to govern the information exchange between and within networks and across geographical boundaries. The standard, which has gained wide acceptance, is a seven-layer model for network architecture known as the ISO Model for Open System Interconnection or simply the OSI model.

The layered approach to network design comes from the design of operating systems. Due to the complexity of most computer operating systems, they are generally designed in sections, each of which contributes a certain function to the operating system. This method of design makes it easier for each section to be refined and redesigned to meet its functional purpose. Finally, all of the layers are integrated to provide a totally functioning operating system.

The same procedure can be used to design a network system. The OSI model specifies a hierarchy of independent layers that contain modules for performing defined functions. The OSI model has seven distinct layers at both the receiver and the transmitter, through which communications must pass, as shown in Figure 10-23.

The OSI model is not itself a communication network standard like Ethernet or Token Ring. Rather, the OSI model is a framework under which the various network standards can operate. The OSI model specifies the characteristics of various network standard, so it functions as a standard for data communications standards.

Although the OSI model contains seven layers, the bottom two layers have the most practical impact on PLC networks.

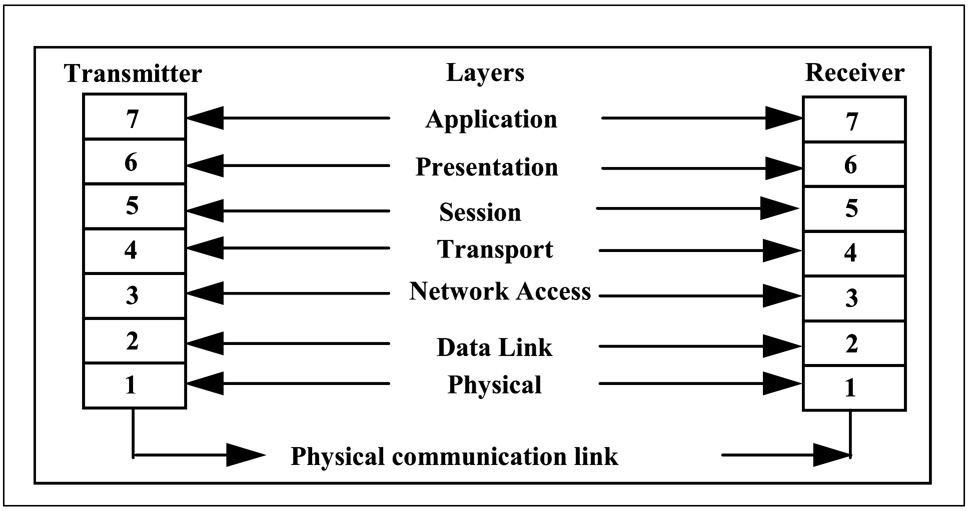


Figure 10-23. ISO/OSI Seven-Layer Communications Standard

1. The *physical* layer defines the electrical and mechanical requirements of interfacing to a physical medium for transmitting information. When used, this layer must include the software driver for each communications device and the hardware (e.g., interface equipment, connectors, modems, and communication cables).

2. The *data link* is used to establish an error-free communications link

between network stations over the physical channel. It formats messages for transmission, checks integrity of received data, controls access to and use of the station, and ensures the proper sequence of the transmitted information.

3. The *network control* layer is used to address messages, set up the path between stations, route messages across intervening stations to their destinations, and control the flow of data between stations.
4. The *transport* layer furnishes end-to-end control of a communication once the path has been established allowing the system to exchange data reliably and sequentially. This layer is normally beyond user control.
5. The *session* layer organizes the dialog of the communication and manages data exchange. This layer is also normally beyond user control, as are layers 6 and 7.
6. The *presentation* layer handles tasks related to data representation and code conversion.
7. The *application* layer handles tasks related to data transfer speed and integrity.

Outside the control of the ISO/OSI model are the communications applications processes. Usually, an application process is at the transmitter and another is at the receiver. Computer operations that require a user at a terminal or a piece of software performing instructions are examples of application processes.

Serial Communications Standards

Serial hardware standards are required for the physical and electrical compatibility among devices that communicate on serial networks. Ports and connectors must match in size, shape, and pin configuration. Transmitting and receiving circuits must agree on how signals are generated and interpreted. The so-called "physical layer" communications requirements are handled by the serial interface standards such as TIA/EIA-232-F, TIA/EIA-422, and TIA/EIA-485-A between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE).

The Electronics Industry Association (EIA) is responsible for setting the physical and electrical standards in the U.S. TIA stands for Telecommunications Industry Association. There are many different communication standards because many are needed to serve different purposes and functions. The DTE describes the end device that originates data to or receives data from, data communications equipment (see Figure

10-24). Some examples of DTE devices are programmable controllers, PCs, and printers, where the data is stored and operated on in the equipment and intended for transmission to another device. The DCE is the device closest to the communication line or bus (media) as shown in Figure 10-24. Typical examples of a DCE device are modems and line drivers.

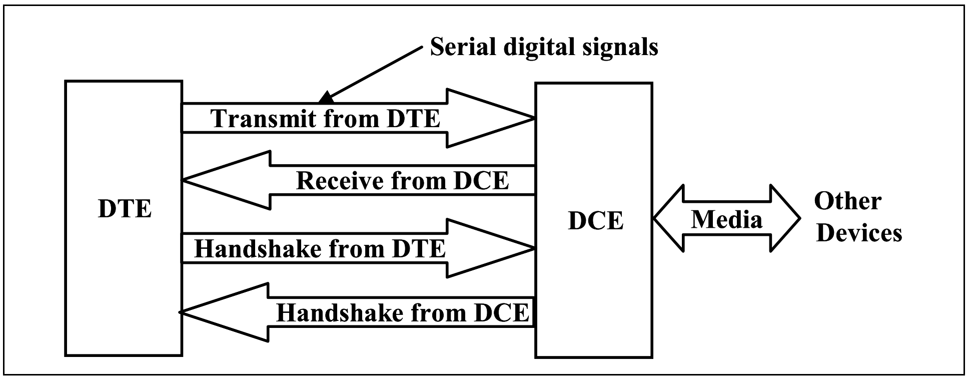


Figure 10-24. Block Diagram of DTE and DCE Devices

The term "serial interface" is often used to mean a specific physical and electrical standard that applies to a given piece of equipment. For example, if we say that a programmable controller has a "TIA/EIA232-F serial interface," we mean that the interface is the circuitry that allows the PLC to communicate with another device on a communication bus or directly.

TIA/EIA-232-F Standard

The TIA/EIA-232-F standard is one of the most common serial interfaces in use today. This standard defines a number of physical and electrical characteristics of the interface circuit and the connector. This serial standard specifies things like: connector pin-outs, line characteristics, and input/output impedances. Most TIA/EIA-232-F connectors have 25 connection pins in a "D" shape plug. The reason for this "D" shape is that it was the connector used on Bell telecommunication modems in the late 1960s. Each pin, or data line, is assigned a special purpose. In this standard or the data lines, a logical 0 is a voltage greater than +3 VDC; logical 1 is a voltage less than -3 VDC. There is a neutral range between +3 V and –3 V where there is no logic value (not 1 or 0). The teletypewriter origin of this signal can be seen since the most positive voltage signal is logical 0 (or Space on a teletypewriter) and the most negative signal is a logical 1 (or Mark on a teletypewriter). Teletypewriter circuits used an "off" condition (Space) with no current and the "on" condition (Mark) with a negative voltage in relation to earth ground.

Data transfer rates of up to 25 Kbps are possible, but cabling between devices is limited to 50 feet. TIA/EIA-232 is a master-slave serial interface, where there is a primary station in control of a secondary station or stations. The term "unbalanced link" is also used to mean TIA/EIA-232 communications.

TIA/EIA-422 Standard

The TIA/EIA-422 standard specifies input and output impedances, line lengths, rise and fall times, signaling speeds, and voltages levels. The standard specifies a ±200 mV to ±6 V signal, with the most positive voltage the logical 0 state.

This interface standard, unlike TIA/EIA232, creates balanced link communications. It uses two data lines or two wires, line A and line B. Each data line has the same impedance to signal ground (normally very high). The receiver electronic circuitry measures the difference in voltage between Line A and Line B. The receiver cannot measure the actual voltage on Lines A or B to signal ground only the voltage difference between the lines. A logical 1 is determined when line A is negative with respect to Line B, and a logical 0 is determined when line A is positive with respect to line B. The signal will have a minimum level of ±200 mV and a maximum of ±6 V.

TIA/EIA-422 is much faster than TIA/EIA-232-F and transmission rates up to 10 Mbps are possible. Wiring or cabling distances are much longer than with TIA/EIA-232, up to 4000 feet. This is because the voltage difference, not its specific value, determines whether a signal is 1 or 0.

This communication standard is specified for one transmitter and up to 16 receivers and is used for point-to-point transmission using four wires (2 pair) for duplex transmission. The one transmitter can drive 16 devices but the standard does not allow for contention of the one master. So if more than one transmitter or master is needed in a system the next communication standard, TIA/EIA-485 can be used.

TIA/EIA-485 Standard

The TIA/EIA-485 interface standard is based on impedance sensing rather than voltage sensing. Like the TIA/EIA-422 interface, TIA/EIA-485 can be run for distances up to 4000 feet and it supports up to 32 device connections. This standard was formulated to describe a transmitter-receiver combination capable of multipoint operation. It allows for any combination of up to 32 transmitters and receivers on the same two-wire line. It calls for a balanced line for data communication much like TIA/EIA-422. However, TIA/EIA-422 permits only one transmitter and all other devices must be receivers, TIA/EIA-485 uses tri-state logic and connects the transmitter and receiver to the same wire. This is a communication bus network, so that the station addressing must be handled in the driver software in the system.

The wiring guidelines under TIA/EIA-485 are very strict. For example, all connections are polarized and cables must be kept a specific minimum distance from power cables and wires. Where power wires cross a TIA/EIA-485 cable, the crossing must be made at 90° angles. TIA/EIA-485 standard has greater immunity to noise interference than either TIA/EIA-232 or TIA/EIA-422.

A TIA/EIA-485 network is very inexpensive to implement. The interface communication circuit cards are similar in cost to an Ethernet interface card. This serial interface system is used in many industrial applications including PLC and instrumentation systems.

Industrial Control Networks

There are two types of control networks used in the PLC industry: proprietary and open. The current trend is to use open communications in process control and programmable controller applications. Open networks will be discussed in the greatest detail, but first, we will discuss Allen-Bradley's proprietary networks.

Allen-Bradley Proprietary Control Networks

There are three proprietary control networks used in Allen-Bradley PLC systems: DataHighway (DH), DataHighway Plus (DH+) and DH485. A typical layout of DH and DH+ networks is shown in Figure 10-25. The DH network is used for peer-to-peer communications between PLC processors and other devices such as PCs and intelligent control devices. It forms a LAN of up to 255 nodes where the nodes are addressed in octal up to 377. The last octal node address of 377 is not used because it is all 1s, so only 255 stations are available not 256. The DH network operates at a transmission speed of 57.6 Kbps and it can have a transmission line length of up to 10,000 ft. (3.048 Km) with drop lines of no more than 100 ft. off the main line as shown in Figure 10-25.

The DH network uses a modified token-passing scheme known as floating master that provides a coded scheme for the devices that need to take control based on their functional priority. Generally, the PLC controllers are given the highest priority on the network because they control the critical operations of the process or machines under control.

The DH+ network uses a token-passing method and allows up to 64 nodes (0 to 77 Octal) on the bus with a maximum allowed transmission rate of 230 Kbps. The network is mainly used processor-to-processor connection for Allen-Bradley PLC2, 3, and 5 series PLCs. It also allows for the use of a PC as a programming terminal or a HMI device by adding a communications circuit card and the appropriate software package to the PC. The main differences between the DH and DH+ networks are that the DH+ network is optimized for the lesser number of nodes and it allows for online programming of the PLC processors. The DH network uses ±14-V peak-to-peak signaling, while the DH+ network uses ±7-V peak-to-peak signaling. They both use bi-phase Frequency Shift Keying (FSK) communications signal transmission (i.e., Manchester system).

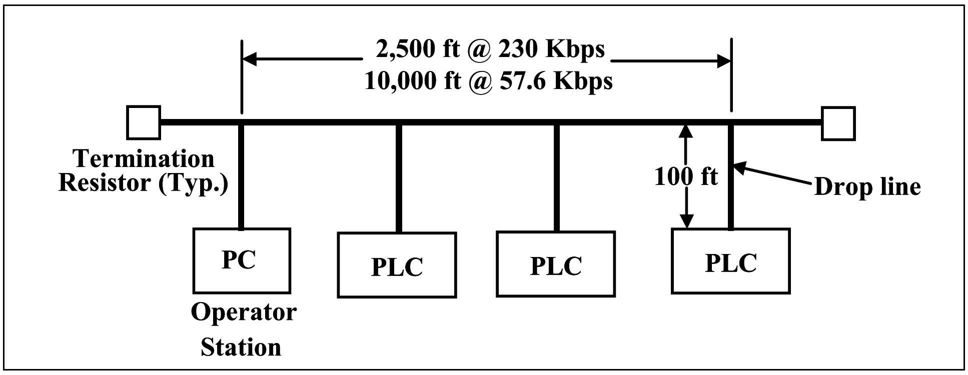


Figure 10-25. Allen-Bradley DH/DH+ Topology

The DH485 network uses the TIA/EIA-485 standard and it supports up to 32 nodes. It has a data transmission rate of up to 115 Kbps, although the TIA/EIA-485 maximum is 10 Mbps for up to 50 feet of transmission line. The TIA/EIA-485 standard refers to the electrical characteristics of the transmission media rather than the transmission signal protocol. Allen-Bradley uses its DF1 asynchronous protocol, which uses byte size ASCII control characters similar to the IBM BISYNC method. A Data Link Escape (DLE) character precedes all control characters in a message string. Some of the other DF1 control characters include DLE STX (start of text frame), DLE ETX (end of text frame), DLE ACK (positive acknowledgment), DLE NAK (negative acknowledgment), and DLE DLE (data that may have embedded control codes). In this DF1 system, full duplex consists of two 2-way paths and half duplex consists of one 2-way path.

Open Communication Control Networks

There are numerous open communication control networks used in PLC and control applications. The industrial communication networks to be discussed are Modbus Remote Terminal (RT), ControlNet, Ethernet, and Profibus.

Modbus RT Network

The Modbus Remote Terminal (RT) network was designed by Modicon for its PLC systems. It has become a de facto open communications standard used by most other PLC manufacturers in addition to their own proprietary method. Modbus RT is a master/slave communications network where there is one master and one or more slaves. It uses a query-response protocol where only the master can query and the slave or slaves cannot query but only respond to a query. This method is similar to the standard polling method. Many PLC manufacturers use Modbus RT as an accepted communications method and the protocol can perform the basic host to PLC processor communications. It is not used on field devices, but from processors to a host and has two modes, an ASCII mode and Remote Terminal Unit (RTU) mode.

The RTU mode has higher information density and is more widely used. There are 246 addresses (1-247) available; address 0 is an all-stations broadcast address. The communication rate depends on the devices attached to the network but a speed of 19.2 Kbps is typical for distances up to 4000 ft. However, higher communications data speeds are achievable for shorter distances.

ControlNet Network

ControlNet is an open network that supports both field device and PLC communications at the same time. Allen-Bradley developed this network, but it has been controlled by the ControlNet International organization of vendors and users since 1997. It uses the product-consumer type of communications and it also supports multiple masters, peer-to-peer, and broadcast type of communications.

ControlNet uses a communication data rate of 5 Mbps, it provides deterministic communications, and it is used for PLC controller-to-PLC controller, controller-to-device, and controller-to-external systems. It provides seamless integration with many other standard networks and by using the correct communication module it interfaces with Foundation Fieldbus. It has been approved for installation in intrinsically safe environments.

Ethernet Control Network

Ethernet is widely used in the commercial and Internet world but due to its nondeterministic nature, it was not used in the control industries until recently. However, with the introduction of higher speed Ethernet networks and the lower cost of Ethernet hardware, the main disadvantages to using Ethernet have disappeared. There are many advantages for high-speed Ethernet. One advantage is that Ethernet components are commodity items and therefore very low cost. Ethernet can operate as a communications medium regardless of the PLC communications protocol by encapsulating standard PLC protocols such as Modbus or Profibus within the Ethernet TCP/IP package. The multi-drop nature of an Ethernet network leads to an easy expansion method for PLC applications. A final advantage is that the PCs used for data collection and displays are in many cases already on plant floor Ethernet networks.

Profibus Control Network

The Profibus network of protocols includes three different communication protocols: Profibus Fieldbus Messaging Specification (FMS) is a general-purpose system for peer-to-peer communications; Profibus DP is a high-speed data network for factory automation; and Profibus PA is a communications system for process automation applications. The physical layer of the Profibus network has three transmission methods: TIA/EIA-485 two-wire copper cable; fiber optics cable if EMC protection is needed; and IEC 61158-2, a two-wire copper cable with provision for power on the transmission line. Profibus differentiates between master and slave devices for bus access and uses a method similar to floating master to determine control of the bus. The only passing of control of the bus is between master units. The slaves are always polled, whether there is one master or many masters. Profibus supports bus, tree, and star LAN topologies, but tree arrangement is the preferred one.

EXERCISES

10.1 What are the three basic components of a communications system?

10.2 Explain the operation of unidirectional, half-duplex, and full-duplex communications.

10.3 Compare and discuss serial and parallel data transmission. Give the advantages of each method and some typical applications encountered in computer systems.

10.4 What are the three common methods of signal multiplexing encountered in communications systems?

10.5 What are the four common data transmission error-checking methods used in PLC communications systems?

10.6 Calculate the LRC/VRC odd parity characters required for the message: M1 ON.

10.7 Explain the difference between asynchronous and synchronous data transmission.

10.8 List three common serial data protocols used in communications systems.

10.9 Assemble the bit streams required to send the message RUN using 8-bit even parity ASCII code and BISYNC protocol. Assume that the header is given by 00000010, which represents station 2, and omit the check character at the end of the transmission.

10.10 What are the three most common LAN topologies?

10.11 List the seven layers of the ISO/OSI communications standard.

10.12 What is an advantage for using an Ethernet network in PLC control application?

10.13 What is a Data Terminal Equipment (DTE) device and give several type examples of DTE devices?

10.14 List several examples of open communication networks used in the control industries.

BIBLIOGRAPHY

1. Marshall, Perry S. and John S. Rinaldi. *Industrial Ethernet, 2nd edition*, ISA – The Instrumentation, Systems, and Automation Society, 2005.
2. Mitchell, Ron. *Profibus: A Pocket Guide*, ISA – The Instrumentation, Systems, and Automation Society, 2004.
3. Thompson, L. M., *Industrial Data Communications Fundamentals and Applications*, 3rd edition, ISA – The Instrumentation, Systems, and Automation Society, 2002.
4. Ivens, K.; Gardinier, K., *Windows 2000: The Complete Reference*, Osborne/ McGraw-Hill, 2000.
5. Harrington, J. L., *Ethernet Networking Clearly Explained*, Academic Press, 1999.
6. *Reference Manual Data Highway/Data Highway Plus Protocol and Command Set*, Allen-Bradley Company, Inc., 1987.

11 System Design and Applications

Introduction

The programmable controller–based control system offers a wide variety of configurations and capabilities. These range from a single machine or process control to an entire industrial plant control and monitoring system. After the decision has been made to use a programmable controller in a control application, the design engineer must complete the system design.

In this chapter we will discuss two typical PLC applications encountered in the control industries. First, we will cover the basic elements used in the design of a PLC application, such as process and mechanical control diagrams, process and machine control descriptions, sizing and selection of a PLC system, control system drawings, I/O wiring diagrams, and control system programming.

Process and Mechanical Control Diagrams

The design of a programmable controller–based control system requires process control diagram or mechanical equipment layout drawings. In the control and instrumentation field, these drawings are called Piping and Instrument Drawings (P&ID) in process control applications or they are called Mechanical Flow Diagrams (MFD) in mechanical control applications. These drawings show the process and/or mechanical equipment to be controlled and the instrumentation used in the control of the process or machine.

A standard set of symbols is used to prepare the P&IDs and MFDs. The symbols used in these drawings are generally based on the standard ISA-

5.1-1984 - (R1992) Instrumentation Symbols and Identification. These drawings show the interconnection of equipment and the instrumentation and PLC I/O points used to control the process. A typical example of a P&ID with PLC and instrumentation symbols is shown in Figure 11-1.

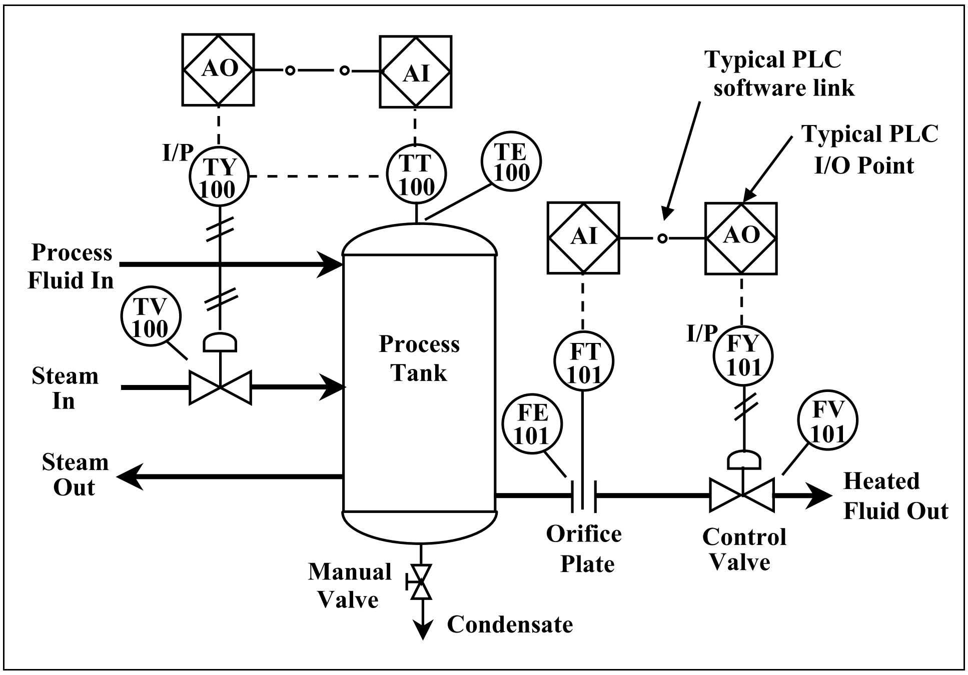


Figure 11-1. Typical Process and Instrument Diagram

In standard P&IDs, the process flow lines, such as process fluid flow and steam flow, are shown as heavy solid lines. The instrumentation signal lines are shown in a way that indicates whether they are pneumatic or electric. A crosshatched line is used for pneumatic lines, for example, a 3–15 psi signal. A dashed line normally represents an electric signal, usually 4–20 mA DC current signal.

A balloon symbol with an enclosed two- to four-letter code is used to represent the instruments associated with the process control loops. For example, the balloon in Figure 11-1 with TT-100 enclosed is a temperature transmitter and the balloon with TY-100 enclosed and the letters I/P on the outside of the balloon is a current (I) to pneumatic (P) converter. Generally, a fixed number is assigned to each control loop; combining the letter code and number into an instrument tag number labels the specific device in the loop.

The square box with a diamond inside is the symbol used to show a PLC or computer input or output point on P&ID, MFD, or other control diagrams. The letters "AO" inside the PLC I/O symbol indicate an analog output and the letters "AI" represent an analog input. The dashed line with very small circles between each dash that is drawn between PLC I/O symbols is the symbol used to indicate a software link inside a PLC or computer.

Special items such as control valves and in-line instruments (for instance, orifice plates) have special symbols, as shown in the P&ID in Figure 11-1. Refer to ISA-5.1-1984 - (R1992) for a more detailed discussion of instrument symbols.

The control system design engineer will normally reduce the P&ID or MFD to a simplified control diagram that shows only the equipment and instrumentation controlled or measured by the programmable controller. For example, Figure 10-2 shows a simplified MFD for a PLC-controlled conveyor system. In this drawing the letters "DO" inside the PLC I/O symbol indicate a digital output used by the PLC to turn on the conveyor drive motors. The letters "DI" inside the PLC I/O symbol indicate a digital input that is connected from the PLC input modules or points to start and stop pushbuttons mounted close to the conveyors.

These simplified control diagrams are used to show the status of the process in each step or state. The designer of the system to aid in the programming also uses them. These drawings can also be used in operator instruction manuals to aid plant or operator personnel in understanding the operation of the control system.

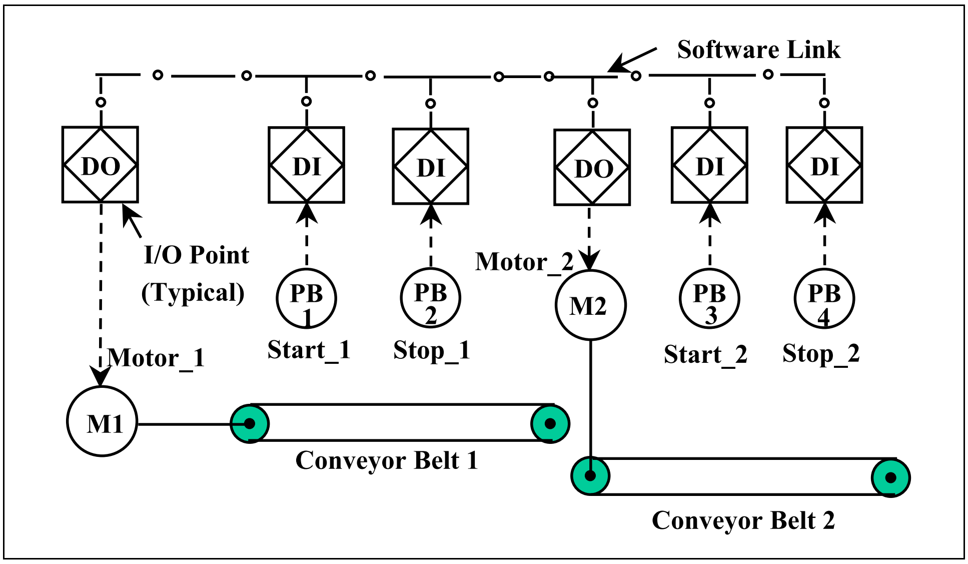


Figure 11-2. Simplified MFD for a PLC-Controlled Conveyor System

Process and Machine Control Descriptions

Writing the process or machine control description document is probably the most important step in the design process because it conveys in simple language the purpose and the steps of the process or machine. It is also important because in most applications it is the main vehicle of communication between the user or customer and the system designer. This document should have two main parts: one is a clear and concise control system definition section and the second is a well-written and thought-out control strategy section.

Control System Definition

The control system engineer in conjunction with the user should begin the design process by defining the control tasks for a process or machine. This definition provides a basis for determining the PLC operations that must be performed and the results expected. Those who are most familiar with the operation of the process or machine should write the control definition. This procedure will help to reduce errors in the design due to misunderstanding of the process or machine.

The control task definition must involve people from all levels and parts of an operation. Personnel from each department involved with the project must be consulted to decide what information is required or has to be provided so that everyone clearly understands the project. For example, in a manufacturing plant automation project where material is to be retrieved from the warehouse and then sent to the automatic packing area after the parts are manufactured, people from both the warehouse and the packaging area must be consulted during this control system definition phase. Plant management should also be contacted if there are data reporting requirements or project cost issues.

If the control project is an upgrade from a manual or relay-based control system to a PLC, each step of the manufacturing process should be reviewed to determine what improvements if any are possible. Although, in most cases relay logic can be implemented directly by a PLC, it is recommended, when possible, to redesign the manufacturing process to meet current requirements and take advantage of the increased capabilities of PLCs.

Control Strategy

Once the control tasks have been defined, the planning for its solution can start. This solution normally involves deciding on the steps in the process that must take place in the control program to produce the required outputs to control the system. This phase of PLC design is called the development of a control strategy or algorithm. The term algorithm may sound strange to some, but every one of us follows algorithms to accomplish tasks daily. For example, the procedure that a person follows to go from home to work or school each day is an algorithm: the person leaves the house, gets into the automobile, starts the engine, and so on; until finally after a finite number of steps, the final destination is reached. The strategy executed for a control task using a PLC closely follows the development of the algorithm. In most cases, it is possible to develop an algorithm to solve a control problem. If developing the algorithm becomes difficult, it may be that further definition of the problem is required. In this case, a return to the problem definition phase may be required.

A basic rule in defining the control program strategy is to think first and program later. A programmer should always consider alternative approaches to solving a control problem. The designer should allow time to polish the solution algorithm and consider adverse consequences of any solution before attempting to program a control function. Using this procedure will shorten the programming cycle, reduce troubleshooting time, accelerate the start-up time, and permit the focusing of attention on the control system design during the project.

The next phase of the design process is to size the PLC system to meet the input and output requirements of the project.

Sizing and Selection of a PLC System

The sizing of a PLC system consists of estimating the number of input/output (I/O) modules required to control the process and calculating the size of the internal processor memory needed. This selection process also consists of choosing the correct programming language and peripheral equipment required for the control application.

I/O Sizing

Many different types of I/O modules may be needed for a given PLC application. Limit switches, pushbuttons, selector switches, motor controls, solenoids, and pilot lights may require either AC or DC modules of different voltage levels. Solid-state displays and some electronic instrumentation may require +5 VDC logic interface modules. Process instrumentation for measurement of level, flow, temperature, or pressure may require analog-to-digital (A/D) conversion interfaces. Incremental encoders and stepping motors might need special-purpose I/O modules.

The interface to these I/O devices can be done with external electronic equipment to condition the signals that would increase overall equipment cost for a system. Therefore, a PLC should be selected with the correct I/O modules to match the type of field devices used in the process. This design process is called I/O sizing.

Each programmable controller has a maximum number of input and output devices that can be monitored or controlled. Most PLC systems have I/O capacities ranging from a few to over 1024. These capacities can be divided into four PLC-size categories: micro, small, medium, and large. Micro PLCs have a limited number of built-in I/O circuits, normally 32 points or less.

A small PLC system would range from 32 to 256 I/O points, a medium-size PLC encompasses 256 to 1024 I/O, and a large PLC system has 1024 and higher I/O points. To determine the PLC system size required, simply add up the number of field and control panel devices and compare the total to the above classifications. The system designer must also define the type and number of I/O because some PLC systems are constrained as to the mixture that can be interfaced to a given I/O system.

If modular I/O systems are being used, input and output point totals can then be used to determine the number and type of I/O modules required. Each type of module can interface a certain number of I/O, such as 4, 8, 16, or 32. Divide the number of inputs or outputs by the number of I/O points per module and round up to the nearest whole number. This calculation must be performed for each type of I/O module (i.e., discrete, pulse, analog, etc.) After defining the I/O requirements, the designer must also consider spares and future system expansions. Most programmable controller users find that 10% to 20% spare capacity is enough for normal system growth.

To illustrate how to calculate the number of I/O modules, let's consider Example 11-1.

Memory Sizing

The amount of memory required for an application is primarily a function of control program complexity and the number of I/O points in the system. The most precise method of determining memory size is to write out the control program and count the number of instructions used in the program. Then, multiply this count by the number of words used per instruction. This number can be obtained from the PLC programming manual. Also consult the programming manual on the amount of memory used by executive programs and the processor overhead. The problem with this method is that, in a large system, the system programming sometimes takes months to complete, and the system must be designed and purchased in advance. A shortcut method consists of multiplying the

EXAMPLE 11-1

Problem: Calculate the number of discrete 16-point I/O modules required for a PLC control application with 136 discrete inputs and 120 discrete outputs. Assume that 10% spare capacity is required for the system.

Solution: To calculate the number of discrete I/O modules, we use the following equation: Number of modules required (N) =

(I/O points + 10% x points)/(16 points/module)

1) For the136 discrete inputs:

N = [(136 + 136 x (0.10))/16] modules = (136 + 13.6)/16 modules = 9.35 modules

Or ten, 16-point discrete input modules are required.

2) For the 120 discrete outputs:

N = [(120+ 120 x (0.10))/16] modules = (120 +12)/16 modules = 8.25 modules

Or nine, 16-point output modules are required.

number of I/O points by 10 to obtain a rough estimate of the memory required.

An illustration of memory sizing using this simpler method is given in Example 11-2.

Selecting Programming Language

As mentioned earlier, the five basic types of programming languages available in programmable controller systems are Ladder Diagram (LD), Structured Text (ST), Instruction List (IL), Function Block Diagram (FBD), and Sequential Function Chart (SFC). The type selected depends on the size and complexity of the control system and the background of the control system programmer and operators. Most PLCs offer the basic ladder logic instructions plus a combination of the other types of languages. The most common type of language selected is ladder diagram since this covers the basic ladder logic instructions and some data transfer and manipulation functions.

EXAMPLE 11-2

Problem: Estimate the memory size required for the PLC application shown in Figure 11-3.

Solution: To calculate the memory size, we first need to calculate the number of I/O points in the system in Figure 11-3.

Remote Area 1: I/O points = 70 + 35 + 6 = 111

Remote Area 2: I/O points = 95 + 50 + 10 = 155

Main Process Area: I/O points = 300 + 156 + 32 + 5 = 493

Total system I/O points = 111 + 155 + 493 = 759

Therefore, memory size = 10 x 759 = 7590 or 8K.

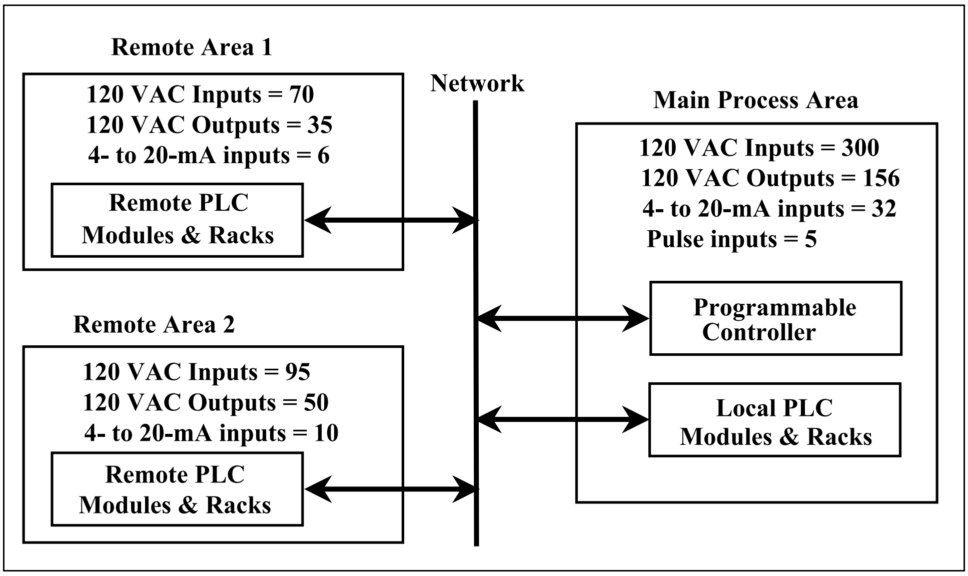


Figure 11-3. Typical Process Plant PLC System

Peripheral Requirements

The term "peripheral" refers to the other equipment in the programmable controller system that is not directly connected to field I/O devices but increases the capabilities of the system. The most common peripheral is the programming device. This is generally available in three formats: a compact portable programming device from the PLC manufacturer, a laptop PC with PLC software installed, or a desktop PC with programming software included. The compact portable programmer normally has a small, limited function keypad and a seven-segment LED and can handle only one or two logic rungs at a time. It is normally used on micro or small PLC systems or for minor field changes to larger systems. The laptop PC-based programming device is normally used for field start-up and troubleshooting. The standard PC-based programmer is normally used in a lab or office environment to perform the development programming on PLC systems.

Another common peripheral used in PLC systems is a mass storage unit. This unit is used to store the control program on magnetic media so that, in the event of a program loss, the backup program can be reloaded into the controller memory. If a personal computer is used in the PLC system, the program can also be saved on a floppy disk or a hard disk for future use.

For hard-copy printouts of the control programs, a printer can be interfaced with the programming device, normally a PC, to obtain a program listing.

It is important for a complete system design that the peripheral equipment is available to back up and document the control program during start-up and field-testing, because reprogramming and documentation can be expensive and time consuming.

Other common peripherals are PROM programmers, process I/O simulators, and communications modules. The PROM programmers are used to write and save control programs on the PROM chips used in some controllers. The I/O process simulators are useful and cost-saving devices for large and complex systems that can be fully tested before installation and start-up in the field. The communication peripherals are used to communicate between the programmable controllers and the plant or personal computers and other controllers in a system via communication networks.

The type of operator interface to be used is one of the most important considerations in a PLC system design. There are four main options for operator interfaces: (1) hard-wired local and main control panels; (2) Human Machine Interface (HMI) software run on a PC; (3) intelligent peripheral devices, such as a touchscreen operator interface; and (4) industrial PC with function keys and HMI software. The system designer might also select a combination of several operator interface methods to implement a control system, such as hard-wired local panels and a HMI on a PC mounted in a central control room.

Control System Diagram

An important phase of the design process is drawing a control system diagram. This system drawing is used to give an overview of the system component interconnections and the communication cable layout. This drawing is also useful in identifying all the interface cables and components by model number.

A PLC system drawing of a simple stand-alone system is shown in Figure 11-4. This system consists of a programmable controller, a PC with HMI graphics and PLC programming software installed, a printer with interface cable, a communications cable, a rack-mounted DC power supply, an 8-module I/O equipment rack, and the associated input/output modules.

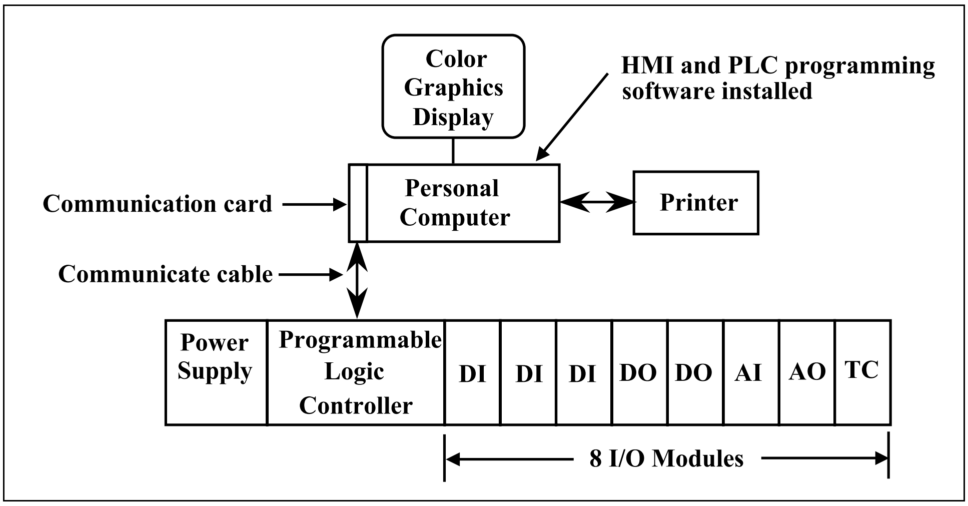


Figure 11-4. Typical PLC System Diagram

I/O Wiring Diagrams

The next step in the design process is drawing the input and output (I/O) wiring diagrams for the project. A typical discrete output module-wiring diagram is given in Figure 11-5. This drawing shows the wiring of process control equipment, such as heaters, pumps, and motors, to an AC output module. The wiring terminal strip in the programmable controller equipment cabinet is designated by TB-1, and a field junction box terminal strip is designated by JB-1 in the example. Field junction boxes are used extensively in process control applications to simplify field installation and maintenance of instrumentation. Field wiring is normally indicated on wiring diagrams by a dashed line, as shown in Figure 11-5. The wire numbers on the wiring diagram are used in the installation and maintenance of the system. The PLC output addresses are given on the left-hand side of the wiring diagram to aid the engineer or technician in the start-up or troubleshooting of the control system.

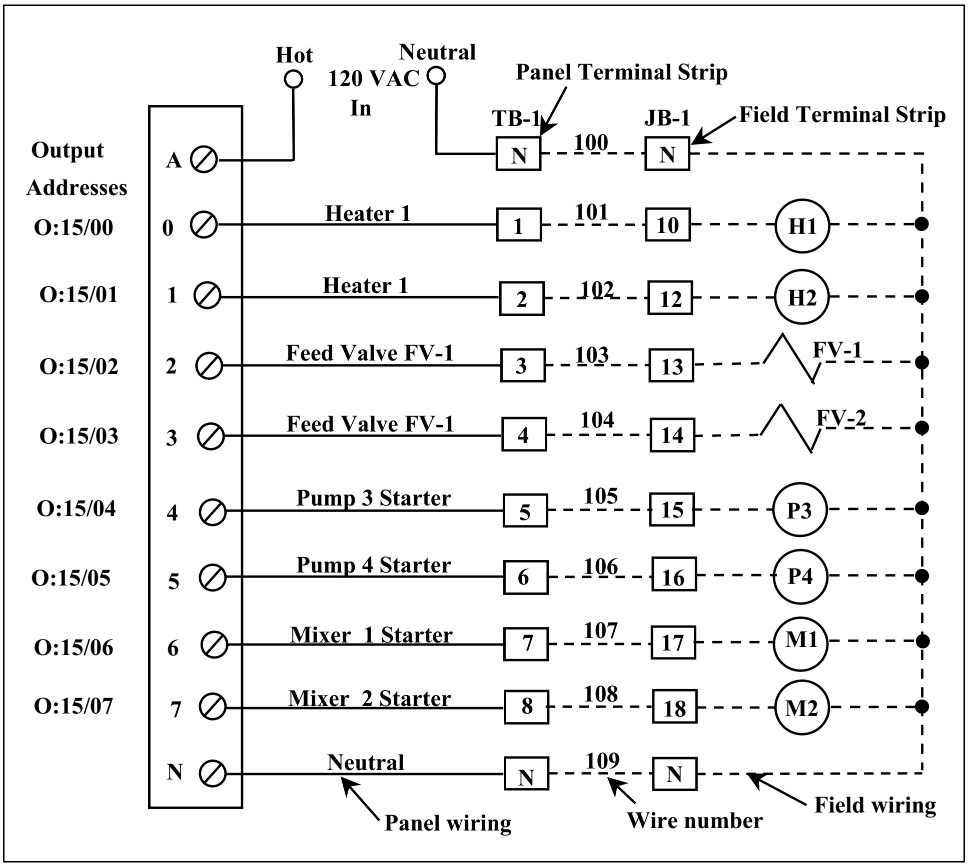


Figure 11-5. Typical Discrete Output Module Wiring Diagram

Control System Programming

The final phase of the design process is programming the PLC system. A design engineer, plant operations personnel, maintenance, or a control system integrator can do the system programming. The selection of programming language type should usually be left to plant operations or maintenance personnel since they will normally have to maintain the software after the system is installed.

A programmable controller is a versatile device but it can only do what is instructed to do by the software. It receives all of its instructions from the control software, the set of instructions created by the system programmer. There are numerous ways to approach and solve problems, but if the application is approached in a systematic way, the probability of success is high. PLC control projects fall into two categories, one is a new system and the other is an upgrade of an existing system using relay and/or analog loop controllers. In both new and upgrade applications, understanding the process or machine operation is the first step in systematic programming of the PLC to solve the control algorithms. In the upgrade project, the user normally completely understands the operation of the machine or process and what needs to be controlled. The relay ladder diagrams for the existing system can be translated almost directly into a PLC ladder program because these logic diagrams already define the sequence of logic for the new system. However, process or machine control description should be written, reviewed, and then followed carefully to avoid possible programming errors and to maintain good documentation. New applications are generally more difficult and require a detailed specification and control description that is written by the person or persons who will design, program, and install the system with extensive input from plant personnel.

We are now ready to cover two typical PLC-based control system applications, a natural gas dehydration system and a two-stage alternating pump system using the design steps discussed in this section. The first application to be discussed is the natural gas dehydration system.

Natural Gas Dehydration Application

The first application to be discussed is the natural gas dehydration process shown in Figure 11-6. This dehydration process removes excessive moisture from natural gas by using small beads in the process tower to absorb the moisture from the gas.

In this process, the differential pressure switches (PSHL-1 and PSHL-2) are used to detect both high and low pressure across the dehydration tanks. The air-operated valves (AOVs) are used to control the gas flow to the two towers shown. A typical diagram of the AOVs is shown in Figure 11-7. An electrically operated solenoid valve is used to supply instrument air at 80 psi to activate the control valves and the valves have limit switches to indicate if the valves are open or closed.

The programmable controller uses these limit switches to determine the status or state of the valves. In some applications, only one limit switch is used and the programmable controller can assume the opposite state (open or closed) if a single switch is used. However, a more reliable control is obtained if two switches are used. For example, if a valve fails to completely open or close on a given operation, the programmable controller is able to detect this failure and then alert the operator.

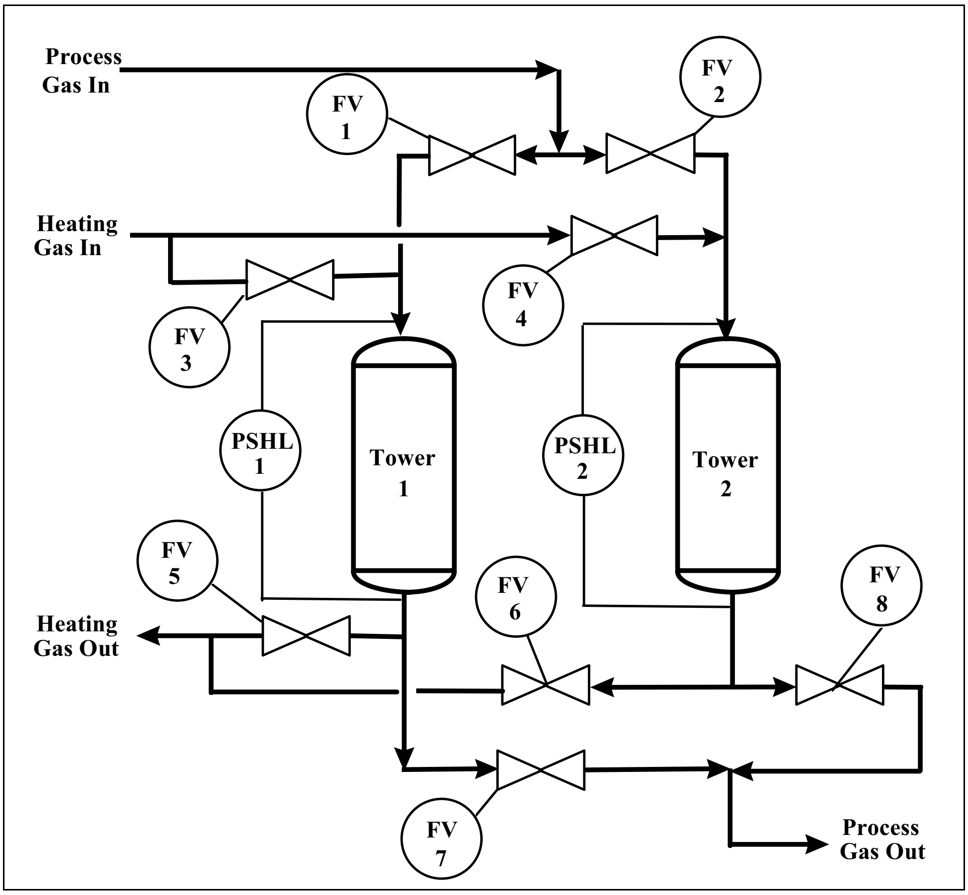


Figure 11-6. Simplified Dehydration Process Flow Diagram

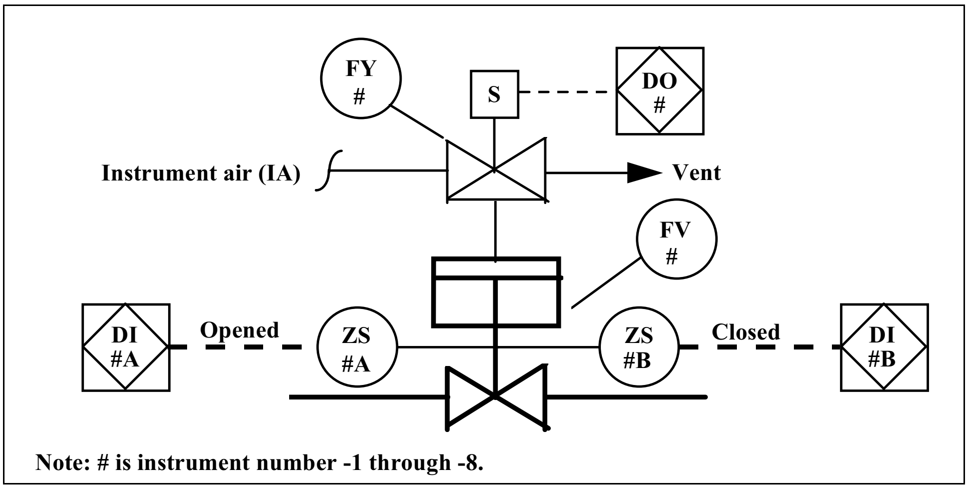


Figure 11-7. Instrumentation Design Detail for Air-Operated Valve

Since we have the simplified process flow diagram of Figure 11-6, we can use it to show the valve and/or equipment position for each state of the process in the process control description. The next step in the design process is to write a preliminary process control description for review.

Dehydration Process Control Description

The two process towers (towers 1 and 2) are used to remove moisture from natural gas. Generally, one tower is *in service* (i.e., removing moisture from the process gas) and the other tower in being dried out or *in regeneration*.

The automatic steps of the process are as follows:

1. We need to assume that the control system has been placed in automatic so that the PLC can control the process based on the field inputs.
2. If the differential pressure across tower 1 becomes high, as indicated by the differential pressure switch (PSH-1), the programmable logic controller would place tower 1 in the regeneration mode by opening valves FV-3 and FV-5 and closing FV-1 and FV-7.
3. At the same time, if the differential pressure across tower 2 is low, tower 2 will be placed in service by opening the gas valves FV-2 and FV-8 and control system will close the heating cycle valves FV- 4 and FV-6 for tower 2.
4. If later, the pressure across tower 2 becomes high and the differential pressure across tower 1 returns to low, the control system will place tower 1 in service and tower 2 into regeneration. This cycle will continue until the operator elects to stop the process.

After this preliminary control description is written, the designer is ready to size and select a programmable controller system.

Sizing and Selecting the PLC System

The first step is to decide on the human machine interface to use, such as local control panel or PC-based HMI. In this application, the dehydration process is a part of an entire natural gas processing plant and plant operations personnel have decided to use a PC-based HMI for operator interface. The graphics screen for tower 1 in service and tower 2 in regeneration for the dehydration process is shown in Figure 11-8.

The operator uses the computer mouse to select the mode of operation required for the process. For example, if the pressure across Tank 1 is low (i.e., the low pressure (Low Press) icon box on the HMI display is highlighted), the operator can click on the In Service (In Serv.) button and place Tank 1 in-service to remove moisture from the natural gas.

The next step in the design process is to calculate the number of input and output modules required. Using the dehydration flow diagram in Figure 11-6 and the instrumentation detail for the air-operated valves in Figure 11-7, we can calculate the I/O requirements. There are eight (8) 120-VAC solenoid valves, so we need eight 120-VAC PLC outputs. There are sixteen (16) limit switches for the open and closed positions of the valves and there are also four differential pressure switches. So, we need 20 discrete input points for this application. We can arbitrarily select 120 VAC for the input signal voltage to save on the types of modules and supply voltages used in the system.

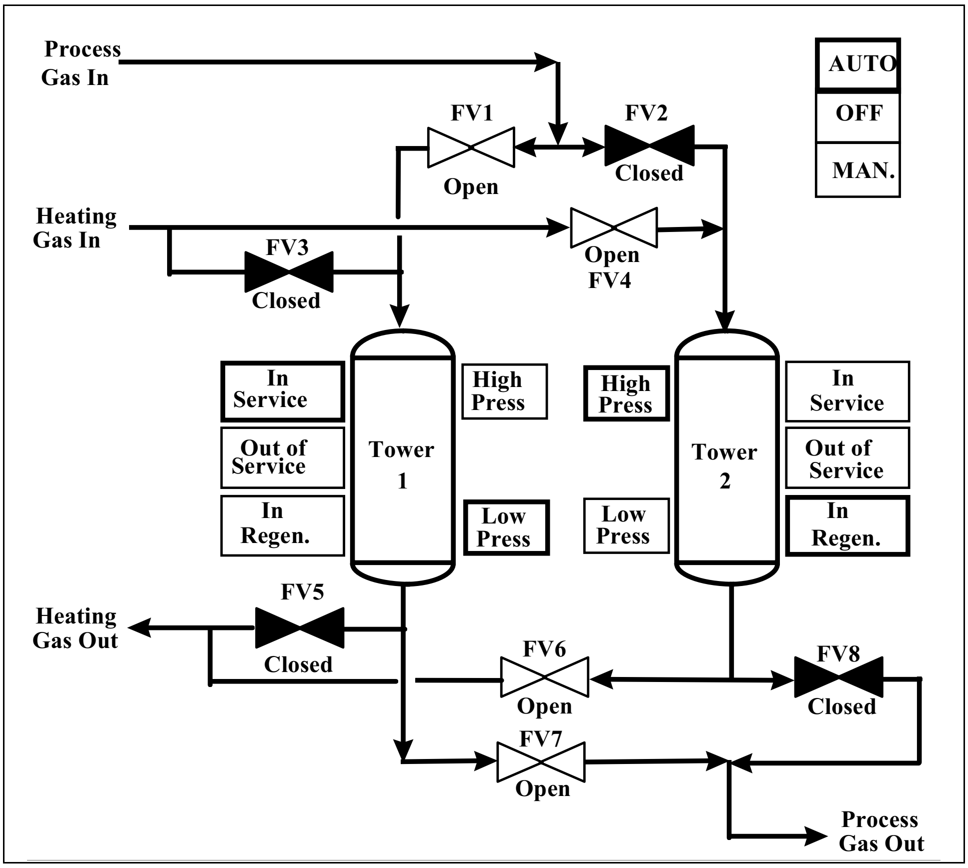


Figure 11-8. Dehydration HMI Graphics Display

Let us assume that we select Allen-Bradley (A-B) 8-point, 120-VAC I/O modules for use in our application. The number of modules required can be calculated as shown in the next example.

EXAMPLE 11-3

Problem: Estimate the number of A-B 8-point, 120-VAC I/O modules required for the PLC dehydration application shown in Figure 11-6.

Solution: To calculate the number of discrete I/O modules, we use the following equation: Number of modules required (N) =

(I/O points + 10% x points)/ (8 points/module)

1) 120 VAC input modules:

N = [(20 + 20 x .20)/8] modules

N= (24/8) modules

Or N= 3 input modules

2) 120 VAC output modules:

N = (8 + 8 x .20)/8 modules

N = (9.6/8) modules = 1.2 modules

Or N = 2 output modules

System Drawing

The system drawing for this application will consist of an A-B PLC5/15 programmable controller, a PC with HMI and PLC programming software installed, a printer to document the program, and a single I/O rack with 8 slots to hold the five I/O modules, as shown in Figure 11-9.

The system diagram is normally plotted on a "D" size (24 inches by 36 inches) drawing sheet and would include a detailed material list. This list includes the equipment number, material description, manufacturer, and model number, as shown in Table 11-1.

We have selected an Allen-Bradley PLC5-15 for this application and an IBM Pentium PC for the HMI and PLC programming software. An 8-slot A-B chassis was chosen to hold the AC input and output modules as listed in Table 11-1.

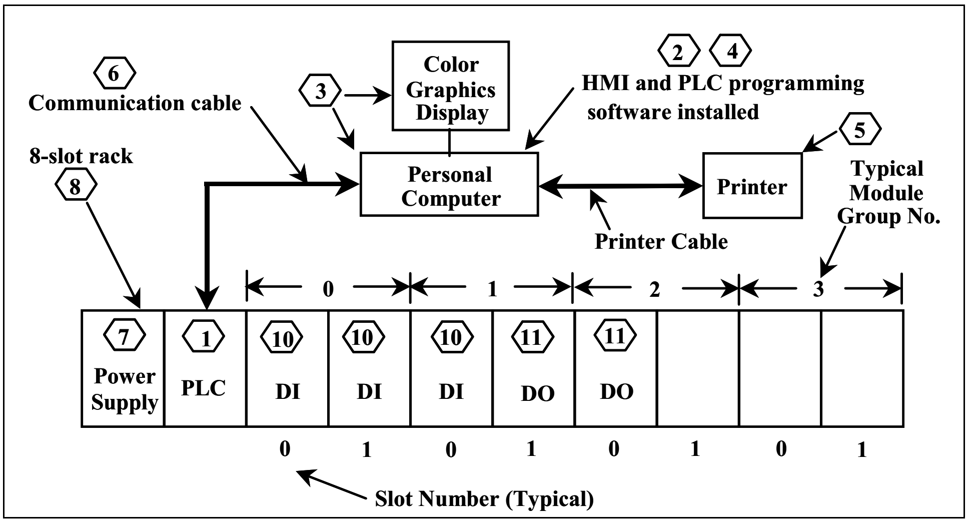


Figure 11-9. Dehydration Application PLC System Drawing

Equip. No.	Material Description	Manufacturer	QTY	Model No.
1	Programmable Controller	Allen-Bradley	1	1785-LT.
2	PLC Programming Software	Allen-Bradley	1	6200
3	Personal Computer	IBM	1	Pentium
4	HMI Software	Rockwell	1	RS-View
5	Graphics Printer	Hewlett-Packard	1	LaserJet
6	Communication Cable	Allen-Bradley	1	1784-CP.
7	PLC5 Rack Power Supply	Allen-Bradley	1	1775-P1
8	8-Slot PLC Chassis	Allen-Bradley	1	1771-APB
9	120-VAC Input Module	Allen-Bradley	3	1771-IA
10	120-VAC Output Module	Allen-Bradley	2	1771-OA

Table 11-1. Dehydration Control System Material List

The placement of the input modules and output modules in an I/O equipment rack is important. For example, if the system has a mix of AC, DC, and analog modules, the AC modules are normally segregated from low-level signal modules, such as analog I/O (4 to 20 mA DC), 5 VDC inputs, and millivolts (mV) input modules, to avoid electrical interference.

I/O Wiring Diagrams

The I/O wiring diagram for the first AC output module is shown in Figure 11-10. This drawing shows the wiring of the system solenoid valves to the AC output module. In our example application, a single AC line is used where the AC hot is normally designated by L1 and AC neutral has wire number L2. However, in larger systems, more than one AC line might be used, depending on loading requirements. The design engineer must also consider maintenance of the system so that in our example application we might connect the wiring for tower 1 to one AC circuit and the wiring to the second tower to a second AC circuit. In this case, one tower could be taken out of service and the other tower could be kept on line during maintenance of the second tower. Furthermore, the wiring for the control panel might be placed on a third AC power circuit.

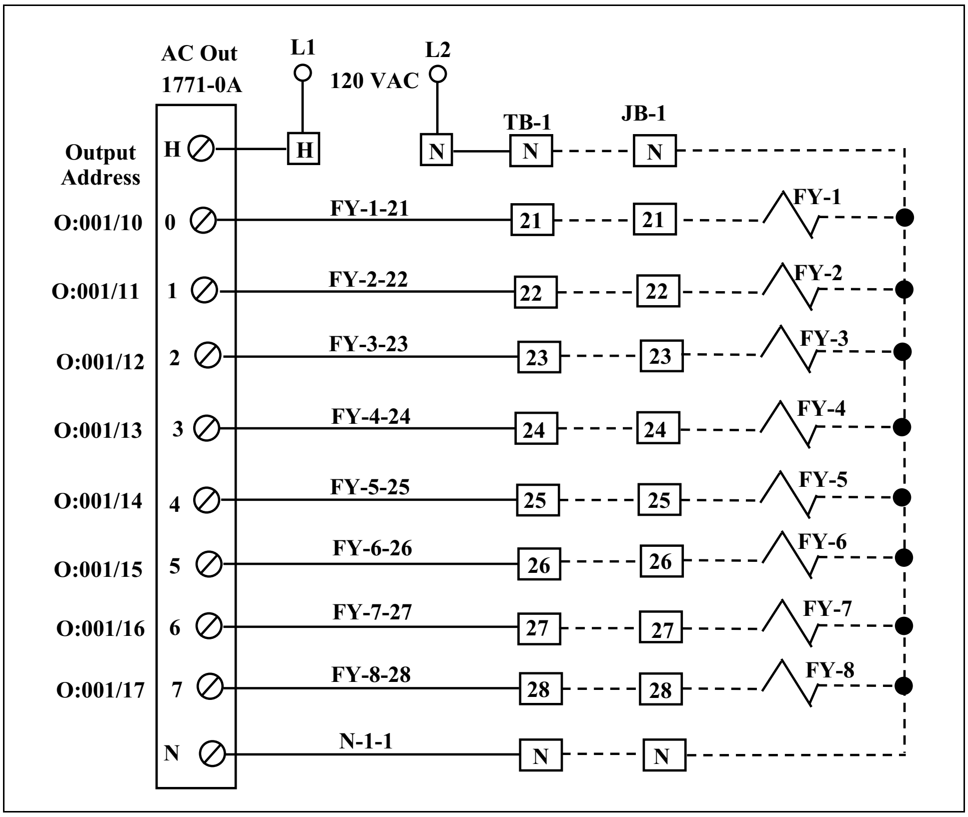


Figure 11-10. Wiring Diagram for Solenoid Valves on Dehydration System

The wiring of the AC input modules is performed in a similar manner, except that the field inputs are normally drawn on the left side of the input module as shown in Figure 11-11. This input wiring diagram shows the connection of the high and low pressure switches for towers 1 and 2 to the first AC input module. As shown in the system drawing, the first AC input module is in rack 00, module group 0, slot 0, so that the Allen-Bradley input addresses are between bit I:000/00 and bit I:000/07 as shown on the right-hand side of the module input drawing.

The input-wiring diagram in Figure 11-12 shows the connection of the valve limit switches for control valves FV-1 through FV-4. As shown in the

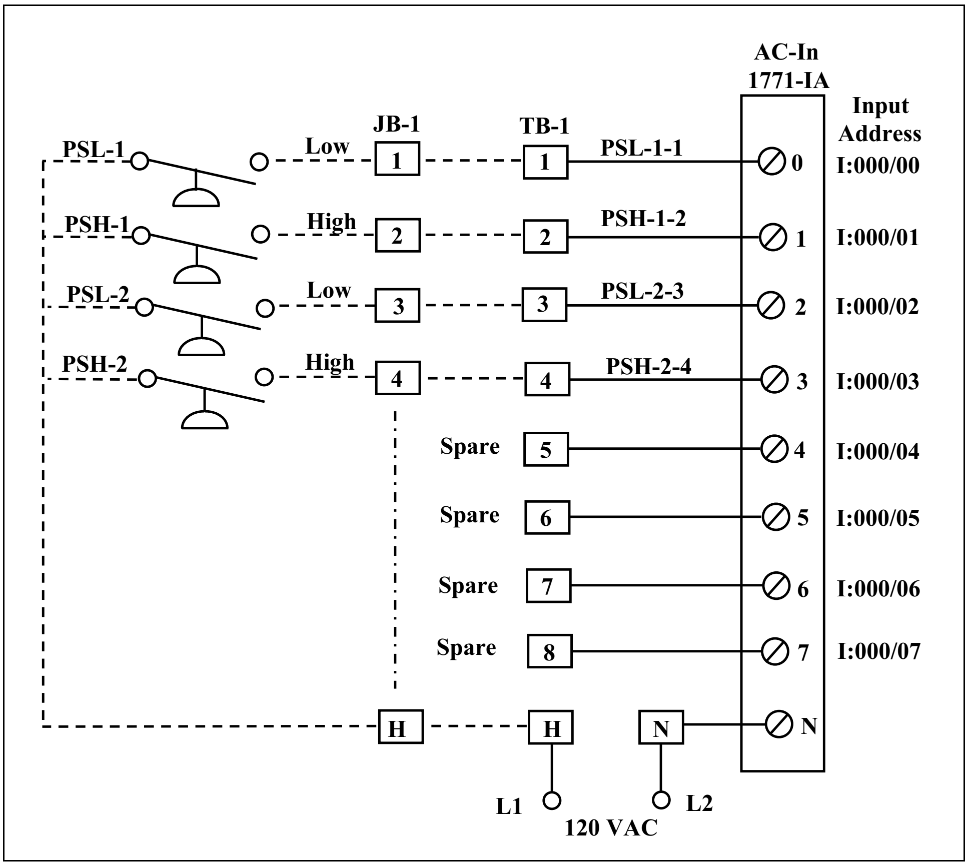


Figure 11-11. Wiring Diagram for Differential Pressure Switches

system drawing, this AC input module is in rack 00, module group 0, slot 1 so that the input addresses are between bit 111/10 and bit 111/17 as shown on the right-hand side of the module input drawing.

The wiring diagram for the limit switches on the remaining four control valves (i.e., FV-5, FV-6, FV-7, and FV-8) is shown in Figure 11-13. This AC input module is in rack 0, slot 0, module-group 1 so that the input addresses are between bit I:001/00 and bit I:001/07 as shown on the right-hand side of the module input drawing.

Application Programming

In a small system like the dehydration application, basic ladder programming is the best choice since the system requires only simple on/off control and there are no complex analog or data manipulations anticipated. The first step in the logic programming is to make a list of internal bits that interface with the HMI. The next step is to list the I/O points in the system using the I/O module wiring diagrams. The internal bits that interface with the HMI are listed in Table 11-2.

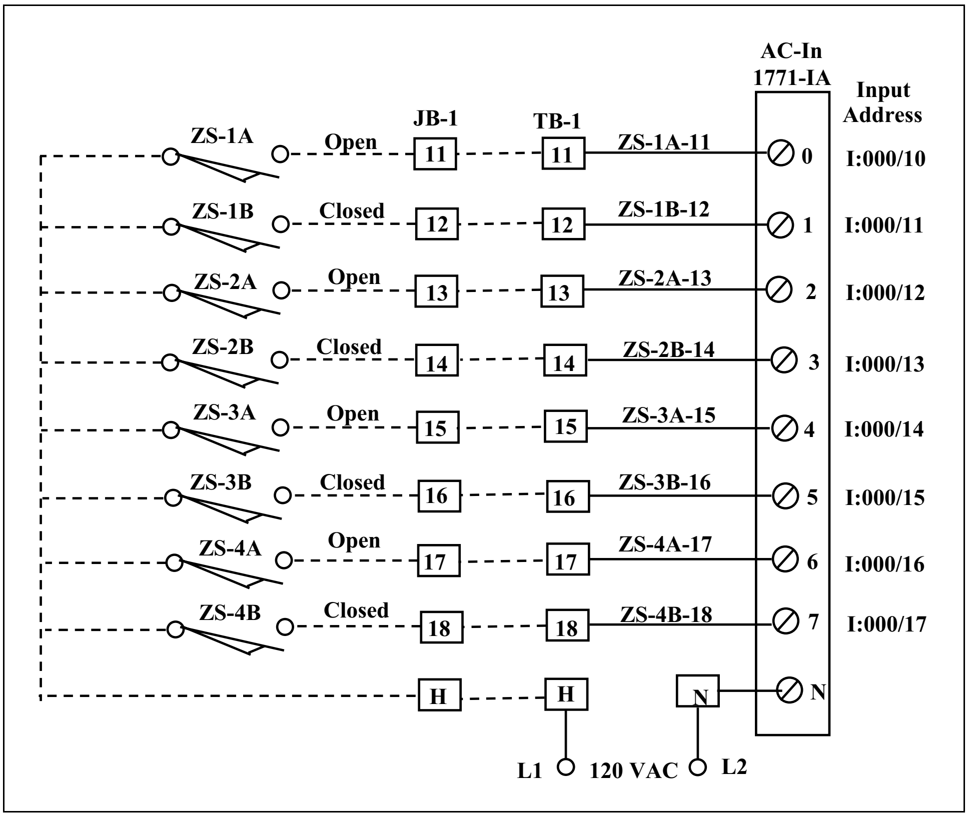


Figure 11-12. Wiring Diagram for Valve Position Switches (ZS-1 through 4)

These bit assignment tables are an aid in programming because they consolidate the I/O information in a single table for easy reference. In most PLC programming software packages, these lists can be entered directly into the control program using the programming software.

The operation of this LD program is relatively simple. When the operator clicks on the AUTO icon shown in the upper right-hand corner of Dehydration HMI graphics display (see Figure 11-8), this sets internal bit B3/00 to logic 1 and places the control system into the automatic mode. The HMI software is configured to deactivate the OFF and manual (MAN.) ions on the screen, if the automatic mode is selected by the operator clicking the "AUTO" icon on the HMI display screen.

We can write the LD program for the automatic mode of the process, as shown in Figure 11-14. Tower 1 is placed in service if the differential pressure in the tower is low, the "AUTO" function (internal bit B3:0/00) is true, and the tower 1 differential pressure is not high (bit I:000/00). The output for "tower 1 in service" is sealed in with its own control bit B3:0/3. Tower 1 will stay "in service" until the pressure across the tower reaches a high level as measured by the high differential pressure (PSH-1) on tower

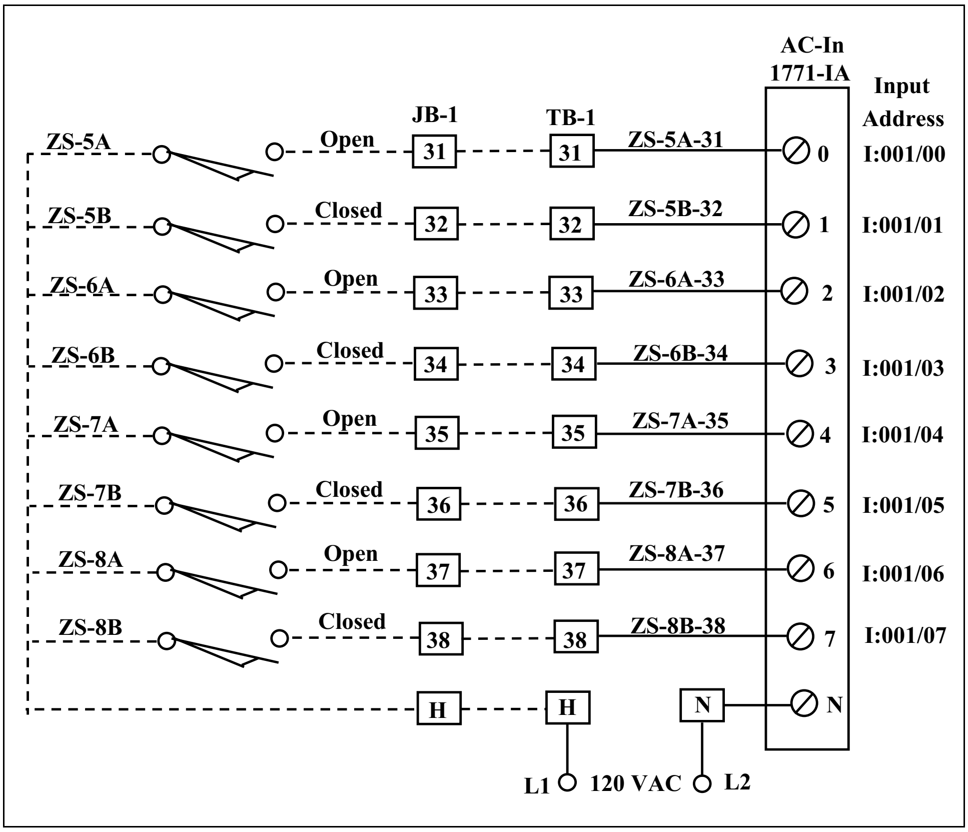


Figure 11-13. Wiring Diagram for Valve Position Switches (ZS-5 through 8)

Bit Address	Description	Bit Address	Description
B3:0/00	Automatic mode	B3:0/05	T1In regeneration
B3:0/01	Off	B3:0/06	T2In service
B3:0/02	Manual mode	B3:0/07	T2Out of service
B3:0/03	T1in service	B3:0/08	T1In regeneration
B3:0/04	T1Out of service		

Table 11-2. HMI Bit Assignments

1. If the differential pressure switch PSH-1 is closed, then input bit I:000/00 is set to true and its normally closed contacts are opened. So, the "tower 1 in service" output coil bit B3/03 is turned OFF.

Tower 1 is placed into regeneration by the control system to remove moisture that has built up in the Dehydration Tower during the service cycle using the ladder logic software shown in Figure 11-15.

We are now ready to control the flow valves on tower 1. All valves are designed to fail closed (FC), so if plant air or electric power is lost, the valve will close. Therefore, the solenoid valves must be energized to open the valve. The ladder diagram program in Figure 11-16 shows the control logic for solenoid valves FY-1, FY-7, FY-3, and FY-5.

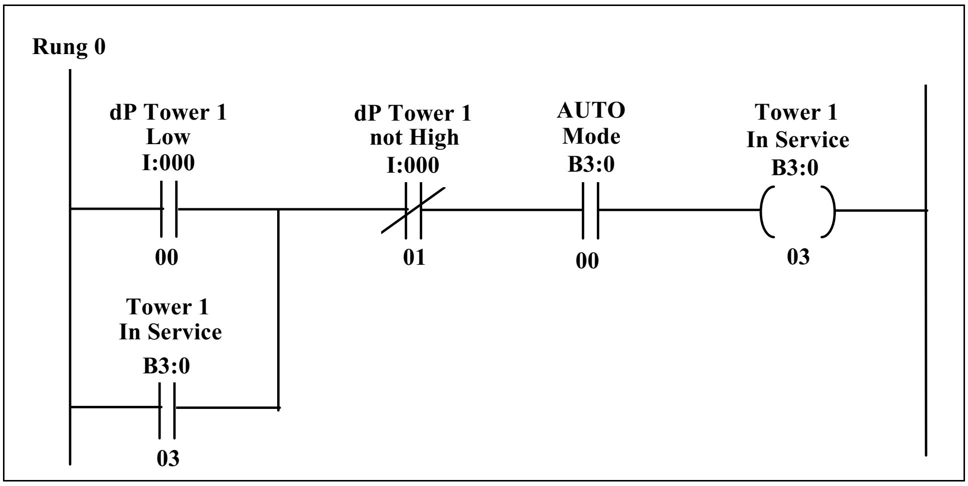


Figure 11-14. LD Program for Tower 1 in Service

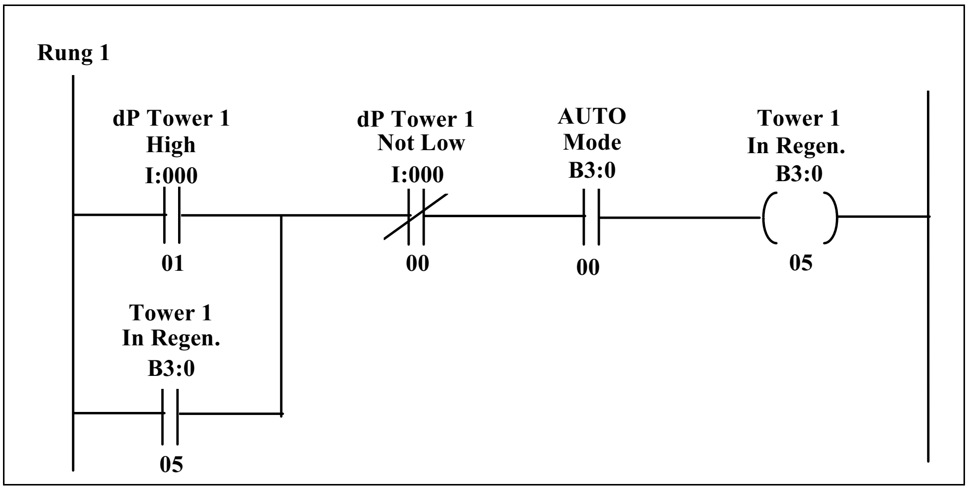


Figure 11-15. LD Program for Tower 1 - In Regeneration

The same design procedure can be used to write the control software for tower 2. To this point, we have not used the open and closed limit switches in our system program. The valve limit switches are used to animate the process graphics valve icons on the HMI display during each step in the process. If a valve is closed, the valve symbol on the graphics screen will be filled in with a selected color (normally red) by the HMI software based on the information sent to the PC from the PLC.

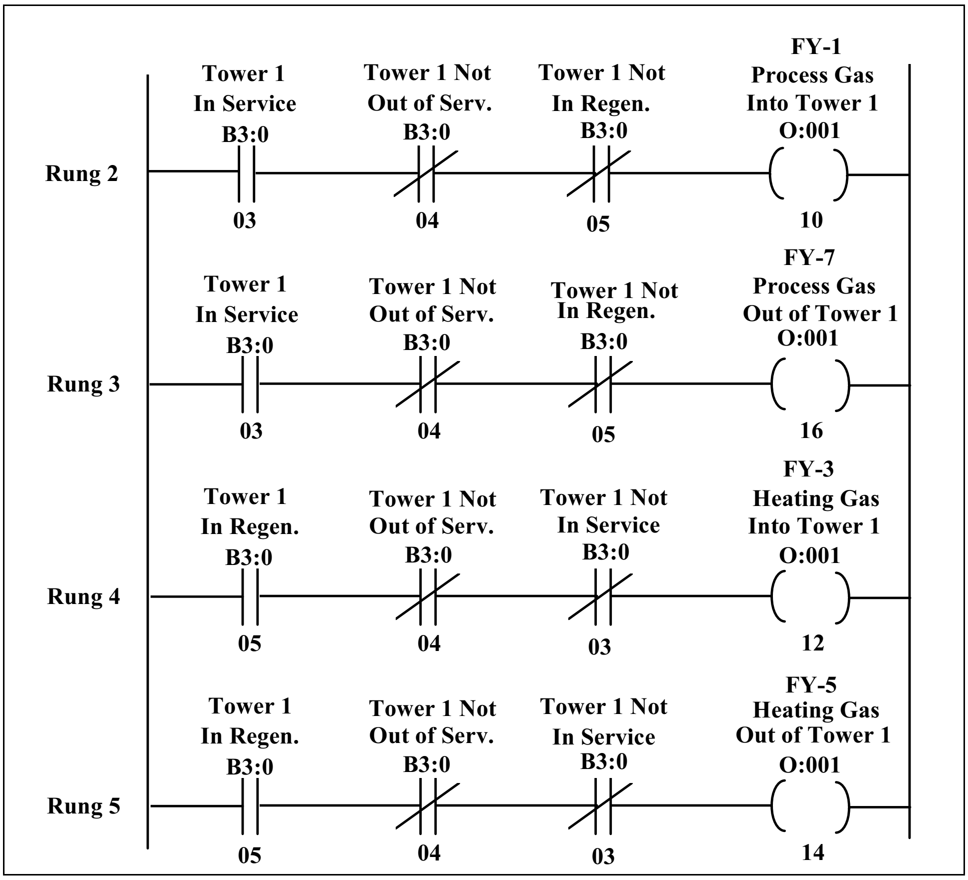

Rung 2
Tower 1 In Service B3:0 03
Tower 1 Not Out of Serv. B3:0 04
Tower 1 Not In Regen. B3:0 05
FY-1 Process Gas Into Tower 1 O:001 10
Rung 3
Tower 1 In Service B3:0 03
Tower 1 Not Out of Serv. B3:0 04
Tower 1 Not In Regen. B3:0 05
FY-7 Process Gas Out of Tower 1 O:001 16
Rung 4
Tower 1 In Regen. B3:0 05
Tower 1 Not Out of Serv. B3:0 04
Tower 1 Not In Service B3:0 03
FY-3 Heating Gas Into Tower 1 O:001 12
Rung 5
Tower 1 In Regen. B3:0 05
Tower 1 Not Out of Serv. B3:0 04
Tower 1 Not In Service B3:0 03
FY-5 Heating Gas Out of Tower 1 O:001 14

Figure 11-16. LD Program for Tower 1 Valves

Two-Stage Alternating Pump Application

In this application, we will develop the ladder diagram program to alternate pumps in processes like the emptying of wells, reservoirs, and tanks where the rate of flow into the tank is not constant. In this type of application, two smaller pumps are frequently used instead of one large one to reduce equipment cost. Another advantage of two pumps over one is that a pump can be taken out of service for maintenance and the system still has some pumping capacity if needed.

Alternating pump operation (pump 1 as the primary, then pump 2 as the primary) reduces the maintenance required on the individual pumps and provides more reliable operation. In this application, the secondary or “standby” pump is available if the rate of water entering the vessel is more than the first pump can handle. If this situation occurs, the second pump will also turn on and assist the primary pump. The triggers for these events could be analog signals, or simple discrete inputs (level switches, etc.).

Process Control Drawing

The first step in the design of the PLC-based control system is to draw the piping and instrument diagram as shown in Figure 11-17.

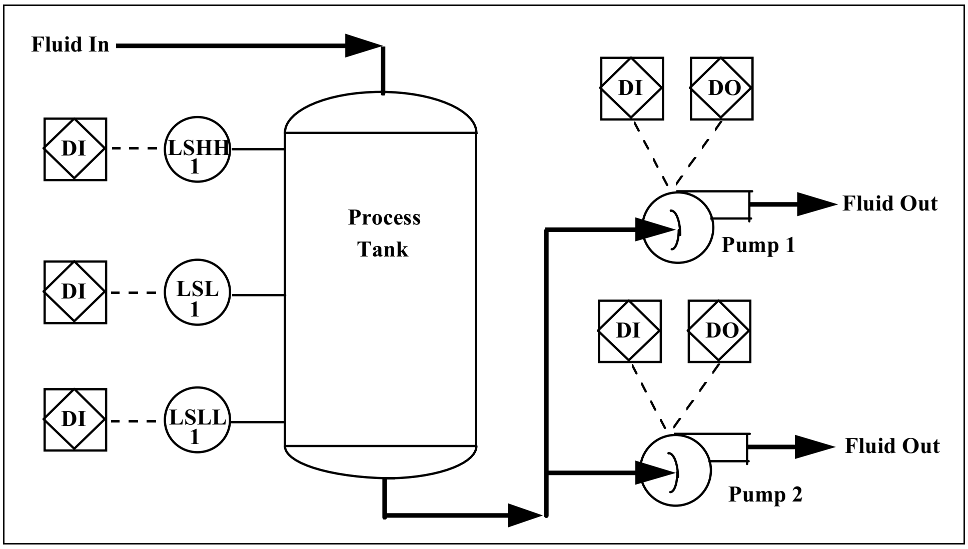


Figure 11-17. Alternating Pump Control of Tank Level

Sizing and Selection of PLC

The next step in the design procedure is to select a PLC and I/O types. Since there are only five discrete inputs and two discrete outputs, a Micro PLC is sufficient for this application. We can arbitrarily select an A-B Micro-1000 PLC with ten discrete inputs and six discrete outputs. The problem with selecting a micro PLC is that there is very little capacity for future additions or changes to the system design. We can now complete the design of an input/output wiring diagram as shown in Figure 11-18.

The ten inputs (addresses I:0/0 through I:0/9) are shown on the left side of the micro PLC and the five AC outputs (addresses O:0/0 through O:0/5) are drawn on the right side of the unit. In this diagram, we have connected three normally open (NO) level switches: LSHH-1, LSL-1, and LSLL-1, to the first three inputs on the micro-PLC at inputs I:0/0, I:0/1 and I:0/2, respectively. We have connected two normally opened auxiliary (aux.) contacts from the pump motor starters (M1 and M2) to

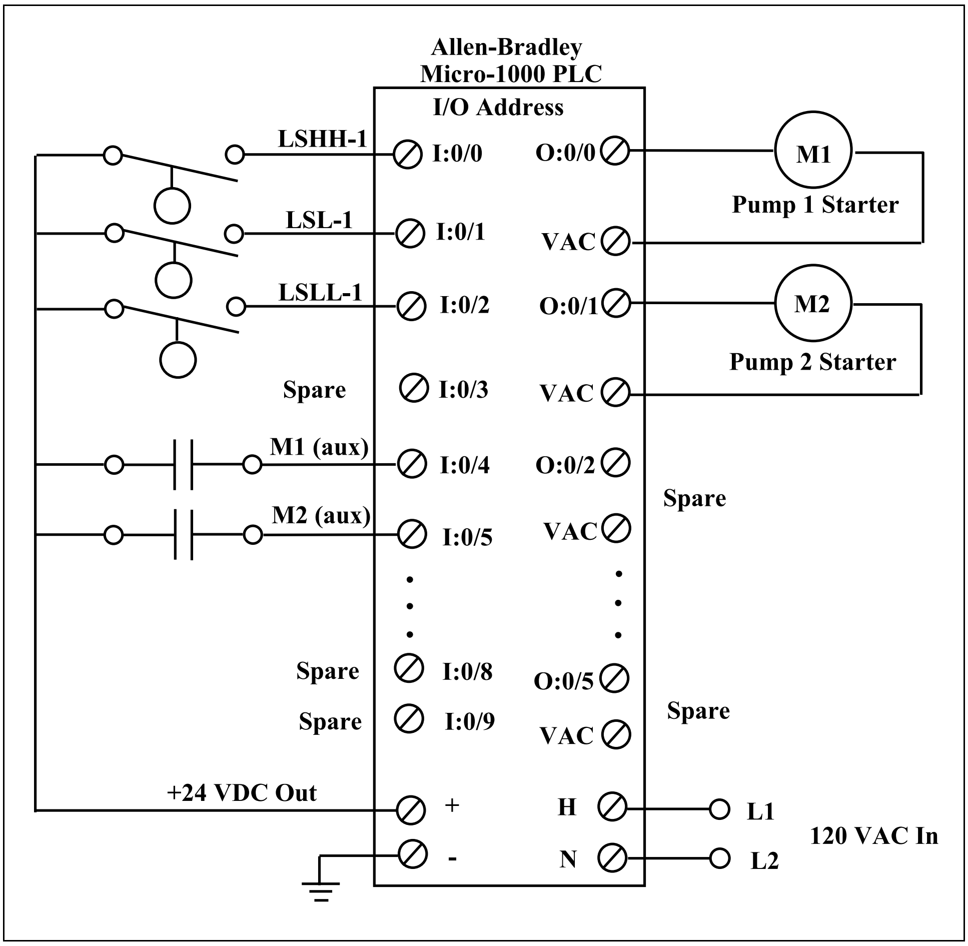


Figure 11-18. Wiring Diagram for Alternating Pump Control

inputs I:0/4 and I:0/5. The pump motor starter relays are connected to the first two AC outputs on the micro PLC at points O:0/0 and O:0/1.

System Programming

The final design step is to write the control program for this application. The ladder diagram program used in this application consists of four rungs. Rungs 0 and 1 form an alternating circuit, so that each time the fluid in the tank is low, switch LSLL-1 is closed and this sets bit I:0/2 to 1, the alternator bit in Rung 1, B3:0/2 changes state. The status of this bit determines which pump will be the first to turn on. The one-shot rising (OSR) instruction in Rung 0 is a specialized instruction that is only energized for one processor scan. This causes internal bit B3:0/1 to be energized for one processor scan, if the low-low level switch (bit I:0/2) is closed as shown in Rung 0 of Figure 11-19.

Rung 2 controls the operation of pump 1, using output bit O:0/0 on the micro PLC. If the low-low-level switch is closed, it sets bit I:0/2 to 1. If the alternator internal bit B3:0/2 is turned off and if the level in the tank has reached the low-level switch (LSL-1) setting bit I:0/1 to 1, then pump 1 will be the first pump to be energized. If the alternator bit B3:0/2 is On, then pump l will be the second pump to be energized as shown in Rung 2 of Figure 11-20.

Rung 3 controls the operation of pump 2 using output bit O:0/1. If the low-low-level switch is on (bit I:0/2 is set), and the alternator bit B3:0/2 is ON, and the level in the tank has reached the low-level switch (i.e., Bit I:0/ l is set to 1), pump 2 will be the first pump to be energized. If B3:0/2 is OFF, pump 2 will be the second pump to be energized as shown in Rung 3 of Figure 11-21.

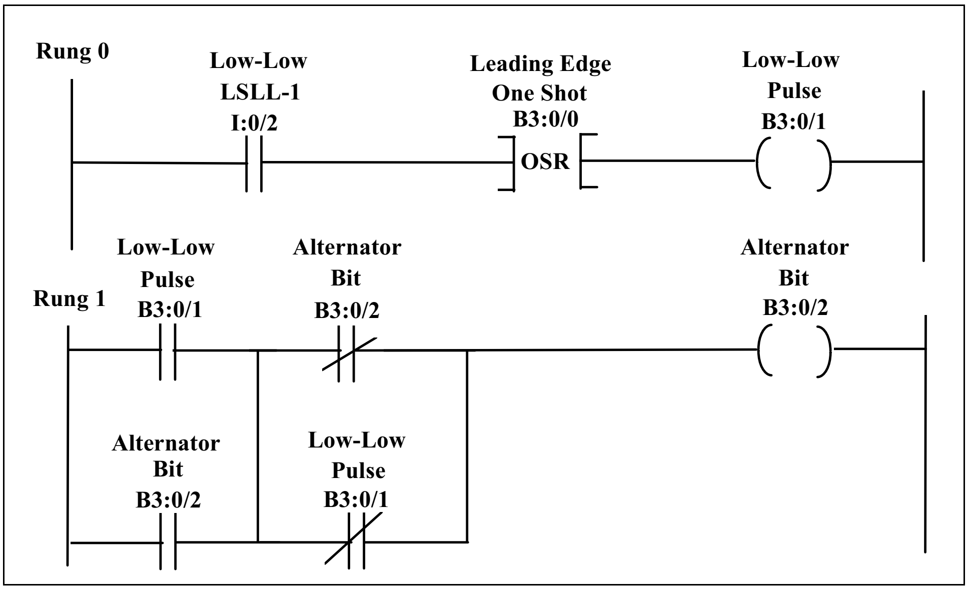


Figure 11-19. LD Program for Alternating Pumps Control (Rungs 0 & 1)

EXERCISES

11.1 List the main design elements required in the design of a PLC control system.

11.2 What is the main function of Piping and Instrument Drawings?

11.3 What is the ISA instrumentation symbol used to indicate PLC input or output signals?

11.4 List several purposes for simplified process or mechanical control diagrams.

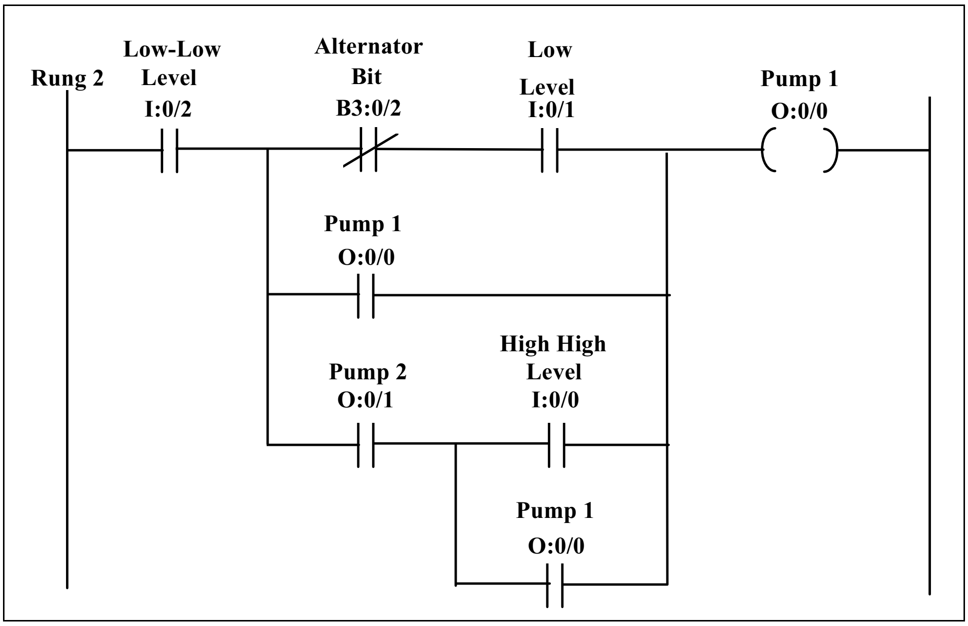


Figure 11-20. LD Program for Alternating Pumps Control (Rung 2)

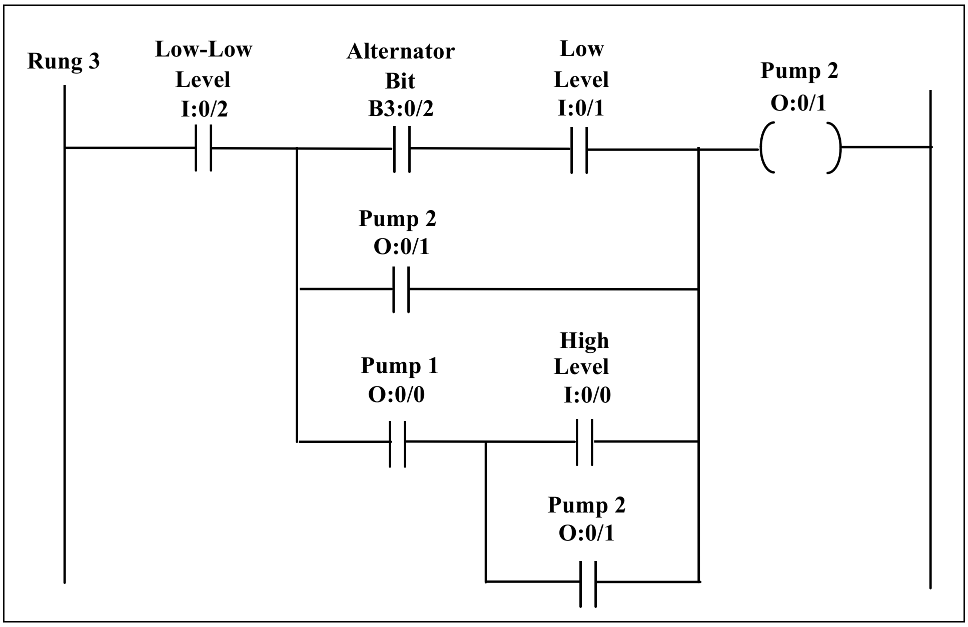


Figure 11-21. LD program for Alternating Pumps Control (Rung 3)

11.5 What is the main purpose for a process control description in PLC system design?

11.6 Estimate the memory size required for a PLC application with the following I/O points: 1) Main Control Room: 150 points, 2) Remote

Process Area 1: 254 points, 3) Remote Process Area 2: 125 points, and 4) Remote Process Area 3: 156 points.

11.7 What is the most common programming language used in PLC applications?

11.8 List some of the common peripheral equipment used in PLC applications.

11.9 What is the purpose of a PLC system drawing?

11.10 Calculate the number of input and output modules required on the dehydration system, if a third process tower is added. Assume that all inputs and outputs are 120 VAC.

11.11 Write a LD program to control the valves on tower 2 of the dehydration application shown in Figure 11-5.

11.12 Revise the alternating pump application LD program to use the pump starter auxiliary (aux.) contact inputs instead of the pump starter output bits.

BIBLIOGRAPHY

1. Hughes, T. A., *Basics of Measurement and Control*, ISA – The Instrumentation, Systems, and Automation Society, 3rd edition, 2002.

2. *Processor Manual PLC-5 Family Programmable Controllers*, Allen-Bradley Co., Inc., 1987.

3. *Micro Mentor: Understanding and Applying Micro Programmable Controllers*, Allen-Bradley Company, Inc., 1995.

12 Design, Installation, and Maintenance

Introduction

To complete the discussion of programmable controller system concepts and design principles, we need to cover control panel design, equipment installation and layout, operational testing, maintenance, troubleshooting and documentation. The reliability and maintainability of a control system is, in a large part, a function of proper system design and documentation that considers the maintenance aspects of a system.

Control Panel Design

The main feature that separates PLCs from other types of computers is that programmable controllers are in many cases installed in harsh industrial environments. They are, in some cases, simply installed on a metal sheet (subpanel) on the production floor. However, in most cases, programmable controller system components are installed in a metal enclosure or a control panel to protect against atmospheric contaminants such as dust, moisture, oils, and other corrosive airborne substances. These metal enclosures also reduce the effects of electromagnetic radiation generated by electrical or welding equipment.

The panel or enclosure design requires a panel layout design, considerations for heating and maintenance, wiring layout and duct design, power distribution design, and normally the writing of a panel specification.

Control Panel Layout

The panel size depends on the amount of equipment to be installed in the enclosure and whether front panel controls and instruments are required on the system. If front panel controls are required, the size and shape of the panel are mainly controlled by the best layout design of these front panel instruments. To ensure correct and easy operation of the system, the indicators and recorders are placed at normal eye level and hand switches are placed below the indicators.

The metal enclosure should conform to industrial standards, such as the National Electrical Manufacturers Association (NEMA) standards. The NEMA standard covers the design of industrial enclosures for different industrial environments.

The equipment layout inside the panel should follow the recommendations contained in the programmable controller installation manual. In this manual, the manufacturer generally lists the minimum spacing allowed between I/O racks, other equipment, and the processor. This spacing generally takes into consideration equipment heating, electrical noise, and safety factors. Figure 12-1 shows a typical minimum equipment spacing of 6 inches for a programmable controller system per a PLC manufacturer's recommendation.

Heating Considerations

To allow for effective convection cooling, most manufacturers recommend that all system components be mounted in a position that allows for maximum airflow in the enclosure. Since the power supplies generate the most heat, they should not be mounted directly underneath another system component. Generally, the main power supply is mounted near the top of the enclosure, but some PLC manufacturers use individual power supplies on each I/O rack and an internal power supply for the processor. In Figure 12-1, notice the 6-inch horizontal gap between the I/O racks to allow for adequate cooling airflow.

The temperature inside the control panel or enclosure must not exceed the maximum operating temperature listed in the manufacturer's installation manual (typically 120°F). If the temperature limit cannot be maintained by convection cooling, a fan or blower must be installed to help dissipate the heat. The fan or fans used will generally be equipped with filters to prevent dust, dirt, and other airborne contaminants from entering the enclosure and affecting system components.

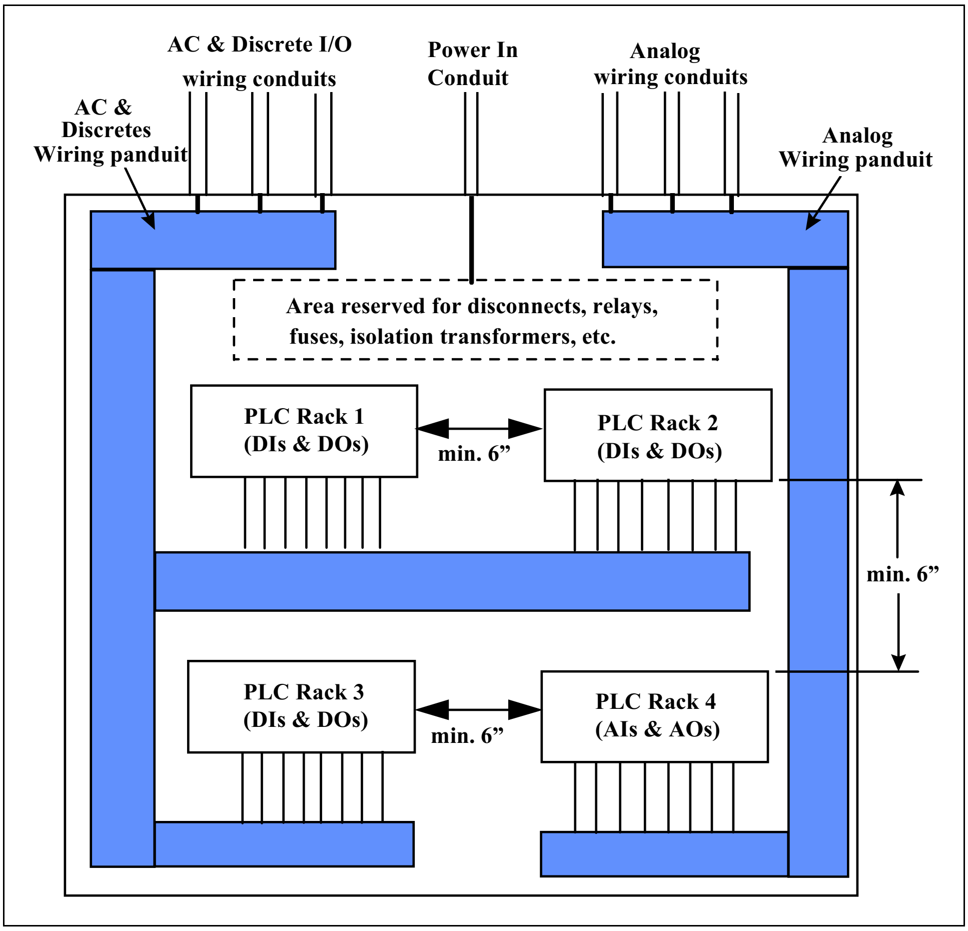


Figure 12-1. PLC Equipment Layout Diagram

Enclosure Standards

All enclosures installed in industrial applications must meet the NEMA standard in publication number 250-1979. The following descriptions are excerpts from this NEMA standard and the enclosure types are rated for "Non-classified Locations" and "Classified Locations" (i.e., explosive locations).

Non-classified Location Enclosure Types

Type 1 Enclosure
Type 1 enclosures are intended for indoor use primarily to provide a degree of protection against contact with the enclosed equipment in locations where unusual service conditions do not exist. The enclosures shall meet the rod entry and rust-resistance design tests.

Type 2 Enclosures

Type 2 enclosures are intended for indoor use primarily to provide a degree of protection against limited amounts of falling water and dirt. These enclosures shall meet rod entry, drip, and rust-resistant design tests. They are not intended to provide protection against conditions such as internal condensation or internal icing.

Type 3 Enclosures

Type 3 enclosures are intended for outdoor use primarily to provide a degree of protection against windblown dust, rain, sleet, and external ice formation. They shall meet rain, external icing, dust, and rust-resistance design tests. They are not intended to provide protection against conditions such as internal condensation or internal icing.

Type 3R Enclosures

Type 3R enclosures are intended for outdoor use primarily to provide a degree of protection against falling rain, sleet, and external ice formation. They shall meet rod entry, rain, external icing, and rust-resistance design tests. They are not intended to provide protection against conditions such as dust, internal condensation, or internal icing.

Type 4 Enclosures

Type 4 enclosures are intended for indoor or outdoor use primarily to provide a degree of protection against windblown dust and rain, splashing water, and hose-directed water. They shall meet hose-down, dust, external icing, and rust-resistance design tests. They are not intended to provide protection against conditions such as internal condensation or internal icing.

Type 4X Enclosures

Type 4X enclosures are intended for indoor or outdoor use primarily to provide a degree of protection against corrosion, windblown dust and rain, splashing water, and hose-directed water. They shall meet the hose-down, dust, external icing, and corrosion-resistance design tests. They are not intended to provide protection against conditions such as internal condensation or internal icing.

Type 5 Enclosures

Type 5 enclosures are intended for indoor use primarily to provide a degree of protection against dust and falling dirt. They shall meet the dust and rust-resistance design tests. They are not intended to provide protection against conditions such as internal condensation.

Type 6 Enclosures
Type 6 enclosures are intended for indoor or outdoor use primarily to provide a degree of protection against the entry of water during occasional temporary submersion at a limited depth. They shall meet submersion, external icing, and rust-resistance design tests. They are not intended to provide protection against conditions such as internal condensation, internal icing, or corrosive environments.

Type 6P Enclosures
Type 6P enclosures are intended for indoor or outdoor use primarily to provide a degree of protection against the entry of water during prolonged submersion at a limited depth. They shall meet air pressure, external icing, and corrosion-resistance design tests. They are not intended to provide protection against conditions such as internal condensation or internal icing.

Type 11 Enclosures
Type 11 enclosures are intended for indoor use primarily to provide, by oil immersion, a degree of protection to enclosed equipment against the corrosive effects of liquids and gases. They shall meet drip and corrosion-resistance design tests. They are not intended to provide protection against conditions such as internal condensation or internal icing.

Type 12 Enclosures
Type 12 enclosures are intended for indoor use primarily to provide a degree of protection against dust, falling dirt, and dripping non-corrosive liquids. They shall meet drip, dust, and rust-resistance tests. They are not intended to provide protection against conditions such as internal condensation.

Type 12K Enclosures
Type 12K enclosure with knockouts are intended for indoor use primarily to provide a degree of protection against dust, falling dirt, and dripping non-corrosive liquids other than at knockouts. They shall meet drip, dust, and rust-resistance design tests. Knockouts are provided in the top and/or bottom walls only. After installation, the knockout areas shall meet the environmental characteristics listed above. They are not intended to provide protection against conditions such as internal condensation.

Type 13 Enclosures
Type 13 enclosures are intended for indoor use primarily to provide a degree of protection against dust, spraying of water, oil, and non-corrosive coolant. They shall meet oil exclusion and rust-resistance design tests. They are not intended to provide protection against conditions such as internal condensation.

Classified Location Enclosure Types

Type 7 Enclosures
Type 7 enclosures are for indoor use in locations classified as Class I Groups A, B, C, or D, as defined in the *National Electrical Code*.

Type 8 Enclosures
Type 8 enclosures are for indoor or outdoor use in locations classified as Class II, Groups A, B, C, or D, as defined in the *National Electrical Code.*

Type 9 Enclosures
Type 9 enclosures are intended for indoor use in locations classified as Class II, Groups E, F, or G, as defined in the *National Electrical Code*.

Type 10 Enclosures (MSHA)
Type 10 enclosures shall be capable of meeting the requirements of the Mine Safety and Health Administration, 30 C.F.R., Part 18 (1978).

Maintenance Design Features

The system designer must include certain features in the design of the enclosure to reduce maintenance time and cost. One important consideration is the accessibility of equipment components and terminal connections. For example, the processor should be placed at a normal working level for ease of operation or maintenance. If the processor and the system power supply are contained in a single unit, they should be placed near the top of the enclosure. However, if there is adequate space in the enclosure, they should be placed to improve operation or maintenance.

Another maintenance feature is that the control panel should have an AC power outlet strip so maintenance can plug in test equipment and a portable light if needed. If the panel is large, interior lighting should be installed to aid maintenance and operations personnel in troubleshooting. Both the AC power strip and the interior lighting should be on a separate AC circuit from the other system components.

In some harsh environmental applications, a gasketed glass window is used to allow for viewing of the processor status lights and/or I/O status indicators by operations and maintenance personnel without opening the panel door. This is an important feature in dirty and corrosive environments to prevent damage to control components in the panel. In some applications, the operator will check the status of process and I/O lights during each step in a process to make sure there are no abnormal operating conditions.

Panel Duct and Wiring Design

The types of signals used in the system design determine the wiring duct layout. It also depends on the placement of the I/O modules in the racks. The main consideration is to reduce the electrical noise caused by crosstalk between I/O signal lines.

All AC power wiring should be kept separated from low-level DC wires. If DC lines must cross AC signal or power lines, it should be done at right angles only. This routing practice minimizes the possibility of electrical interference.

Power Distribution Design

Most programmable controller manufacturers recommend that a power isolation transformer be installed between the AC power source and the PLC equipment to provide for signal isolation from other equipment in the process area. A typical power distribution drawing is shown in Figure 12-2. In this application, the incoming AC lines (L1, L2, and L3) are 460 VAC used for power devices, such as motors, heaters, or pump starters in the system. There are three fuses on the incoming lines to protect against any over-current condition. In the drawing, the 460 VAC power lines L1 and L2 are connected to the primary of the step-down transformer. This transformer steps down or reduces the AC voltage to 120 V and provides isolation for the programmable controller components from the outside electrical equipment.

The AC distribution circuit contains a ″master control relay,″ labeled as MCR in the circuit diagram. This relay is used to stop the operation of the programmable controller or the controlled machine or process when any emergency stop (E-stop) pushbutton is depressed. Any number of E-stop switches can be used in the AC distribution circuit to improve system safety.

It is also recommended that emergency stop circuits be designed into the system for every machine being directly controlled by the programmable controller. These circuits should be hard-wired and totally independent of the programmable controller to provide for maximum safety in the control system.

Programmable controllers are very reliable devices, but failure of the central processor can cause dangerous and erratic behavior of the control system. Therefore, the operator must be able to quickly and safely turn off process equipment and machinery by using emergency stop (E-stop) switches that are placed in locations easily accessible to the operator. So in cases where people are exposed to moving equipment or machinery that

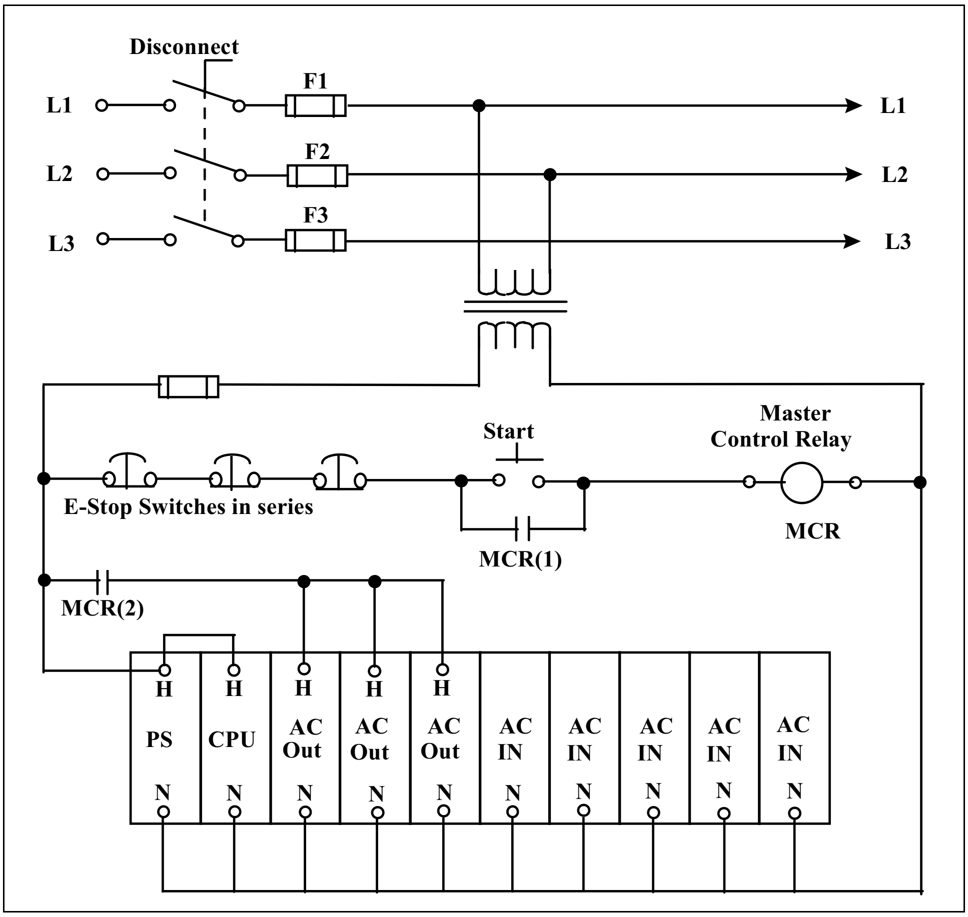


Figure 12-2. Typical PLC AC Power Distribution Drawing

can cycle automatically and injure personnel, an electromechanical override, independent of the PLC, should be used to interrupt electrical power.

The following example problem will illustrate the use of emergency stop safety circuits in PLC control applications.

Grounding Considerations

The reliability of any electrical control system is highly dependent on the proper design of system grounding. Correct electrical grounding design is also required to guarantee a safe electrical installation. When designing and installing any electrical equipment in the U.S., the designer should read and understand Section 250 of the National Electrical Code (NEC). This section provides information on the wire color code, size, and type of conductor required as well as connection methods required for safe grounding of electrical equipment.

EXAMPLE 12-1

Problem: Design a safety circuit for a PLC-based conveyor control system where unexpected operation of the two conveyors in the system could cause injury to system operators. Assume that a 120-VAC output module is used to turn on the conveyor motor starters for the two conveyor motors.

Solution: Design uses two emergency stop switches placed near the conveyors within reach of any operator using the system. The safety circuit used in the design is shown in Figure 12-3.

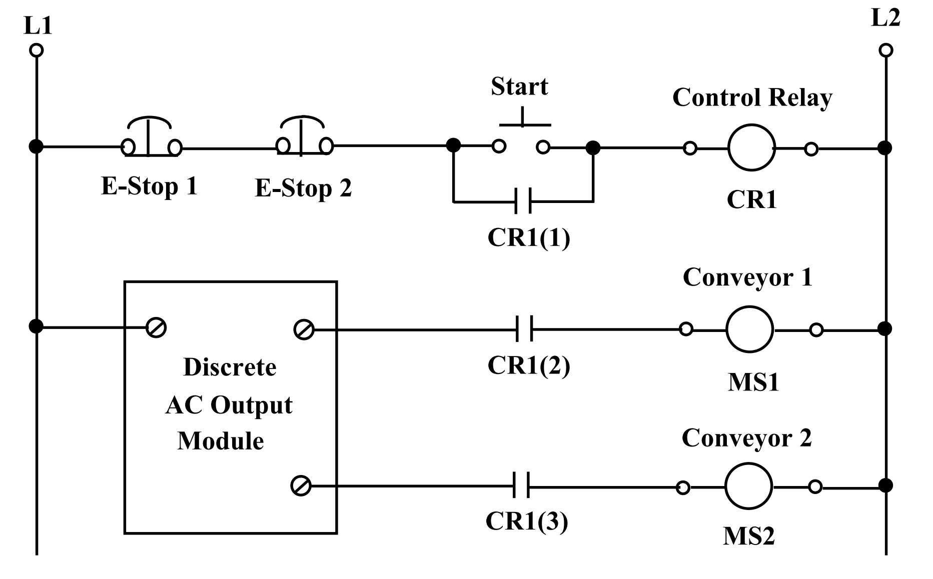


Figure 12-3. Design Solution for Example Problem 12-1

The code states that a ground must be permanent (i.e., no solder connections), continuous, and able to safely conduct the current in the system with minimal resistance. The AC hot wire must have black-colored insulation, the AC neutral wire insulation must be white, and the ground conductors must use green-colored insulation. It is common practice to use red insulation for positive signal wires and black-colored wires for negative DC signal lines, but NEC does not require it.

The ground wire should be separated from the AC hot and AC neutral wires at the point of entry to the control panel. To minimize the ground wire length within the panel, the ground reference point should be located as close as possible to the point of entry of the panel supply line. All I/O racks, power supplies, processors, and other electrical devices in the system should be connected to a single ground bus in the control panel.

Paint or other nonconductive materials should be scraped away from the area where an I/O rack makes contact with the panel. In addition to the ground connection made through the rack mounting bolts, a metal braid of the size recommended by the PLC manufacturer should be used to connect each chassis and the panel at a single mounting bolt.

I/O Module Installation and Wiring

An important part of the hardware design of a PLC system involves the proper layout and wiring diagrams for the input/output module. The actual installation of the modules is a relatively simple procedure with most of the programmable controllers on the market, since the installation crew simply plugs the modules into the I/O racks per the drawing provided by the system designer.

However, in some cases, intelligent modules (such as thermocouple, analog, communications, etc.) might require switch settings to properly configure the module. For example, a thermocouple (T/C) module might be designed to accept the various T/C types, such as J, K, T, S, etc., and the user sets toggle switches in the module to accept the T/C being used in the process. The design engineer must document the switch setting on the I/O drawings or in the installation instructions to guarantee that the modules are properly installed and configured.

Another important procedure required for installation of a programmable controller system is the selection of the I/O chassis number for each I/O rack. Many PLC manufacturers use switch assemblies in the I/O rack to number the racks. A typical programmable controller I/O rack selection switch arrangement is shown in Table 12-1.

I/O Rack Number	Switch Position			
	4	3	2	1
00	Closed	Closed	Closed	Closed
01	Closed	Closed	Closed	Open
02	Closed	Closed	Open	Closed
03	Closed	Closed	Open	Open
04	Closed	Open	Closed	Closed
05	Closed	Open	Closed	Open
06	Closed	Open	Open	Closed

Table 12-1. Typical I/O Rack Number Selection Table

Control Panel Specification

In most cases, an outside control panel vendor fabricates the equipment enclosure for a programmable controller system so that an equipment specification will be required to cover all requirements of a system. The following is a typical control panel specification:

The control panel furnished under this specification shall be supplied complete with the instruments and equipment listed on the enclosed drawings, installed and electrically wired, and ready for wiring to field instruments and equipment.

The control panel shall conform to the following:

1. The control panel shall be fabricated with cold-rolled steel plate.
2. All miscellaneous items, such as wire raceways, terminal strips, electrical wire, etc., where possible, shall be made of fire-resistant materials.
3. The grounding bus bar and studs shall be pure copper metal.
4. A minimum of two 120-VAC, 60-Hz utility outlets shall be installed in the panel.
5. An internal fluorescent light with a conveniently located ON/OFF switch shall be installed in the panel.
6. A circuit breaker panel with circuit breakers to accommodate all panel lighting, power supplies, instruments, I/O modules, processors, and any other loads listed on the system drawings shall be provided.
7. Each electric wire over 12 inches in length shall be identified at each end with a wire number per the electrical drawings for the system.
8. Terminal strips with screw-type connectors shall be used and no more than two wires shall be terminated on any single terminal.
9. Nameplates shall be made of laminated plastic with white letters engraved on a black background for both front and rear panel-mounted instruments and components, such as power supplies, transformers, and PLC I/O racks.
10. The AC wiring shall be separated from 4- to 20-mA DC current and digital signals by a minimum of 24 inches, and they must be wired to separate terminal strips.

11. All electrical wires and cables shall enter the control panel through the top. Sufficient space shall be provided to allow AC power cables entering the top of the panel to continue directly to the circuit breaker panel.
12. All wires and cables shall be routed and tied to provide clear access to all instruments and components for maintenance and removal of defective components.
13. All DC wires shall have a minimum insulation voltage rating of 600 VAC, and all DC wires shall have a minimum insulation voltage rating of 300 VDC.
14. All electrical conductors shall be copper of the correct wire size for the current carried with 98% conductivity, referenced to pure copper.
15. Wireways shall be attached securely to the control panel.
16. Metal surfaces with wires or cables passing through them shall be furnished with insulated polyethylene grommets to prevent damage to the conductors or cables.
17. All wire bundles or cables shall be clamped to the panel at all right angle turns.
18. All wires entering or leaving a wire bundle shall be tied to the bundle at the point of entering or leaving the main wire bundle.

Equipment Layout Design

Proper control system equipment layout design can reduce installation costs and improve system reliability and maintainability. In addition to the programmable controller components, the equipment layout must also take into account the other system components, such as field devices and instruments, power disconnect boxes, power transformer, and the location of process equipment and machines.

In general, placing the processor near the process equipment and using remote I/O racks where possible will reduce wire and electrical conduit runs. It is possible that the cost of wiring and conduit installation will far exceed the programmable controller equipment costs if care is not taken in the equipment layout design.

The control panel should be placed in a position that allows the doors to be opened fully for easy access to wiring terminals and system components for maintenance and troubleshooting. The National Electrical

Code (NEC) requires that the panel doors open at least to 90°, and there must be a minimum of 36 inches of clearance from the rear of the panel to the nearest grounding surface or wall. Before starting the design of a control panel, the designers should read and understand section 110 of the NEC to avoid any code violations or safety problems.

An emergency disconnect switch should be mounted on or near the control panel in an easily accessible location. If the location where the control panel will be placed contains equipment that generates excessive radio frequency interference (RFI) or electromagnetic interference (EMI), the panel should be placed a reasonable distance from these sources. Examples of such sources include electric machinery, welding equipment, induction heating units, and electric motor starters.

System Start-Up and Testing

The first requirement for successful system start-up is to have a written system operational (SO) test procedure. This procedure should be written by the control system designer and carefully reviewed and approved by all parties involved in the project. A timesaving procedure is to functionally test the control system program using a process simulator before the system is installed on the process.

The SO test procedures will generally contain the following main sections: visual inspection, continuity test, input signal testing, testing of outputs, and process operational testing. A typical SO test procedure follows.

Typical SO Test Procedure

I. Visual Inspection

1. Verify that all system components are installed per the system drawing.
2. Check I/O module location in equipment racks per I/O drawings.
3. Inspect switch settings on all intelligent modules per drawings.
4. Verify that all system communication cables are correctly installed.
5. Check that all input wires are correctly marked with wire numbers and terminated at correct points on the input modules.
6. Check that all output wires are correctly marked with wire numbers and terminated at correct terminals on the output modules.

7. Verify the power wiring is installed per the AC distribution drawing.

II. Continuity Check

1. Use ohmmeter to verify that no AC wire is shorted to ground.
2. Verify continuity of AC hot wiring.
3. Check AC neutral wiring system.
4. Check continuity of system grounds.

III. Input Wiring Check

1. Place the programmable controller in program mode.
2. Disable all output signals.
3. Turn "on" AC system power and power to input modules.
4. Verify the E-stop switch removes power from the system.
5. Activate each input device, observe the corresponding address on the programming terminal, and verify that the indicator light on the input module is energized.

IV. Output Wiring Check

1. Disconnect all output devices that might create a safety problem, such as motors, heaters, or control valves, etc.
2. Place programmable controller in "Run/Program" mode.
3. Apply power to the programmable controller and the output modules.
4. Depress the E-stop pushbutton and verify that all output signals are deenergized.
5. Restart system and use the forcing function in the programming terminal to energize each output individually.
6. Measure the signal at the output devices and verify that the output light on the module is energized for each output tested.

V. Operation Test

1. Place processor in "Run/Program" mode and turn on main power switch.

2. Load the pre-tested control program into the programmable controller.
3. Disable all outputs, select the "run/program" mode on the processor, and verify that the run light on the processor is activated.
4. Check each rung of logic for proper operation by simulating the inputs and verifying on the programming terminal that the correct output is energized at the proper time or sequence in the program.
5. Make any required changes to the control program.
6. Enable output modules and place processor in "Run" mode.
7. Test control system per process operating procedure.

Maintenance Practices

Programmable controller components are designed to be very reliable, but occasional repair is still required. System maintenance costs can be greatly reduced by using good design practices, complete documentation, and preventive maintenance programs.

Preventive Maintenance

A systematic preventive maintenance program will reduce the down time of the control system. The preventive maintenance of programmable controller components is usually scheduled at the same time as the machine or process that is down for maintenance or repair. Normally, since programmable controller equipment is more reliable than some machinery or process equipment, it requires less frequent preventive maintenance operations.

It is very important in a preventive maintenance program to carefully check the various connectors in the system. It is estimated that 70% of the problems found in electronic or computer-based systems are caused by loose, dirty, or defective connectors or connections. The connections to the I/O modules should be checked periodically to make sure no wiring has come loose. The seating of the I/O modules in the equipment rack should also be checked. If the system is located in process areas that have high vibration levels, the preventive maintenance check should be performed more often.

Excessive heat is another major cause of failure in programmable controller systems. Therefore, if an enclosure is cooled with fans, the filters used on the panel must be cleaned or replaced on a regular basis. It is important that dirt and dust is not allowed to build up on system components because a dirt buildup on electronic components can reduce heat dissipation and cause an overheated condition in the system.

Electrical noise can cause erratic and dangerous operation of a PLC system, so the maintenance department must check to make sure that equipment producing high levels of RFI or EMI noise is not moved near the programmable controller equipment.

Maintenance personnel should also check to make sure that unnecessary items are not stored on or near the PLC equipment. Leaving items such as tools, test equipment, drawings, or instruction manuals in the control panel can obstruct the airflow and cause heating problems.

Recommended Maintenance Measures

To reduce the risk of accidents and improve ease of maintenance, it is recommended that the following minimum protective practices and measures be taken: 1) stocking of critical spare parts, 2) documentation of problems encountered, 3) monitoring of internal PLC faults, 4) monitoring of PLC output faults, 5) monitoring of power supplies, 6) strict control of program access, and 7) configuration control of documentation.

Stocking of critical spare parts. The stocking of spare parts is a very important maintenance practice, because it reduces down time in the event of a component failure.

Documentation of problems encountered. Detailed maintenance records should be maintained of all faults and problems encountered. The records should include a code of the problem type, a detailed description of the problem encountered, a description of the corrective action taken, and a list of the part or parts replaced or repaired to correct the problem.

Monitoring of internal PLC faults. Most PLC programming packages include diagnostic software that monitors for internal software faults. If a fault or faults are detected, internal bits are set. These bits can be used to alert the user of the problem and the fault bits can be used in the control program to halt operations if the problem is serious.

Monitoring of PLC output faults. By using additional wiring, other inputs, and software logic, it is possible to ensure that critical outputs react in an appropriate way. Alarms or safety relays can also be activated if, for example, safety related outputs fail to operate properly.

Monitoring of external power supplies. The consequences of the loss of power supplies or power sources on the control system should be studied. In some cases, the input or output circuit, either completely or partially, are powered by different electrical sources than the source supplying the PLC processor. These sources should be monitored and the consequences of their loss to the control system should be clearly defined.

Strict control of program access. In order to avoid problems caused by unauthorized program modifications, it is recommended that access to programs be limited and controlled with passwords and security locks.

Updating of documentation. After each control program change is made, installed, and tested, the software should be saved and a new printout generated. It is important to maintain an easily accessible and understandable printout of the control program. Backup copies of program diskettes should be made and stored in a different location. A written description of the program changes should be made so that the program's evolution from its original installation to present configuration can be tracked in case problems occur at a later time.

Troubleshooting PLC Systems

Troubleshooting can be defined as the methods used to determine why a system or component is not functioning properly. Troubleshooting, as with many practical skills, is an art as well as having an analytical or logical basis. As such, troubleshooting procedures are a trainable skill. Figure 12-4 shows a flow diagram for the basic analytical methodology that should be used to solve any problem.

The first step in troubleshooting is to identify the problem. Typically, this consists of a description of the problem that lists the symptoms and possible solutions. The next step is to collect information about the problem. This data gathering includes questioning of the problem reporter for more detailed information, viewing physical symptoms, and reviewing the components and systems involved.

The third step is to narrow down the problem. If the problem involves complex, interactive components, try to narrow it down to the component that is causing the problem. Once the problem has been identified, apply a troubleshooting method or methods to solve the problem. If successful, correct the component; if not, return to the appropriate step and start again.

Troubleshooting Methods

There are many different methods of troubleshooting and each has its advantages and disadvantages. The approach used is often chosen due to a personal preference or convenience, but in some cases it is dictated by

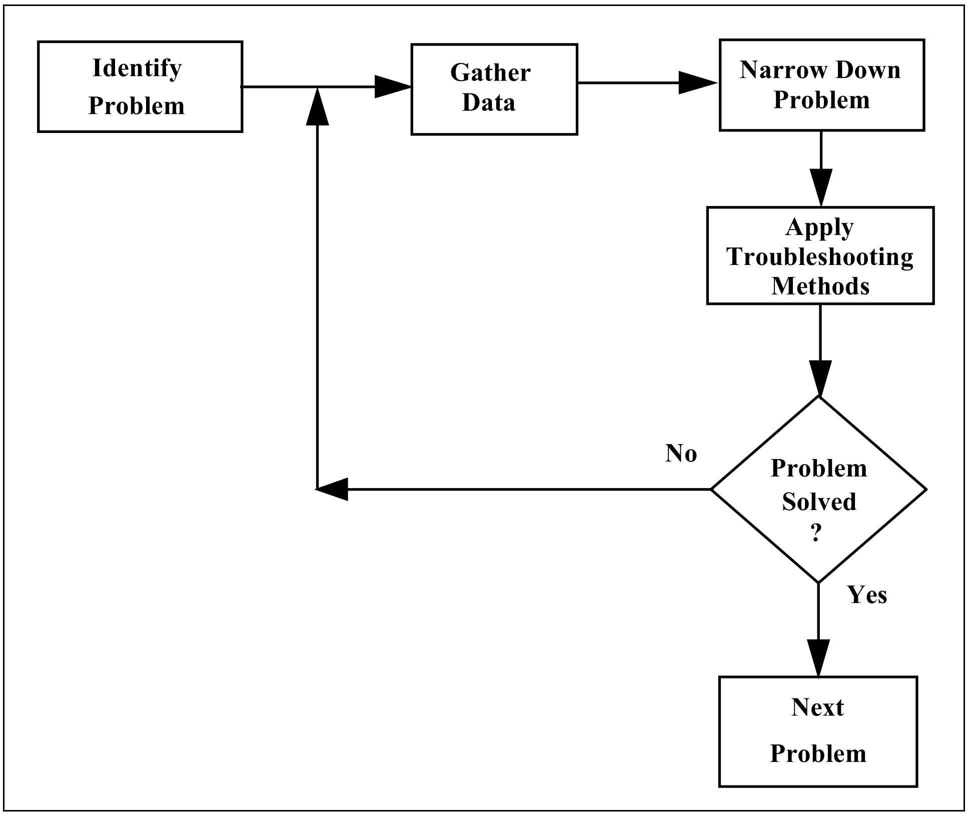


Figure 12-4. Troubleshooting Flowchart

the problem itself. In most cases, more than one method will be used or sometimes even intermixed.

We will discuss the following seven main troubleshooting methods: (1) experience, (2) process of elimination, (3) divide and conquer, (4) remove and conquer, (5) substitution, (6) setting a trap, and 7) freezing control sequence.

Experience

Using your experience is the most common troubleshooting method and many times the simplest. You know the problem and its solution because you have seen or heard of it before. If there are a number of possible solutions, experience can be used to find the area to attack first. The selection can be based on the highest probability, the one with the least risk or upset, the easiest, the closest, the most easily testable, etc.

Experience is primarily an on-the-job learned skill: the more work you've done, the more experience you have. Experience can also be gained with hands-on laboratory and classroom training, but if this training is not reinforced, it may soon be forgotten. Experience can be greatly enhanced if the troubleshooter develops the skill to extrapolate his or her experience based on a wider range of problems. This method can sometimes have the disadvantage of knowing what are causing the symptoms, but not necessarily knowing why the problem is occurring.

Experience can also be formalized by keeping good maintenance records, but this requires the ability to search and find appropriate information.

Process of Elimination

This troubleshooting method is a simple question and answer procedure. It starts by asking the following questions:

- What is the problem, and what is it not?
- Where is the problem, and where is it not?
- When is the problem, and when is it not?
- What has changed, and what has not?

This technique uses observation, experience, and testing to narrow down the problem into a more workable form. This method is not always linear; that is, the next troubleshooting step may not depend on the prior step.

Divide and Conquer

This method is commonly taught in technical schools and courses. It consists of dividing or breaking the system down into two parts, testing, and finding out which is working properly and which is not. The part that is not working is further divided and testing is applied again until the cause of the problem is obtained, as shown in Figure 12-5.

The secret to this method is knowing where and how to divide the system. The dividing point may be based on experience, ease of access, test points, etc. When in doubt, divide it in half. This method works well in PLC systems because you can easily divide the system components into field devices, I/O modules, CPU, or control software.

Remove and Conquer

This method works best with loosely coupled systems. An example might be several programmable logic controllers (PLCs) connected on a communication network. If data is being corrupted, removing each PLC in turn may help determine if one of the PLCs is causing the problem.

In complex systems, many discrete modules and electronic circuit cards are called on to do different and/or similar functions. A method to solving

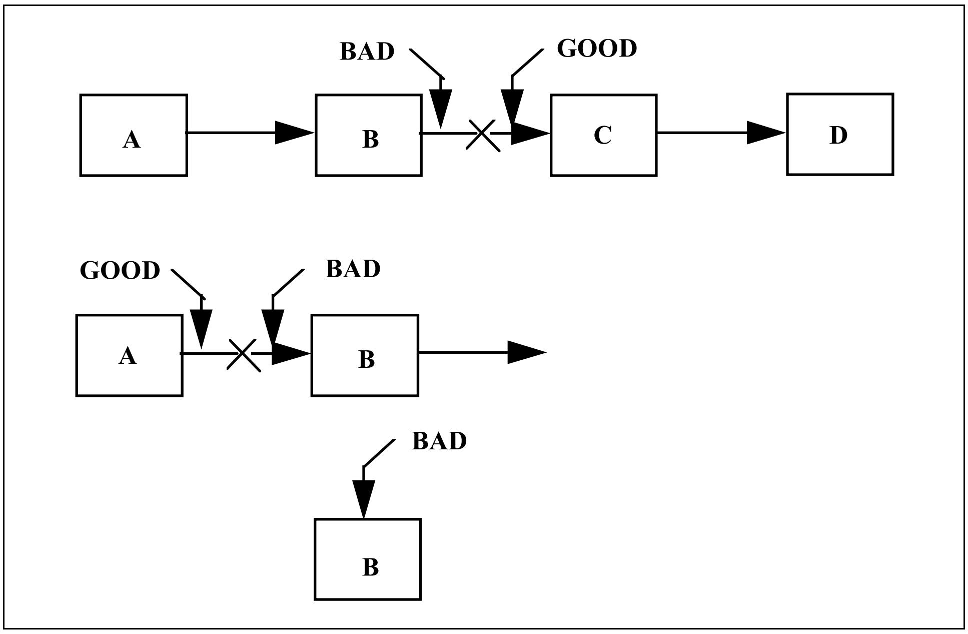


Figure 12-5. Divide and Conquer Troubleshooting Method

problems in these types of systems is to remove the modules or cards one at a time from the system and see if the problem goes away. Alternatively, remove all the modules or cards and add them back in until the problem comes back. This method is very effective in modular PLC I/O systems. A block diagram to illustrate this method is given in Figure 12-6.

Substitution

This troubleshooting method consists of substituting a known good component into the system to see if the problem goes away. This is typically useful on complex, black-box systems. Experience and testing can limit the number of boxes to be substituted.

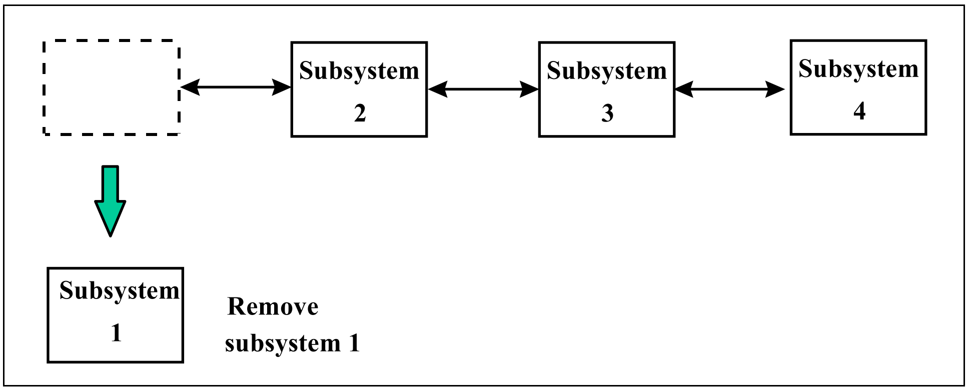


Figure 12-6. Remove and Conquer Troubleshooting Method

A good conceptual understanding of the system being troubleshot is critical to the success of this method. It is also the method of last resort for subtle or intermittent problems. A block diagram to illustrate this troubleshooting method is given in Figure 12-7.

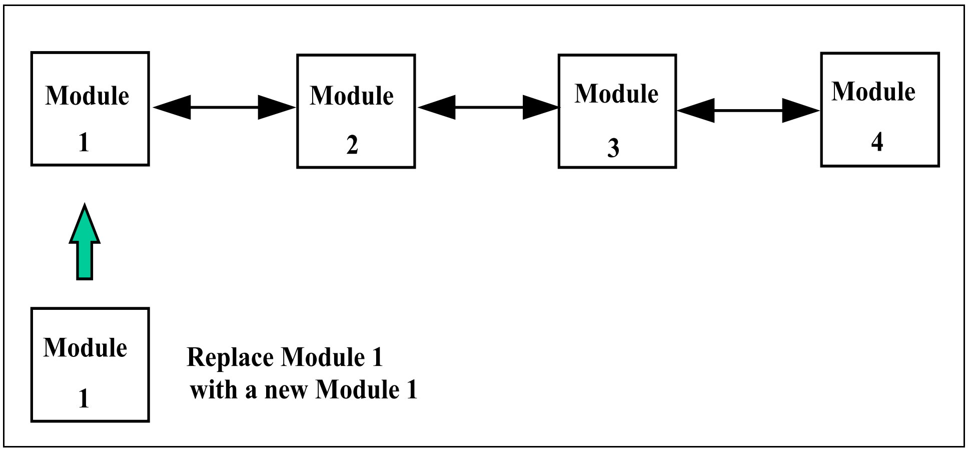


Figure 12-7. Substitution Troubleshooting Method

Setting A Trap

Setting a trap is often the only effective method for catching the cause of a spurious or transient problem. With the advanced PLC systems that include data logging, archiving, and trending, much more data is available than in the past. Even so, this sometimes is not enough and additional monitoring points or traps are needed to catch the "prey" in question. This method is very effective in software troubleshooting where the program can be halted if a certain condition or conditions are detected.

Freezing Control Sequence

Freezing a process or machine sequence when a problem is detected is an effective method that can be used in the troubleshooting of PLC control systems. When the sequence is stopped in its tracks, the user or control engineer can view both the faulted sequence logic and the faulted machine or process. The determination of the cause of the problem is easier if you can see the state of each component in the process or machine that might be causing the problem. If the control system has an HMI installed it can be used to view the status of I/O points in the system. The diagnostic tools in the PLC programming system can also be used to view the state of a system after the operation has been stopped by a fault. Freezing of a machine or conveyor system in a given state is generally easier than stopping a chemical process at a given stage of a process.

PLC System Malfunctions

PLC processors are the most reliable devices in a PLC system and they require little scheduled maintenance, except for backup battery replacement. Although the processor is rarely the source of a system failure, malfunctions can occur.

A defect anywhere in the control loop can result in a system failure. The malfunction can occur in the PLC processor, in the I/O modules, in field devices or instruments in the loop, or in the communication lines or wires between them. Failures are more likely to occur in the I/O modules or the field devices or field sensors. They are less likely to occur in the PLC processor or the software of the PLC.

I/O faults result in an interruption of the signal between the controlled equipment or machine and the PLC processor. An input module failure will result in improper transfer of the incoming signal to the processor and a malfunction of an output module will result in the improper transfer of a control signal to a field device. A processor malfunction is normally indicated by a failure of a large part of the control system. Problems can also occur in the power that provides the operating current and voltages for the PLC control system. The input and output racks and the PLC processor may have individual power supplies. When a power supply does not provide the correct voltage or current, the system may shut down.

Proper attention to environmental considerations and proper wiring and grounding can help to avoid system malfunctions. Although PLC circuits and components are environmentally hardened, excessive humidity, high vibration, high ambient temperatures, and airborne contaminates can cause system failures. Proper signal and power cable routing is important to minimize electrical noise and proper grounding and shielding protects the signal transmission and avoids a path for induced electrical noise.

PLC System Troubleshooting

When a system failure occurs, the technician must quickly and accurately determine the most likely cause of the malfunction. Using the troubleshooting methods discussed in the last section assists in isolating problems, identifying the defective components, and allows for the correction of the problem quickly to minimize system downtime. To troubleshoot effectively, it is important to examine the system as a whole and to consider all possible explanations for a malfunction.

PLC systems operate by continuously gathering information, making control decisions based on that information, and taking control action based on those decisions. Field devices such as process sensors, pushbuttons, or switches communicate process conditions to provide the PLC processor with necessary information.

Let’s examine some typical I/O faults and the methods used to troubleshoot the problem. For example, a high-level switch on a process tank is connected to a PLC input module and when the liquid level in the tank reaches the switch, the level switch contacts are suppose to close and signal the PLC. However, when high liquid level is reached in the process tank the PLC system does not respond to the high level in the tank. Any or all of the seven troubleshooting methods: experience, process of elimination, divide and conquer, remove and conquer, substitution, setting a trap, or freezing the sequence could be used to troubleshoot the problem or a combination of methods could also be used. For example, “experience” tells us that the level switch on the tank has a history of failure, so we start by checking the operation of the level switch. Assuming the level switch is a simple dry contact switch, we can remove the two wires connected to the switch at the process tank and short them out to see if the PLC responds. If the PLC does not respond, we can eliminate the level switch as the cause of the problem by the “process of elimination”. Next, we go to the control room where the input module is located and discover that the input indicator for the level switch is still not lit. This indicates that either the field wiring or the input module is defective. The next step is to “divide and conquer” by removing the field wiring for the level switch from the input module and simulate the level switch with a jumper wire. If the input indicator still does not light up, this eliminates the field wiring. Next we “remove and conquer” by replacing the input module with a new module. This should eliminate the problem. If it does not, we next check the voltage supply connected to the input module.

If the malfunction occurs in an output module, the logic decisions of the PLC processor will not be correctly translated into actions at the final control elements. The failure might be in the output module, a fuse, or the field wiring. The output module can be damaged by power surges. An open or short circuit in an output module will cause a final control device to respond to processor commands improperly. If a signal active indicator or LED on an output module fails to light on command from the processor, this may indicate an open circuit or an output module failure. An output indicator that is always on may indicate a short in the output wiring.

When a defect cannot be found in either the input module or output module, poor electrical connections, a misalignment of the module with the backplane connector, or an open wire may be the cause of the failure.

An example problem can be used to illustrate the troubleshooting of a PLC output signal fault.

EXAMPLE 12-2

Problem: Analyze an output circuit where a PLC control system is commanding an electrical solenoid valve to turn on but the solenoid valve is not turning on. Assume that a 120-VAC output module is used to turn on the solenoid valve and the indicator for the output signal on the PLC output module connected to the solenoid valve is turned on.

Solution: The fact that the output signal indicator is "on" indicates that the command from the PLC processor is properly activating the output module point. It also means that the supply voltage to the output module is functioning properly. A first step in the troubleshooting process might be to remove the wire connected to the solenoid valve from the output module and directly applying 120 VAC to the solenoid. If the solenoid valve does not turn on, the problem is with the wiring to the solenoid valve or the valve itself. Next go to the solenoid valve and apply 120 VAC directly to the valve. If the valve does not turn on, replace the solenoid valve. If the valve activates, test the wiring and electrical connections between the PLC output module and the solenoid valve.

A PLC processor malfunction is normally indicated by a failure of a large portion of the control system. The processor controls and executes the overall PLC control functions. It directs the control systems logic and decision-making functions.

The system software for most PLC processors contains a self-diagnostic program that detects processor errors and failures. The processor malfunction may be an error in programming, a memory failure, a mathematic calculation overrun, a program time-out, or a processor scan overrun. A programming or system configuration error can cause the processor to send a discrete logic signal or analog signal to the wrong I/O address. Since the processor identifies each field device by its I/O address if the signal transfers to or from the incorrect address, it will not be recognized and the control loop will not respond.

Signal transmission between the processor and memory may be interrupted or distorted by electrical interference or poor connection. Electrical noise or bad connections can change the value of the signal and cause it to be misinterpreted by the processor. Electrical interference and poor connections can also affect signal transmission between subsystems in the PLC system.

If the processor sends a request for communication with I/O module racks or peripheral devices and does not receive a response within a set time interval, it will time-out. Some processor will issue a time-out error message and continue; others will shut down.

Good programming practice requires organizing the logic into functional blocks or subroutines to reduce PLC processor scan time and processor task loading. A process or machine that suddenly requires an excessive number processor task to run at the same time might cause the processor to go into overrun. A processor overrun is a condition that exists if the amount of time required to perform all tasks is greater than the allocated time.

Malfunctions can also occur in the system power supplies. When a power supply does not provide the proper voltage or current, the PLC system may shut down. Overloads that occur when the system load has been miscalculated or the I/O modules or field devices draw too much current can cause the PLC system to shut down.

The most likely cause of a lack of power to a field device is a blown fuse in the output module. It is important to know where all the fuses are in a PLC system. Some fuses may be in field termination panels or hidden on circuit cards in electronic devices.

Field device failures are the most common cause of a PLC control system malfunction. Input devices, such as dry contact switches or analog field instruments, may fail to transfer proper signals to the processor.

Control System Documentation

Proper documentation of the hardware and software reduces design, fabrication, testing, operating, and maintenance costs of control systems. Documentation of complex control systems generally consist of the following: 1) software program listings, 2) backup copies of control programs and configuration software, 3) operations and maintenance (O&M) manuals, 4) system architecture drawings, 5) input/output wiring diagrams, 6) instrument loop diagrams, 7) process and instrument drawings (P&IDs), 8) equipment layout drawings, and 9) mechanical flow diagrams (MFDs).

The system design drawings should contain a detailed list of the components in the system to allow the maintenance department to order and stock replacement parts. The parts list should include the item number cross-referenced to the component on the drawing, a detailed description of the part, the manufacturer name, and the part number.

The facility should have a configuration control system to maintain all drawings and documentation in an "As-Built" state. The first step is to verify that all documentation for a control system is accurate and complete during start-up and commissioning of a system. After the system is placed in use, the configuration control system must keep track of all changes and update the documentation on the installed systems.

EXERCISES

12.1 Discuss the reasons and advantages for placing programmable controller components in metal enclosures.

12.2 Discuss the need for designing control panels to meet the heat requirements of programmable controller system components.

12.3 List some important design practices used to reduce maintenance costs on programmable controller systems.

12.4 Discuss the importance of proper grounding techniques in programmable controller design.

12.5 List the various phases of a typical control system operational test procedure.

12.6 Discuss the important features of an effective preventive maintenance program for programmable controller systems.

12.7 Discuss several recommended maintenance measures that should be used on PLC control systems.

12.8 Discuss the common troubleshooting methods used in finding and correcting problems in PLC control systems.

12.9 List the typical PLC processor malfunctions.

12.10 What is the most common cause of power failure in a PLC system?

12.11 List the typical types of documentation used on complex control systems.

12.12 Design a power distribution system for the two-tower dehydration PLC control system discussed in Chapter 11. Assume the system is mounted in a control panel with a 6-socket AC outlet power strip, two cooling fans, and a single fluorescent light.

BIBLIOGRAPHY

1. *Assembly and Installation Manual PLC5 Family Programmable Controllers*, Allen- Bradley, Publication 1785-6.6.1, November 1987.

2. *National Electrical Manufacturers Association (NEMA)*, Enclosures for Electrical Equipment (1000 Volts Maximum), NEMA Standard Publication 250-2003.

3. *Troubleshooting PLCs Instructor's Guide, Produced* by Industrial Training Corporation for ISA, 1993.

Appendix A

Answers to Exercises

Chapter 1

1.1 The processor reads the inputs, executes logic as determined by the application program, performs calculations, and controls the outputs accordingly.

1.2 The main purpose of the I/O system is to provide the physical connection between the process equipment and the PLC system.

1.3 Some typical discrete input devices found in process industries are process switches for temperature, flow, and pressure.

1.4 Some typical discrete output signals encountered in industrial applications are control relays, solenoid valves and motor starters.

1.5 Some typical analog signal values found in process control applications are 4- to 20-mA DC, 0 to 10 VDC, and 1 to 5 VDC.

1.6 The difference between volatile and nonvolatile memory is that volatile memory will lose its programmed contents if all operating power is lost or removed. Nonvolatile memory will retain its data and program even if there is a complete loss of operating power.

1.7 The common applications for personal computers in programmable controller systems are programming, HMI, and data collection.

1.8 The personal computer is the most common device used to program PLCs.

1.9 The three basic instructions used in ladder logic programs are normally open, normally closed, and coil output.

1.10 Power flow in a relay ladder diagram is the flow of electric current from the left side to the right side of a ladder rung.

1.11 Logical power flow is obtained in a ladder diagram program if there is logical continuity from the left side to the right side of a ladder rung.

1.12 IL instruction statements have two basic structures. One, a statement made up of an instruction alone (e.g., NOT) and another where the statement is made up of an instruction and an address.

1.13 The most common methods used to produce a visual representation of a process or machine under control are as follows: 1) wire directly from the programmable controller I/O modules to hard-wired lights and digital indicators on a display or control panel or 2) to use personal computers with Human Machine Interface (HMI) software that generate a visual representation of the process or machine being controlled.

1.14 The first method is a system that consists of an industrial grade PC running under Windows operating software and connected to standard PLC input and output modules. The second method is a software system called SoftPLC® that converts an industrial PC (x86 based) into a full-function PLC-like process controller.

1.15 The main advantage of a SoftPLC system is that it is an open architecture, non-proprietary system, so the user can easily interface to other vendor control devices and also easily connect to PC networks and other industrial networks.

Chapter 2

2.1 $1010_2 = 10_{10}$.

2.2 $10011_2 = 19_{10}$.

2.3 $10101011_2 = 253_8$.

2.4 $101111101000_2 = 5750_8$.

2.5 $33_{10} = 41_8$.

2.6 $451_{10} = 703_8$.

2.7 $47_{16} = 71_{10}$.

2.8 $157_{16} = 343_{10}$.

2.9 $56_{10} = 38_{16}$.

2.10 BCD code is 0011 0111.

2.11 BCD code is 0010 0111 0000.

2.12 The ASCII code is 4C 65 76 65 6C 20 4C 6F 77 in Hex format.

2.13 To verify the logic identity $A + 1 = 1$, we let B =1 in the two input OR truth table shown below.

Inputs		Output
A	*B*	*Z*
0	1	1
1	1	1

Since B = Z = 1 in the truth table, then A + 1 = 1.

2.14 Let Start function = A, the Stop function = $\overline{B}$ and the Run Request = C, so the logic equation is: $(A + C) \bullet \overline{B} = C$.

2.15 Let tank high level = A, the tank low level = B and the Pump Running = C, so the logic equation is: $(\overline{A} \bullet C) \bullet (B + C) = C$.

2.16 The logic gate circuit for a standard Start/Stop circuit is shown in Figure 2-16e.

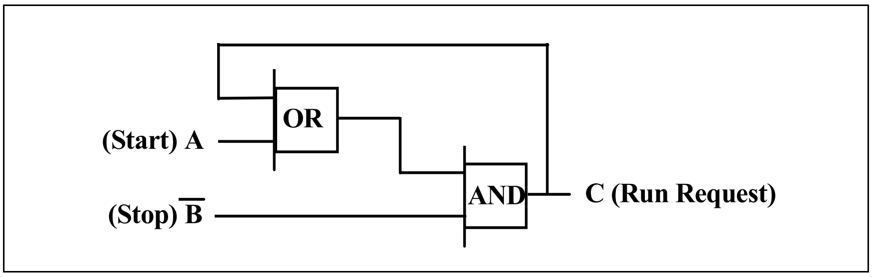


Figure 2-16e. Answer to Exercise 2.16

2.17 The control actions that will stop conveyor one are as follows: 1) activating the overloads on starter coil M2; 2) depressing any one of the stop buttons (PB1, 2, or 3); and 3) depressing the e-stop (PB1).

Chapter 3

3.1 The current flow is 15 amps.

3.2 The current is 4.8 amps.

3.3 The area is 9 cmil.

3.4 The resistance of 500 feet of copper wire having a cross-sectional area of 4110 cmil is 1.26 Ω.

3.5 The resistance of 500 feet of silver wire with a diameter of 0.001 inch is 4900 Ω.

3.6 The distance to the point where the wire is shorted to ground is 1238.2 ft.

3.7 The total current (I_t) in a series circuit with two 250 Ω resistors and an applied voltage is 24 VDC is 48 mA. Also, V_1 = 12 volts and V_2 = 12 volts.

3.8 I_1= 1 A, I_2 = 0.5 A, and I_t = 1.5 A

3.9 I_1 = 1 mA, I_2 = 5 mA, R_x = 400 W, V_1 = 2 volts, and V_2 = 10 volts.

3.10 First, you can increase the amount of electrical current by increasing the applied voltage. Second, you can increase the number of turns of wire in the coil. Third, you can insert an iron core through the center of the coil.

3.11 One purpose of a transformer is to transfer power from the primary to the secondary. A second purpose is to isolate the primary circuit from the secondary load. A final function is to step up or step down the input AC voltage to provide a different value of AC voltage to the output load.

3.12 It is easier to filter the output of a full-wave rectifier than it is to filter the output of a half-wave rectifier.

3.13 Electric relays operate as follows: when a changing electric current is applied, a strong magnetic field is produced and the resulting magnetic force moves the iron core that is connected to a set of contacts. These contacts in the relay are used to make or break electrical connections in control circuits.

3.14 The three common types of solenoid valves are direct acting, internal pilot-operated, and manual reset.

Chapter 4

4.1 The input/output (I/O) system provides the physical connection between the process equipment and the PLC. It uses various interface circuits to convert the sensed or measured physical quantities of the process, such as motion, level, temperature, pressure, and position, to a logic signal to be used by the PLC processor. Based on the status of the sensed or measured values, the control program in the PLC processor uses various output circuits or modules to activate devices, such as valves, motors, pumps, and alarms, to exercise control over a machine or process.

4.2 The backplane on a PLC I/O rack is a printed circuit card that contains the parallel communication lines to the processor and the DC voltages that are required to power the logic and interface circuits in the I/O modules.

4.3 First, I/O circuits are packaged in groups of 4, 8, 16 and 32 in points in industrial grade modular units. These modules are then mounted in rugged equipment housings. In the second configuration, a standard din-mounting rail is used to hold point type I/O blocks. Finally, in the case of small PLCs, the I/O circuits are an integral part of the PLC.

4.4 Discrete inputs signals are on/off or open/closed type signals from field devices. Five typical examples of discrete input signal devices are pushbuttons, pressure switches, relay contacts, level switches, and temperature switches.

4.5 Five examples of typical discrete output field devices are alarm lights, electric control relays, motor starters, electric valves and alarm horns.

4.6 The purpose of the bridge rectifier in the AC input circuit shown in Figure 4-2 is to convert the AC input voltage to a DC voltage signal to be used by the PLC.

4.7 The purpose of the optical coupler in the AC input circuit shown in Figure 4-2 is to provide the electrical isolation between the input AC voltage and the DC voltage used by the logic section of the PLC.

4.8 The operation of the typical AC output module shown in Figure 4-5 is as follows: first, the processor sends out a logic signal of 0 or 1 to the logic section of the AC output circuit, the signal from the logic section is then passed through an isolation circuit. The logic signal from the isolation circuit is finally fed to an AC power switch circuit and a filter section to finally control the AC operated field device.

4.9 The wiring diagram for this exercise is shown in Figure 4.9e.

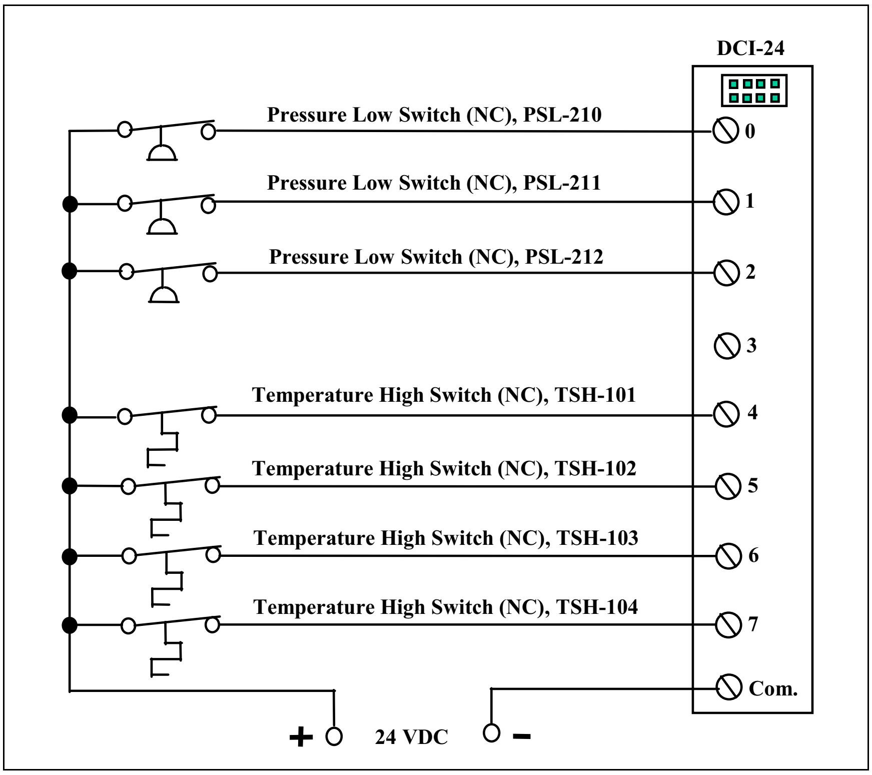
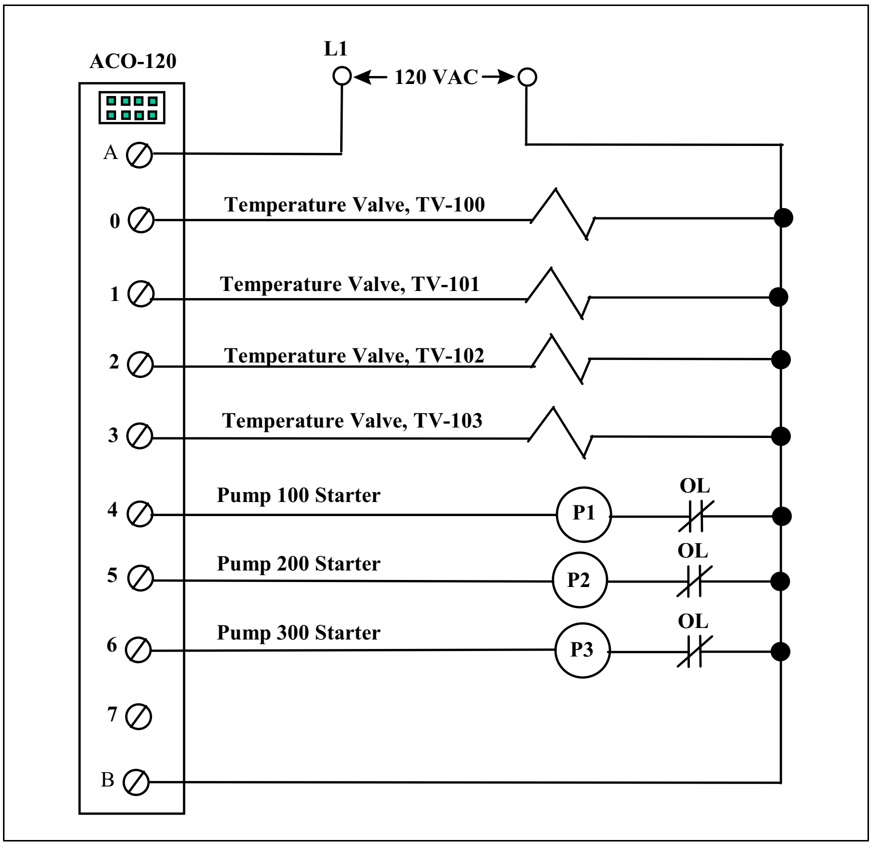

DCI-24
Pressure Low Switch (NC), PSL-210
Pressure Low Switch (NC), PSL-211
Pressure Low Switch (NC), PSL-212
Temperature High Switch (NC), TSH-101
Temperature High Switch (NC), TSH-102
Temperature High Switch (NC), TSH-103
Temperature High Switch (NC), TSH-104
0
1
2
3
4
5
6
7
Com.
+
24 VDC
–

Figure 4-9e. Answer to Exercise 4.9

4.10 The wiring diagram for a typical +120 VAC output module connected to four solenoid valves (TV-100, 101, 102, and 103) and three pump starters (P-100, 200, and 300) with overload switches is shown in Figure 4.10e.

4.11 A four-wire RTD input operates as follows, when a small current from the RTD module is applied to the field-mounted RTD, it creates a voltage that varies with temperature. Then, this voltage is processed and converted by the RTD module into a temperature value that can be used and displayed by a process operator.

4.12 The backplane current requirements for a 16-slot I/O rack with the following modules installed: (a) four 12-VDC input modules (model number DCI-12), (b) six 12-VDC output modules (model number DCO-12), (c) four analog input modules (model number AI-4-20MA), and (d) two analog output modules (model number


ACO-120
L1
120 VAC
A
0
1
2
3
4
5
6
7
B
Temperature Valve, TV-100
Temperature Valve, TV-101
Temperature Valve, TV-102
Temperature Valve, TV-103
Pump 100 Starter
Pump 200 Starter
Pump 300 Starter
P1
P2
P3
OL
OL
OL
Figure 4-10e. Answer to Exercise 4.10

AO-4-20MA) using back-plane current values listed in Table 4-5 is calculated as follows:

DCI-12: 200 mA,
DCO-12: 200 mA,
AI-4-20MA: 400 mA, and
AO-4-20MA: 400 mA.

So that, the total backplane current required is

```
4 x 200 mA =    800 mA
6 x 200 mA =   1200 mA
4 x 400 mA =   1600 mA
2 x 400 mA =    800 mA
               4400 mA
```

Chapter 5

5.1 The memory types are Read Access Memory (RAM), Read-Only

Memory (ROM), Programmable Read-Only Memory (PROM), and Electrically Erasable Programmable Read-Only memory (EEPROM).

5.2 Read/Write (R/W) memory is designed so that data or information can be written into or read from any unique location. Data or programs are placed into R/W memory using the write mode and data or programs can be retrieved from R/W memory using the read mode. A typical application for R/W memory is to store a PLC control program.

5.3 The main purpose of Read-Only Memory in PLC applications is to store the executive or operating system program of the PLC.

5.4 The most common integrated circuit memory devices used in PLC applications are RAM and ROM chips.

5.5 The most common magnetic memory-storage devices used in PLC control applications are floppy disk drives and hard disk drives.

5.6 There are two main parts of a typical programmable logic controller memory: the system memory and the application memory. The *system memory* is a permanently stored collection of programs and registers that consists of the operating system program, diagnostics software, and system status registers. The operating system directs activities such as execution of the control program, communication with the peripheral devices, and other system housekeeping functions. The *application memory* consists of the input area, output area, data or information storage registers, internal storage bit area, and the control program.

5.7 The Input and Output Image Tables are used to store the I/O bits that are used by the PLC control program.

5.8 The memory size required for a programmable controller system with 235 input points and 195 outputs and assuming 25% spare memory capacity is 10,750 words.

5.9 The memory bit address for a discrete field device connected to an Allen-Bradley PLC5 input module in rack 1 at I/O group 3 and terminal 1 is I:013/01.

5.10 The memory bit address for a discrete field device connected to an Allen-Bradley PLC5 output module in rack 2 at module group 1 and terminal 7 is O:021/07.

5.11 The memory bit address for a discrete field device connected to an Allen-Bradley PLC5 output module in rack 2 at I/O group 2 and terminal 13 is O:022/13.

5.12 The bit addresses available for use by a 16-bit discrete output module installed in slot 4 of rack 1 are Q32.0 through Q32.7 and Q33.0 through Q33.7.

5.13 The bit addresses available for use by a 16-bit discrete input module installed in slot 5 of rack 2 are I68.0 through I68.7 and I69.0 through I69.7.

Chapter 6

6.1 The three main PLC programming languages are ladder diagram, Instruction list, and functional block diagram.

6.2 The unlatch coil instruction is used to reset a latched output instruction with the same referenced address.

6.3 The ladder diagram program to start a pump that fills a process tank with fluid until a high level is reached in the tank is shown in Figure 6-3e below. The program assumes that the fill tank pushbutton (PB1) is wired to an Allen-Bradley PLC5 discrete input module at bit I:010/00 and a NC tank high-level switch is connected to input bit I:010/01. The program also assumes that the pump start relay is connected to output O:000/01 on the A-B PLC.

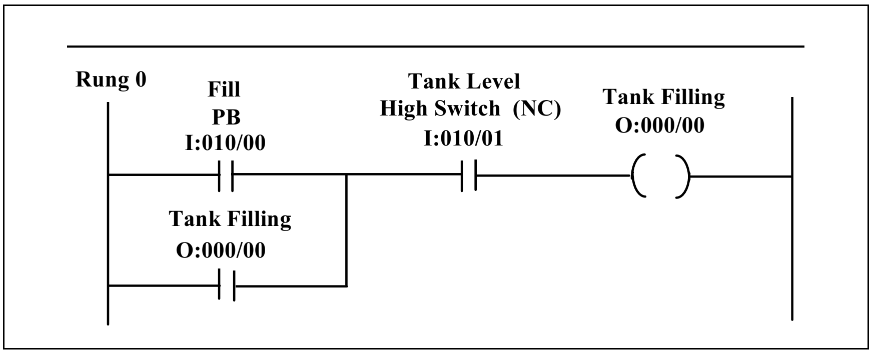


Figure 6-3e. LD Program - Answer to Exercise 6.3

6.4 The ladder diagram program to control an electric motor is shown in Figure 6-4e. The program assumes that a normally opened start pushbutton is wired to an input module at I:000/00 and a normally closed stop pushbutton is connected to the same input module at address I:000/01. The program also assumes that PLC5 output O:001/03 is connected to the motor starter and that the normally opened auxiliary contacts on motor start contactor are connected to PLC input I:000/02.

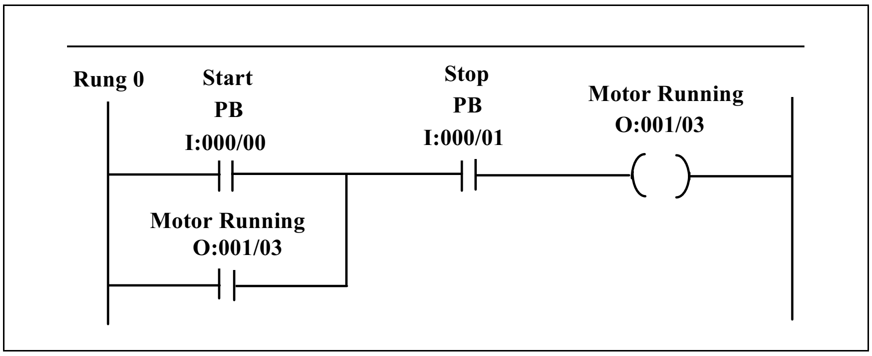


Figure 6-4e. Start/Stop LD program - Answer to Exercise 6.4

6.5 The LD program to control a process pump is shown in Figure 6-5e. The program assumes the following conditions: the pump is turned on 5 seconds after both the inlet and outlet valves to the pump have been opened and the pump is turned off, if either the inlet or the outlet valves are closed. The program also assumes: a pump starter is connected to output bit O:001/00, a valve open position switch on the inlet valve is wired to input I:000/02, and a valve open position switch on the outlet valve is wired to input I:000/03.

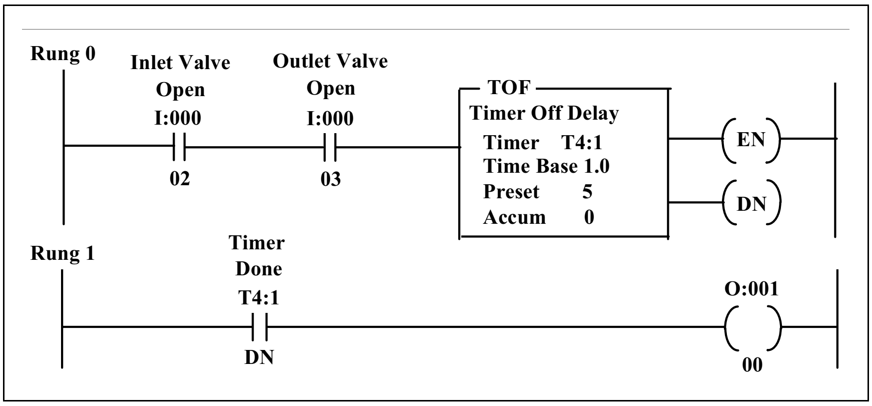


Figure 6-5e. Answer to Exercise 6.5

6.6 The LD program to open the outlet valve (LV-1) on process tank of Figure 6-6e, the program assumes that if the normally opened level switch (LSH-1) closes when the automatic position of the HOA (HS-1) is selected or if the operator places the HOA switch in the hand position then valve LV-1 will be opened. The program also assumes that the high-level switch is connected to input I:002/01, the output valve is connected to output O:003/01, the "Auto" position of the HOA switch is connected to input I:002/02, and the "Hand" position of the switch is wired to input I:002/03 on a A-B PLC5.

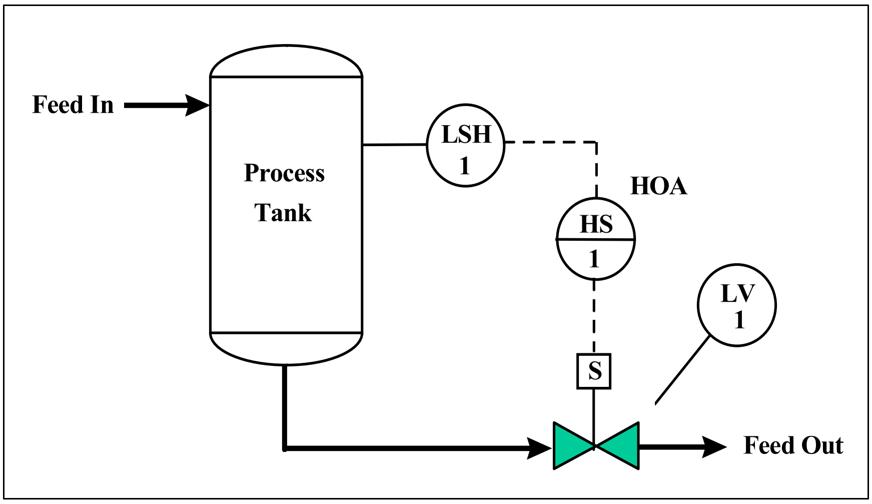


Figure 6-6e. Process Tank Application for Exercise 6.6

6.7 The LD program using timer instructions to flash two process alarm lights on a control panel is shown in Figure 6-7e. The two alarm lights are driven by PLC output signals O:010/00 (alarm 1) and O:010/01 (alarm 2). The program assumes that alarm 1 is activated by input I:000/01 and alarm 2 is activated by input I:000/02.

6.8 The ladder diagram program to control the temperature of the fluid in a process tank close to 400°C is shown in Figure 6-8e. The program assumes that the heater contactor is connected to PLC5 output O:001/01, a temperature low switch set at 395°C is connected to input I:000/00 and temperature high switch set at 405°C is connected to input I:000/01. The temperature low switch is closed if the fluid temperature is below 395°C and opens at a temperature of 395°C or higher. The temperature high switch is closed if the fluid temperature is below 405°C and opens at 405°C or higher temperatures.

6.9 The LD program for an Allen-Bradley PLC5 to turn off a conveyor belt on a production line after 50 parts have been produced is shown in Figure 6-9e. The program assumes the following: 1) when output bit O:001/12 set to 1 the conveyor belt is turned on; 2) input I:002/01 changes from 0 to 1 and then back to 0 each time a

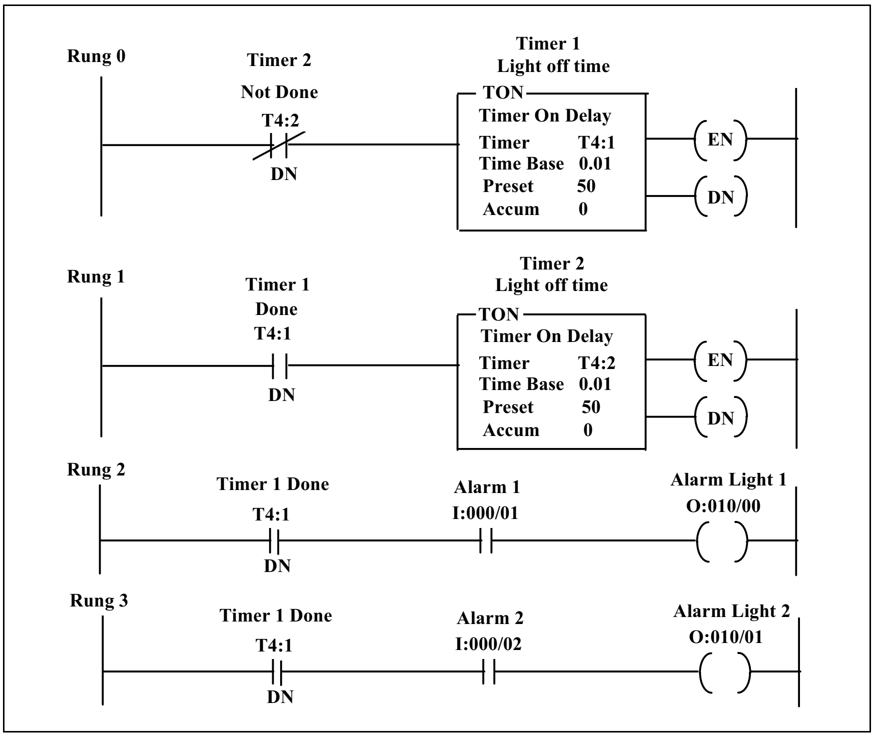


Figure 6-7e. Answer to Exercise 6.7

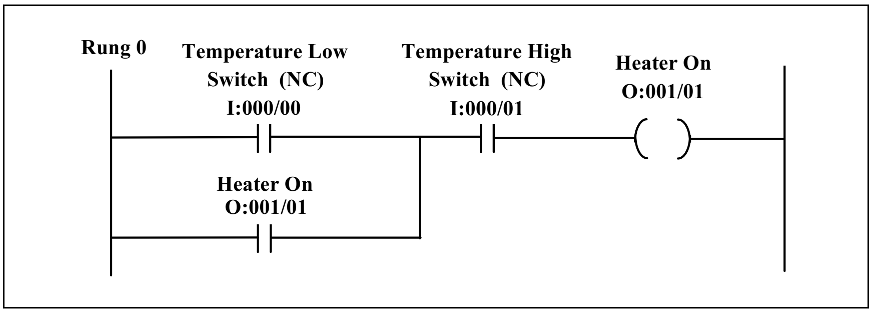


Figure 6-8e. Answer to Exercise 6.8

production part is rejected; 3) input I:002/02 changes from 0 to 1 and then back to 0 each time a new part is produced; 4) a normally open (NO) pushbutton connected to input I:002/03 is used to set the production count to 50, and 5) a NO pushbutton connected to input I:002/04 is used to reset the counter to zero and to stop the conveyor belt.

6.10 The A-B PLC5 program to add the integer number in word N7:2 to the integer number in word N7:4, then divide the result stored in N7:20 by 5 and store the final result in word N7:6 is shown in Figure 6-10e.

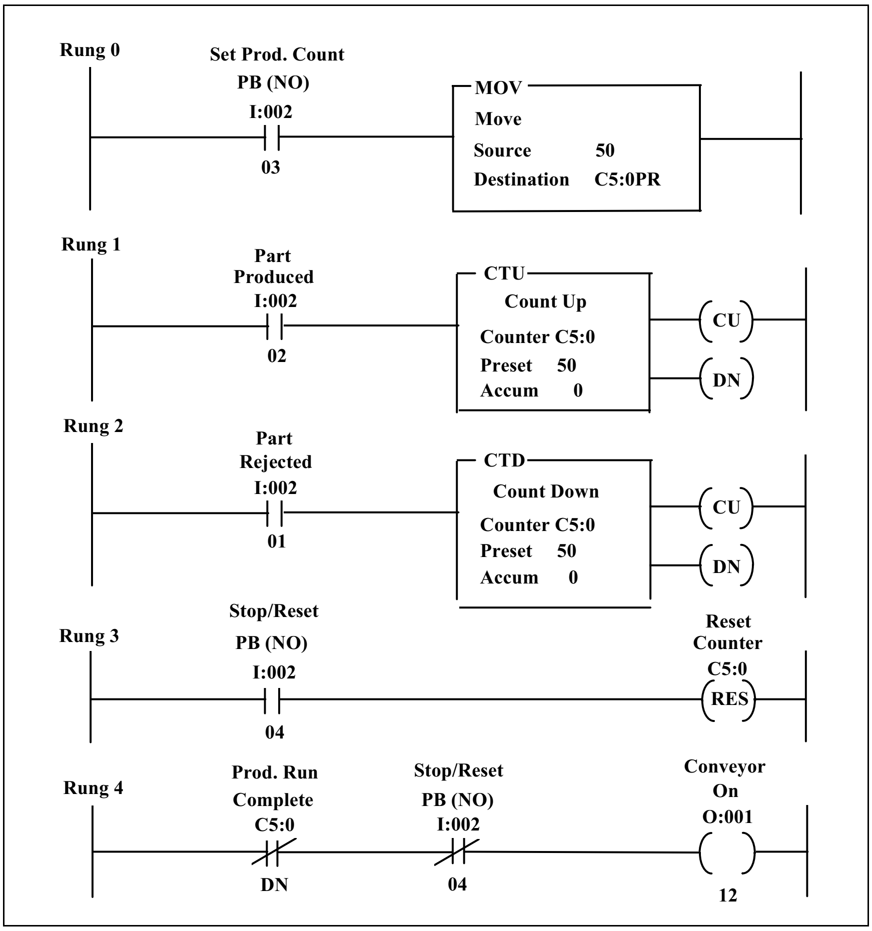


Figure 6-9e. Answer to Exercise 6.9

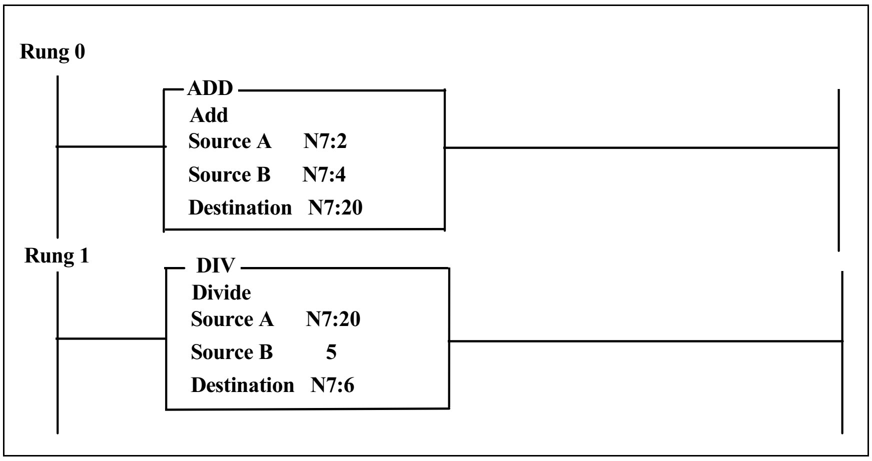


Figure 6-10e. Answer to Exercise 6.10

6.11 The LD program to transfer the integer number in word N7:0 to an output display at word N7:10, if the number in word N7:0 is greater than the number in word N7:1 and less than the number in location N7:2 is shown in Figure 6-11e.

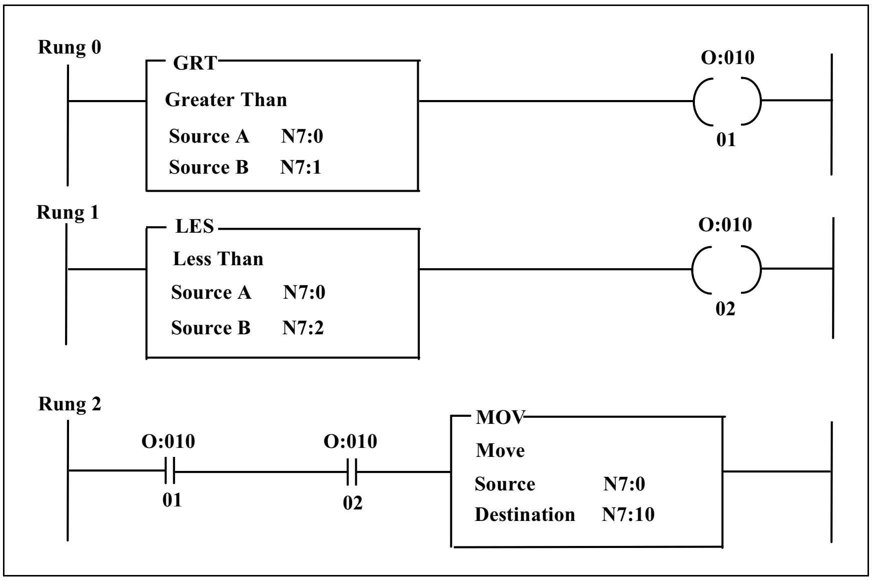


Figure 6-11e. Answer to Exercise 6.11

6.12 Subroutines are used in programming to produce a more structured program and to reduce the amount of memory used for a program. Subroutines are used to store recurring logic functions that can be accessed from different parts of the main ladder diagram program. This saves memory space because the function has to be programmed only once, but is used many times in the control application.

6.13 The two types of project files used in the Rockwell Automation RSLogix5 software package are processor and database files.

6.14 The five types of descriptive text used to document a typical PLC control program are: 1) rung comments, 2) instruction comments, 3) page titles, 4) address descriptions, and 5) symbols.

Chapter 7

7.1 The most common advanced LD instructions encountered in the Allen-Bradley PLC5 family are file, shift register, sequence and block transfer.

7.2 A file is a group of consecutive data table words used to store information in a PLC system.

7.3 The Allen-Bradley FAL instruction performs copy, arithmetic, logic, and function operations on the data stored in files. The FAL instruction is an output instruction that performs the operations defined by the source addresses and the operators listed by the programmer in the expression field.

7.4 The PLC5 LD program to copy the process data located in integer file #N30, words 0, 1, 2, and 3 to integer file #N31 starting at word 0, if input bit I:001/01 is true, is shown below in Figure 7-4e.

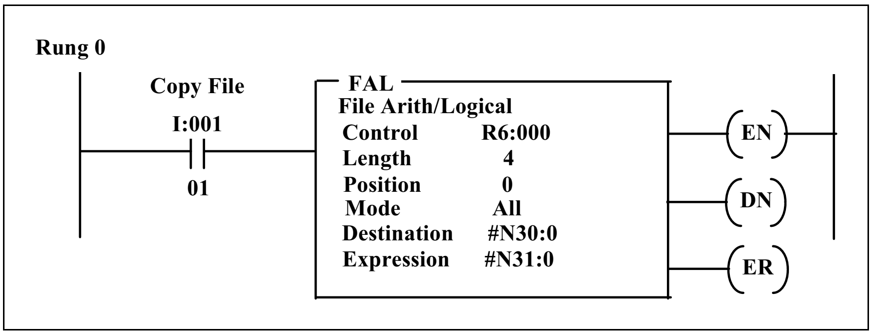


Figure 7-4e. Answer to Exercise 7-4

7.5 The Allen-Bradley PLC5 LD program to review the data in integer file #N50, words 0 through 20 and compare the data for an equal condition to the data stored in file #N60 starting at word 1 is shown in Figure 7-5e.

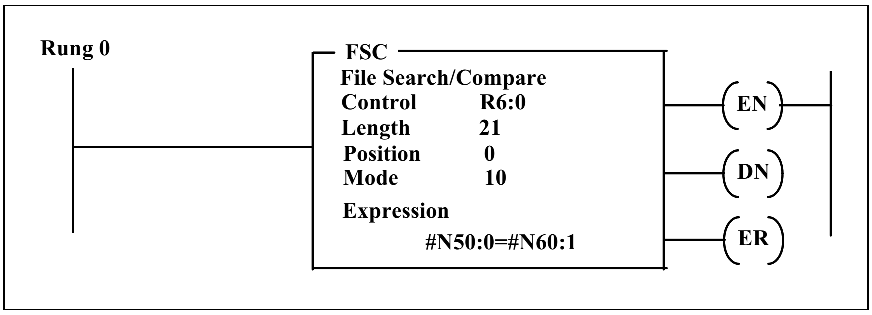


Figure 7-5e. Answer to Exercise 7-5

7.6 The *control* is the address of the control structure in a control type (R) file. The processor uses this information to run the instruction. The *length* is the number of words (0 to 999) in the data block on which the file instruction operates. The *position* is the current element within the data block that the processor is accessing. The *mode* is the number of file elements operated on each time the rung is scanned in the program. There are three modes: All, Numerical, and Incremental. In the All mode, the entire file is operated on before continuing on to the next rung of the program. The Numerical mode distributes the file operation over a number of program scans. The Incremental mode manipulates one word of the file each time the rung goes from false to true. The *destination* is the address where the processor stores the result of the operation. The instruction converts to the data type specified by the destination address. The *expression* contains addresses, program constants, and operators that specify the source of data and the operations to be performed.

7.7 Shift register instructions are used to track the movement or flow of parts and information in industrial applications.

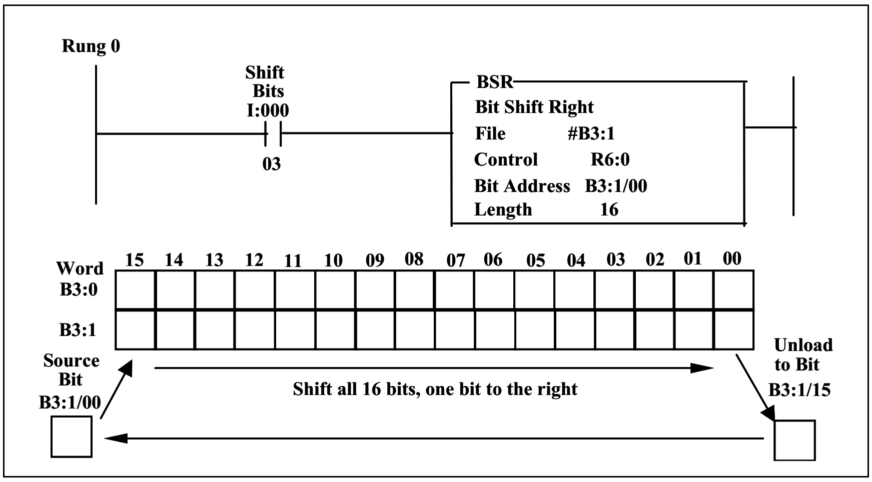
7.8 There are three common sequence instructions: sequencer input, sequencer output, and sequencer load. The sequencer instructions are generally used to transfer data from the memory to discrete output modules for the control of sequential process operations or sequential batch operations (sequencer output). They are also used to compare I/O word data with data stored in tables so that process-operating conditions can be examined for control and diagnostic purposes (sequencer input). The instruction is also used to transfer I/O word data into the memory (sequencer load).

7.9 Length is the number of words in a sequencer file.

7.10 Position is the current position of the word in the sequencer file that the processor is using.

7.11 The answer to the exercise is shown in Figure 7-11e.

7.12 The value of the mask is 00FF.



7-11e. Answer to Exercise 7-11

Chapter 8

8.1 The five PLC programming languages defined in the IEC 1131-3 standard are: 1) Ladder Diagram, 2) Sequential Function Chart, 3) Function Block Diagram, 4) Instruction List and 5) Structured Text.

8.2 The two text-based PLC languages are Instruction List and Structured Text. Instruction List, or IL, is a low-level programming language. It is very effective for small simple applications or for optimizing parts of an application. Instructions always relate to the current result and the operator indicates the operation that must be made between the current value and the operand. The result of the operation is stored again in the current result. Structure Text (ST) is a high-level structured language designed for automation processes. This language is used mainly to implement complex procedures that cannot be easily expressed with graphical languages. ST is the default language for the description of the actions within steps and conditions attached to the transitions of the SFC language.

8.3 The three graphical PLC languages are Ladder Diagram, Function Block Diagram, and Sequential Function Chart.

The Ladder Diagram is the most common programmable controller language and it consists of a set of instructions that will perform the most basic type of control functions: relay type logic, timing and counting, and basic math operations. However, depending on the programmable controller model, the instruction set may be extended or enhanced to perform other operations.

These additional functions are used for analog control, data manipulation, reporting, complex control logic, and other functions.

The Functional Block Diagram is a graphical programming language that uses logic blocks similar to those used in Boolean algebra to represent basic logic. It also uses more complex function blocks to perform operations such as timing, counting, math, loading data, transferring data, and data comparison. The programmer is able to build complex control schemes by using functions from the FBD library and then interconnecting them in a graphical diagram area.

SFC is a graphical language used to describe sequential operations. The control process is represented as a set of well-defined steps, linked by transitions. A Boolean logic condition is attached to each transition. Actions within the steps and the logic transitions between the steps can be performed by using instructions from the other standard PLC languages.

8.4 The "initial" step instruction in a Sequential Function Chart (SFC) program is used to start the program.

8.5 The "transition" instruction in a SFC program is the logic condition that the processor checks after completing the active step. When the transition logic is true, the step preceding the transition is disabled, and the step following it becomes active.

8.6 After a true transition, the processor scans the step once more to reset all timer instructions and then executes the next step.

8.7 a) valid; b) invalid (mix of integer and Boolean variable); and c) valid.

8.8 The programming tips that can be used to improve the readability of a Structured Text program are as follows: 1) do not write more than one statement on one line, 2) use tabs to indent complex statements, and 3) insert comments to increase readability of lines.

8.9 IL instruction statements have two basic structures. The first is a statement made up of an instruction alone (e.g., NOT) and the second is a structure where the statement is made up of an instruction and an address.

8.10 The IL program to control the pump is as follows:

A I124.0, A I124.2, = Q124.0

8.11 The five timer instructions available in the Siemens S7 IL programming instruction set are Pulse Timer (SP), Extended pulse timer (SE), On-delay timer (SD), Retentive on-delay timer (SS), and Off-delay timer (SF).

8.12 The IL program to open the fill valve for 30 seconds is listed below. An Extended Pulse Timer (SE) was used because the pushbutton to open the valve is a momentary type, so the "fill valve" logic signal is present only for a short time.

IL Instruction	Explanation
A I124.6	Check logic of fill tank input I124.6
L S5T#30S	Load 30-second time delay if Pump Run bit is 1
SE T3	Start timer T2 as an "Extended Pulse" timer
A I124.2	Check for bit I124.2 = 1 to reset timer
R T3	Reset timer, if I124.2=1
A T3	Check timer output status
= Q124.2	Open fill valve for 30 seconds

8.13 The IL program to control the production line is listed below:

IL Instruction	Explanation
A I124.7	Part rejected
CU C4	Count up
A I124.6	Part produced
CD C4	Count down
A I124.2	Production count input bit check
L C#10	Load production count value
S C4	Set count to 10
A I124.3	Check for bit I124.3 = 1 to reset or stop
R C4	Reset count to zero, if I124.3 = 1
A C4	Check counter output status
= Q124.5	Run conveyor if count value not zero

8.14 The IL program to perform the math operation is as follows:

IL Instruction	Explanation
L MW60	Load word MW60 into accumulator 1
L MW62	Load word MW62 into accumulator 1
+I	Add MW60 to MW62
L 10	Load integer 10 into accumulator 1
*I	Divide the result of addition by 10
T MW50	Store result in MW64

8.15 The IL program to perform the math operation is as follows:

IL Instruction	Description of Instruction
L MD50	Load value in MD70 into Accumulator 1
L MD54	Load value in MD74 into Accumulator 1, CPU transfers MD70 value into Accumulator 2
-R	Subtract MD70 from MD74
L 1.000E+01	Load 10 into Accumulator 1
/R	Divide 10 into result of (MD74 - MD74)
T MD78	Transfer results to MD58

Chapter 9

9.1 The four basic combinations of elements and boxes used in FBD programming are: 1) instructions as elements, 2) instruction as a box with address, 3) instruction as a box with address and value, and 4) instruction as a box with parameters.

9.2 The FBD program to control filling a tank is shown in Figure 9-2e. The program starts a pump that fills a process tank with fluid until a high level is reached. The program assumes that a normally opened (NO) fill tank pushbutton is wired to an input at I124.4 and a normally closed (NC) tank high-level switch is connected to input I124.5 on a Siemens PLC model CPU314IFM. It also assumes that the pump start relay is connected to output Q124.0 on the same Siemens PLC.

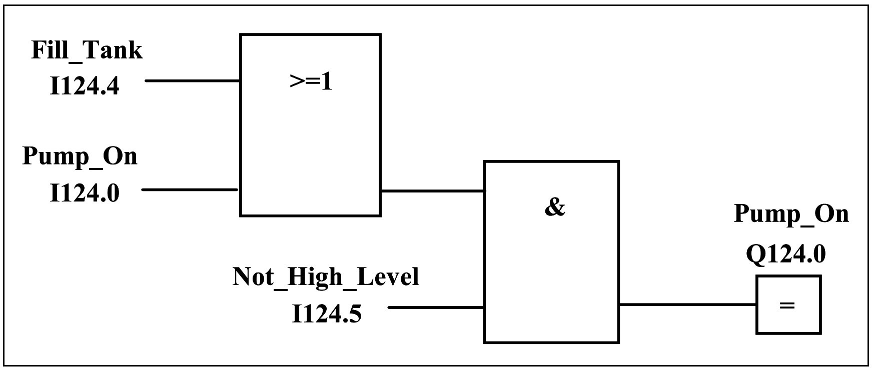


Figure 9-2e. Answer to Exercise 9-2

9.3 The FBD program to control an electric motor is shown in Figure 9-3e. This program assumes that a NO start pushbutton is wired to input I124.0 and a NC stop pushbutton is wired to input I124.5 of a Siemens PLC model CPU314IFM. Also, the program assumes that output Q124.1 of the same PLC controls a motor start relay and that the auxiliary (NO) contacts of the motor starter are connected to the PLC input I124.1.

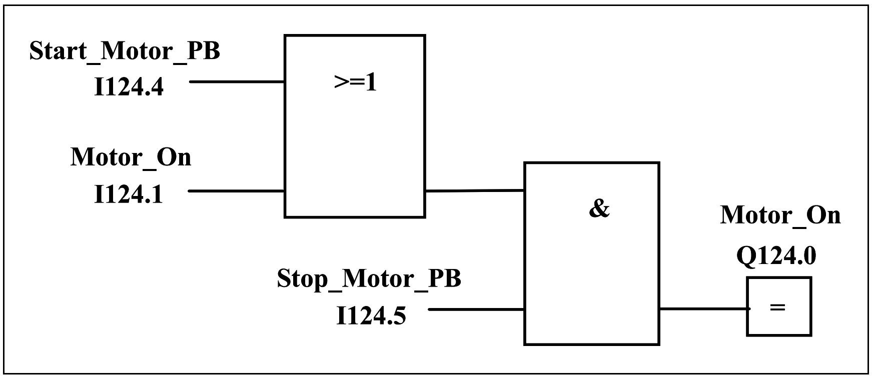


Figure 9-3e. Answer to Exercise 9-3

9.4 A typical application for a timer instruction is to delay an operation for a fixed time interval.

9.5 The five timer instructions available in the Siemens S7-300 FBD programming instruction set are: 1) Pulse, 2) Extended Pulse, 3) On-delay, 4) Retentive On-delay, and 5) Off-delay.

9.6 The FBD program to control the temperature of the fluid in a process tank is shown in Figure 9-6e. The program assumes that the heater contactor is connected to output Q124.1, a temperature low switch is connected to PLC input I124.0, and temperature high switch is connected to PLC input I124.1 on a Siemens model CPU314IFM programmable controller. The temperature low switch is closed if the fluid temperature is below 395°C and opens at a temperature of 395°C or higher. The temperature high switch is closed if the fluid temperature is below 405°C and opens at 405°C or higher temperatures.

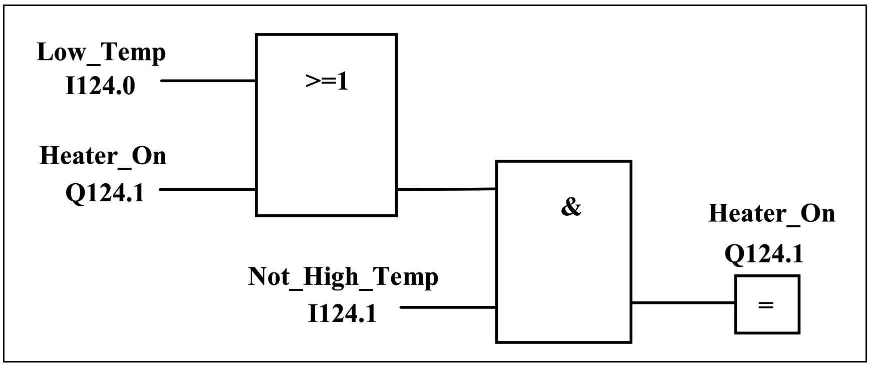


Figure 9-6e. Answer to Exercise 9.6

9.7 The FBD program to control the pump is shown in Figure 9-7e. This program assumes that a pump starter relay connected to output point Q124.0 and that if the inlet control valve FV-1 on a process tank is opened (input bit I124.0 =1) and the process tank is at a high level (input bit I124.2 = 1) the pump will be turned on.

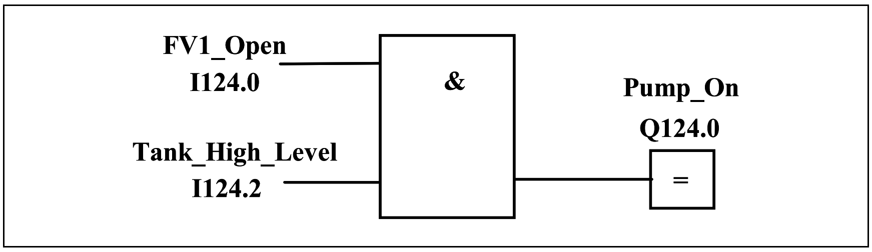


Figure 9-7e. Answer to Exercise 9.7

9.8 The FBD program for a Simatic CPU314IFM PLC to open a fill valve on a process tank to allow an ingredient to be added to the tank for 10 seconds each time the Tank Level High Switch LSH-100 is activated is shown in Figure 9-8e. Note switch LSH-100 is connected to input I124.0 and the fill valve are wired to PLC output point Q124.0.

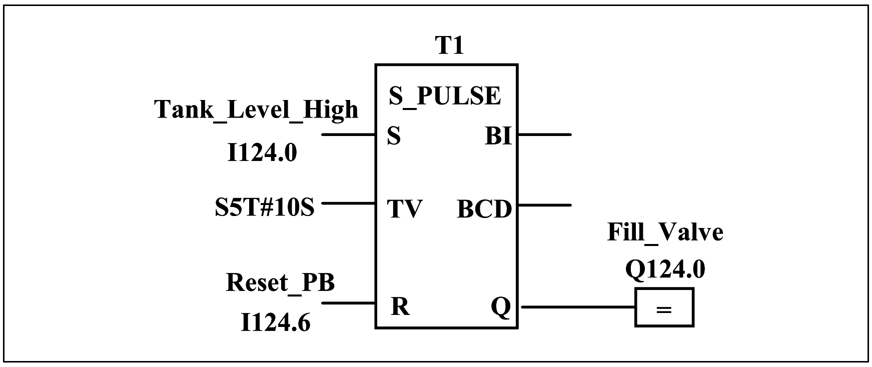


Figure 9-8e. Answer to Exercise 9-8

9.9 The FBD program to control the conveyor belt on a production line is shown in Figure 9-9e. The control program turns off the conveyor after 50 parts have been produced. The program assumes the following: 1) Output bit Q124.5 = 0, turns off the conveyor belt; 2) Input I124.1 is set to 1 each time a production part is rejected; 3) Input I124.2 is set to 1 each time a new part is produced; 4) A normally open (NO) pushbutton connected to input I124.6 is used to set the production count to 50, and 5) A NO pushbutton connected to input I124.7 is used to reset the counter to zero and to stop the conveyor belt.

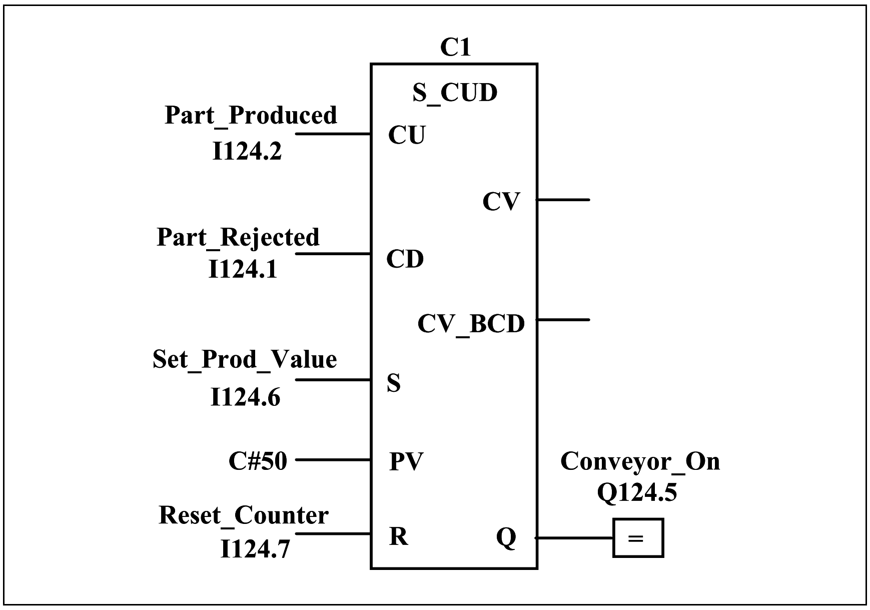


Figure 9-9e. Answer to Exercise 9-9

9.10 The purpose of FBD integer math instructions is to perform mathematical operations on positive and negative whole numbers (1,2,3, etc.) and zero.

9.11 The FBD program to perform the math function is shown in Figure 9.11e. This program subtracts the integer data in word MW20 from the integer data in word MW18 and stores the result in word MW22, if the input bit I124.0 is 1. The program then divides the result of the subtraction by 2 and stores the final result in word MW24.

9.12 The FBD program to perform the math function is shown in Figure 9.12e. This program subtracts a 32-bit floating-point number in word MD70 from a 32-bit floating-point number in word MD74 and stores the result in word MD78 if input I124.0 is set to 1. The program then divides the result by 4.5 and stores the final result in word MD82.

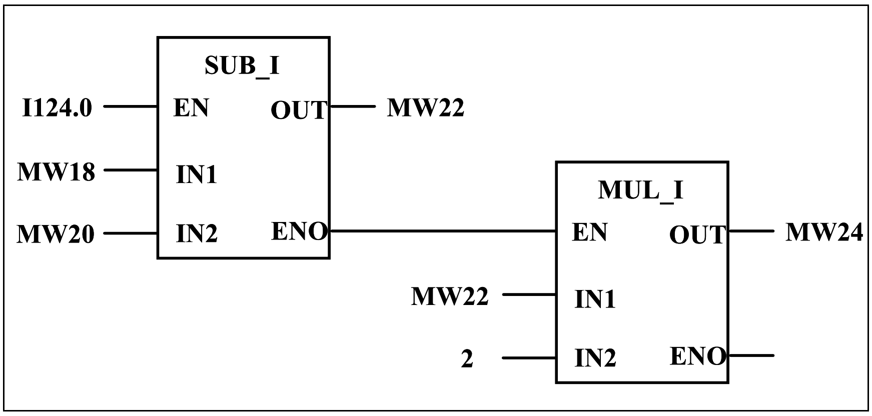


Figure 9-11e. Answer to Exercise 9.11

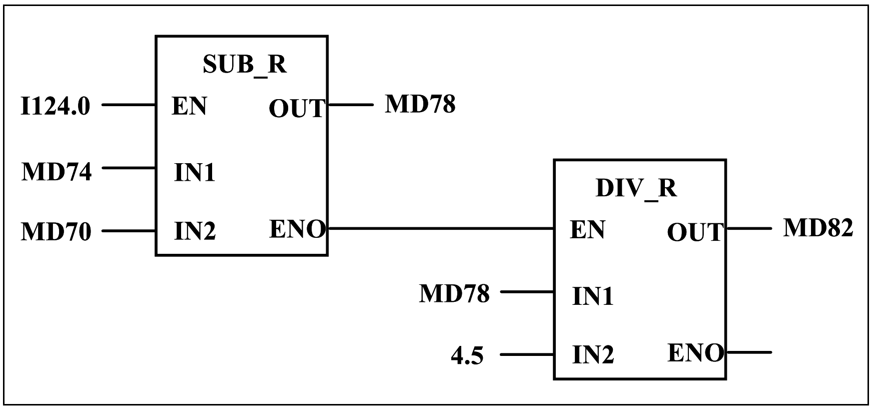


Figure 9-12e. Answer to Exercise 9.12

Chapter 10

10.1 The three basic components in a data communication system are: the *transmitter* to generate the information, the *medium* to carry the data, and the *receiver* to detect the data.

10.2 In unidirectional communications, a single channel is used and communication is only one way (i.e., from transmitter to receiver), so the receiver can never respond. Two-way communication allows the receiver to verify that the data was received. One kind of two-way communication is called half-duplex. In half-duplex, a single channel is used and communication is two-way, but communication can occur only in one direction at a time. Two-way communications where data can flow in both directions at the same time is called *full-duplex communications*. In this case, two physical paths are required so information can flow from both directions simultaneously.

10.3 In parallel transmission, all bits are transmitted at the same time and each bit of information requires a unique channel. For example, if we transmit an 8-bit ASCII character, 8 channels are required. The term “parallel”' refers to the position of the bits of the character and the fact that the characters are transferred as a group of bits in parallel. The use of a group of channels results in a high data transfer rate. In serial transmission, the bits of the encoded character are transmitted one after another in a single channel. The transmission takes the form of a bit stream that the receiver must assemble into characters (normally 8 bits). The main advantages of serial communications are lower cost and the elimination of byte synchronization problems.

10.4 The three common methods of signal multiplexing encountered in communications systems are frequency division, time division, and statistical.

10.5 The four common data transmission error-checking methods used in PLC communications systems are echo check, vertical redundancy check, longitudinal redundancy check, and cyclical redundancy check.

10.6 The LRC/VRC odd parity characters required for the message: M1 ON are given in Table 10-6e below:

Table 10-6e. Answer to Exercise 10.6

ASCII String	**M**	**1**		**O**	**N**	**LRC odd parity bit**
Bit 1	1	1	0	1	0	0
Bit 2	0	0	0	1	1	1
Bit 3	1	0	0	1	1	0
Bit 4	1	0	0	1	1	0
Bit 5	0	1	0	0	0	0
Bit 6	0	1	1	0	0	1
Bit 7	1	0	0	1	1	0
VRC odd parity bit	1	0	0	0	1	1

10.7 In *asynchronous*, successive data appear in the data channel at arbitrary times with no specific clock control governing the time delays between information. In *synchronous*, each successive datum in a data stream is controlled by a master data clock and appears at a specific interval of time.

10.8 Three common serial data protocols used in communications systems are BISYNC, DDCMP, and HDLC.

10.9 The bit streams required to send the message RUN using 8-bit even parity ASCII code and BISYNC protocol are as follows: SYNC = 10010110, SYNC = 10010110, SOH = 10000010, STX = 10000010, R = 11010010, U = 110110101, N = 01001110, ETX = 10000011.

10.10 The three most common LAN topologies are ring, bus, and star.

10.11 The seven layers of the ISO/OSI communications standard are: 1) physical, 2) data link, 3) network, 4) transport, 5) session, 6) presentation, and 7) application.

10.12 An advantage for using an Ethernet network in a PLC application is low cost.

10.13 The DTE is the end device that originates data to or receives data from data communications equipment. Some examples of DTE devices are PLCs, PCs, and printers.

10.14 Examples of open communications networks used in PLC applications are: Modbus Remote Terminal, ControlNet, Ethernet, and Profibus.

Chapter 11

11.1 The basic design elements required in the design of a PLC-based control system are: process and mechanical control diagrams, process and machine control descriptions, sizing and selection of a PLC system, control system drawings, I/O wiring diagrams, and control system programming.

11.2 Piping and instrument drawings show the process equipment to be controlled and the instrumentation used in the control of the process.

11.3 The ISA instrumentation symbol used to indicate a PLC I/O signal is a square box with a diamond inside.

11.4 Simplified process or mechanical control diagrams are used to show the status of the process in each step or state. The designer of the system also uses them to aid in programming. These drawings can also be used in operator instruction manuals to aid plant or operator personnel in understanding the operation of the control system.

11.5 The process control description conveys in simple language the purpose and the steps of the process. It is also the main vehicle of communication between the user or customer and the system designer.

11.6 The memory size required for a PLC application with the following I/O points: 1) Main Control Room: 150 points, 2) Remote Process Area 1: 254 points, 3) Remote Process Area 2: 125 points, 4) Remote Process Area 3: 156 points is 7K.

11.7 Ladder Diagram (LD) is the most common programming language used in PLC applications.

11.8 Some of the common peripheral equipment used in PLC applications is programming devices, mass storage units, and printers.

11.9 The purpose of a PLC system drawing is to give an overall view of the system component interconnections and the communication cabling layout.

11.10 If a third process tower is added, we would need to add four solenoid operated air valves with open and closed limit switches. Therefore, we would need to add four AC output points or one additional AC output module and a single AC input module for the 8 discrete inputs from the 8 valve position switches. A third set of pressure switches (high and low) would be required in the application but the two discrete inputs from these pressure switches could be connected to the spare discrete inputs on the existing AC input module shown in Figure 11-10.

11.11 The ladder diagram program to control the valves on tower 2 of the dehydration system application is shown in Figure 11-11e.

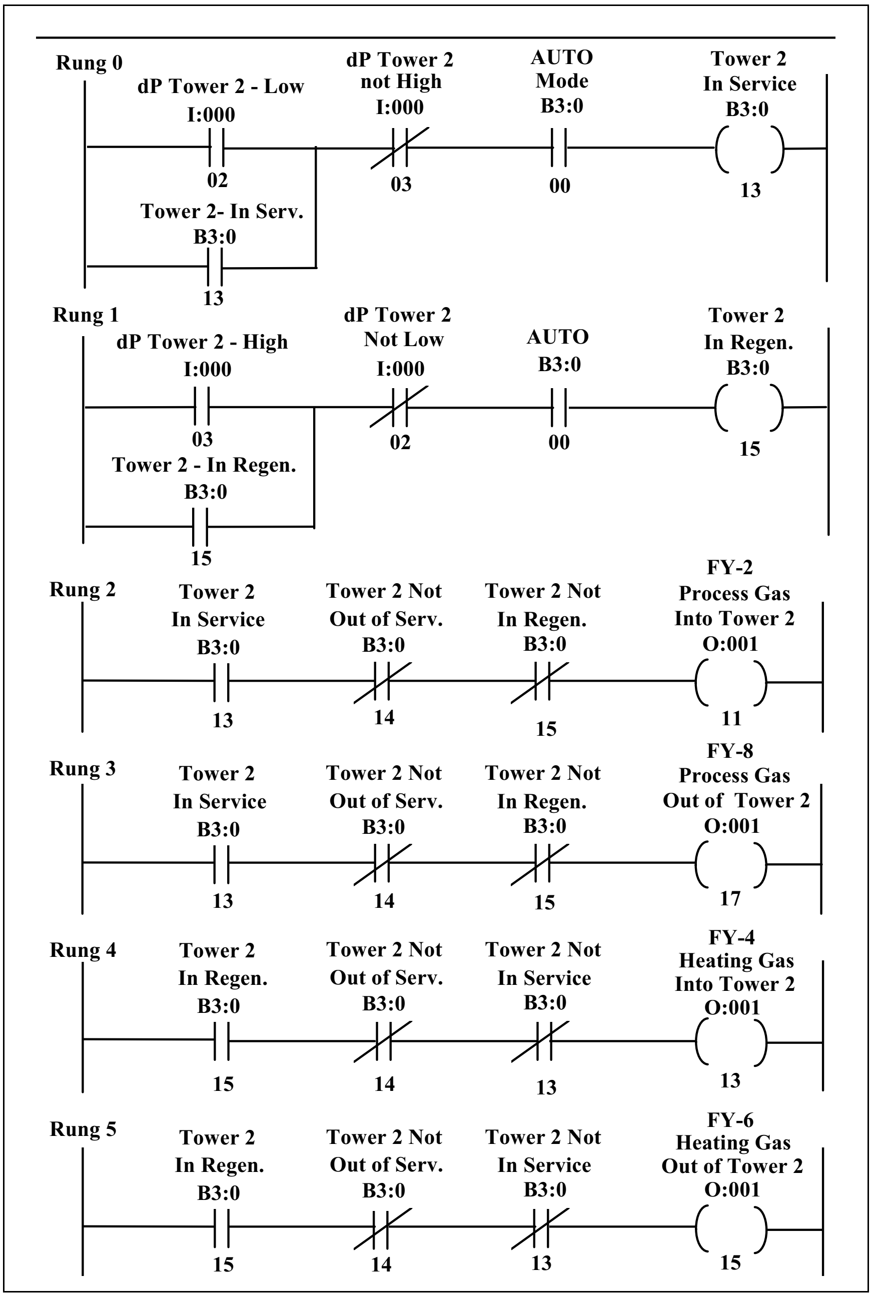


Figure 11-11e. Answer to Exercise 11.11

11.12 The revision of Rungs 2 and 3 of the LD program for alternating pump application are shown in Figure 11-12e.

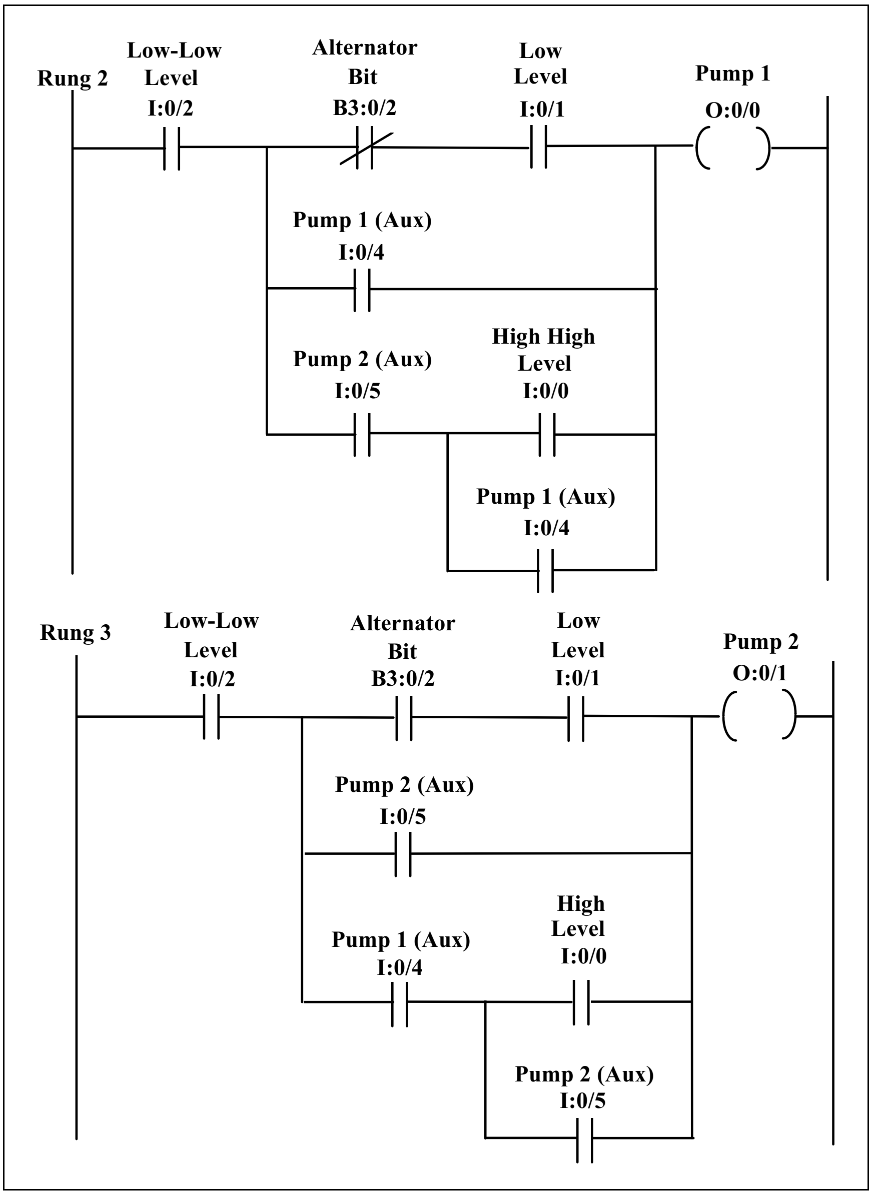


Figure 11-12e. Answer to Exercise 11.12

Chapter 12

12.1 Programmable controller components are placed in metal enclosures to protect them against atmospheric contaminants such as dust, moisture, oils, and other corrosive airborne substances. These metal enclosures also reduce the effects of electromagnetic radiation generated by electrical or welding equipment.

12.2 The temperature inside the control panel or enclosure must not exceed the maximum operating temperature listed in the manufacturer's installation manual (typically 120°F). If the temperature limit cannot be maintained by convection cooling, fans or an air conditioner must be installed to help dissipate the heat. The fans or air conditioner will generally be equipped with filters to prevent dust, dirt, and other airborne contaminants from entering the enclosure and affecting system components.

12.3 The following is a list of the design feature to reduce maintenance cost: 1) proper placement of system components, 2) an AC power outlet strip in control panel, 3) interior lighting in large panels, and 4) in harsh environmental, a gasketed glass window should be installed in panels to allow for viewing of the processor status lights and/or I/O status indicators.

12.4 The National Electric Code states that a ground must be permanent (i.e., no solder connections), continuous, and able to safely conduct the current in the system with minimal resistance. The ground wire should be separated from the AC hot and AC neutral wires at the point of entry to the control panel. To minimize the ground wire length within the panel, the ground reference point should be located as close as possible to the point of entry of the panel supply line. All I/O racks, power supplies, processors, and other electrical devices in the system should be connected to a single ground bus in the control panel. Paint or other nonconductive materials should be scraped away from the area where an I/O rack makes contact with the panel. In addition to the ground connection made through the rack-mounting bolts, a metal braid of the size recommended by the PLC manufacturer should be used to connect each chassis and the panel at a single mounting bolt.

12.5 The phases of a typical control system operational test procedure are: 1) visual inspection, 2) continuity check, 3) input wiring check, 4) output wiring check, and 5) operational test.

12.6 The main features of a preventive maintenance program for PLC systems are: 1) check all electrical connections on a regular schedule, 2) inspect all air filters in system at scheduled intervals, 3) do not move equipment that produces high levels of electrical noise near operating PLC systems, and 4) do not store unnecessary items on or near PLC equipment.

12.7 The recommended maintenance practices are: 1) stocking of critical spare parts, 2) documentation of problems encountered, 3) monitoring of internal PLC faults, 4) monitoring of PLC output faults, 5) monitoring of power supplies, 6) strict control of program access, and 7) configuration control of documentation.

12.8 The common troubleshooting methods used in finding and correcting problems in PLC control systems are: (1) experience, (2) process of elimination, (3) divide and conquer, (4) remove and conquer, (5) substitution, (6) setting a trap, and (7) freezing the control sequence.

12.9 The typical PLC processor malfunctions are: errors in programming, memory failure, mathematical calculation overrun, program time-out, and processor scan overrun.

12.10 The most common cause of a power failure in a PLC system is a blown fuse.

12.11 The typical types of documentation used on complex control systems are: 1) software program listings, 2) backup copies of control programs and configuration software, 3) operations and maintenance (O&M) manuals, 4) system architecture drawings, 5) input/output wiring diagrams, 6) instrument loop diagrams, 7) process and instrument drawings (P&IDs), 8) equipment layout drawings, and 9) mechanical flow diagrams (MFDs).

12.12 The power distribution system drawing for the two tower dehydration PLC control system is shown in Figure 12-12e.

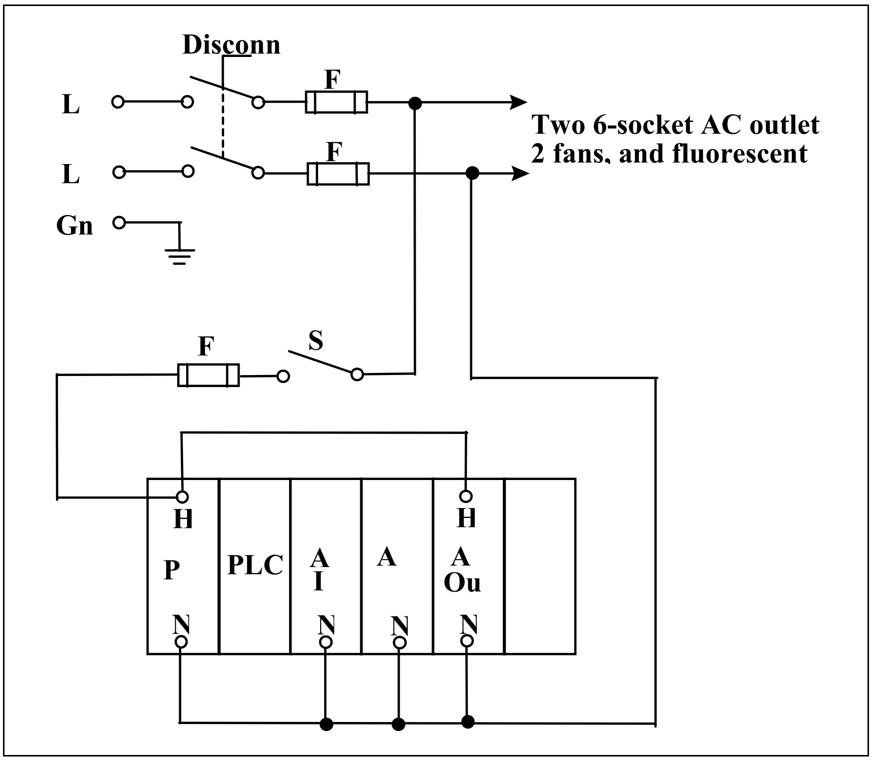


Figure 12-12e. Answer to Exercise 12.12

INDEX