Programmable logic controller

Contents

- 1 History
- 2 Development
  - 2.1 Programming
- 3 Functionality
- 4 PLC topics
  - 4.1 Features
  - 4.2 Scan time
  - 4.3 System scale
  - 4.4 User interface
  - 4.5 Communications
  - 4.6 Programming
- 5 PLC compared with other control systems
- 6 Digital and analog signals
  - 6.1 Example
- 7 Flowchart
- 8 References
- 9 Further reading

History

The PLC was invented in response to the needs of the American automotive manufacturing industry. Programmable logic controllers were initially adopted by the automotive industry where software revision replaced the re-wiring of hard-wired control panels when production models changed.

Before the PLC, control, sequencing, and safety interlock logic for manufacturing automobiles was accomplished using hundreds or thousands of relays, cam timers, and drum sequencers and dedicated closed-loop controllers. The process for updating such facilities for the yearly model change-over was very time consuming and expensive, as electricians needed to individually rewire each and every relay.

Digital computers, being general-purpose programmable devices, were soon applied to control of industrial processes. Early computers required specialist programmers, and stringent operating environmental control for temperature, cleanliness, and power quality. Using a general-purpose computer for process control required protecting the computer from the plant floor conditions. An industrial control computer would have several attributes: it would tolerate the shop-floor environment, it would support discrete (bit-form) input and output in an easily extensible manner, it would not require years of training to use, and it would permit its operation to be monitored. The response time of any computer system must be fast enough to be useful for control; the required speed varying according to the nature of the process.[1]

In 1968 GM Hydramatic (the automatic transmission division of General Motors) issued a request for proposal for an electronic replacement for hard-wired relay systems. The winning proposal came from Bedford Associates of Bedford, Massachusetts. The first PLC, designated the 084 because it was Bedford Associates' eighty-fourth project, was the result.[2] Bedford Associates started a new company dedicated to developing, manufacturing, selling, and servicing this new product: Modicon, which stood for MOdular DIgital CONtroller. One of the people who worked on that project was Dick Morley, who is considered to be the "father" of the PLC.[3] The Modicon brand was sold in 1977 to Gould Electronics, and later acquired by German Company AEG and then by French Schneider Electric, the current owner.

One of the very first 084 models built is now on display at Modicon's headquarters in North Andover, Massachusetts. It was presented to Modicon by GM, when the unit was retired after nearly twenty years of uninterrupted service. Modicon used the 84 moniker at the end of its product range until the 984 made its appearance.

The automotive industry is still one of the largest users of PLCs.

Development

Early PLCs were designed to replace relay logic systems. These PLCs were programmed in "ladder logic", which strongly resembles a schematic diagram of relay logic. This program notation was chosen to reduce training demands for the existing technicians. Other early PLCs used a form of instruction list programming, based on a stack-based logic solver.

Modern PLCs can be programmed in a variety of ways, from ladder logic to more traditional programming languages such as BASIC and C. Another method is State Logic, a very high-level programming language designed to program PLCs based on state transition diagrams.

Many early PLCs did not have accompanying programming terminals that were capable of graphical representation of the logic, and so the logic was instead represented as a series of logic expressions in some version of Boolean format, similar to Boolean algebra. As programming terminals evolved, it became more common for ladder logic to be used, for the aforementioned reasons and because it was a familiar format used for electromechanical control panels. Newer formats such as State Logic and Function Block (which is similar to the way logic is depicted when using digital integrated logic circuits) exist, but they are still not as popular as ladder logic. A primary reason for this is that PLCs solve the logic in a predictable and repeating sequence, and ladder logic allows the programmer (the person writing the logic) to see any issues with the timing of the logic sequence more easily than would be possible in other formats.

Programming

Early PLCs, up to the mid-1980s, were programmed using proprietary programming panels or special-purpose programming terminals, which often had dedicated function keys representing the various logical elements of PLC programs.[2] Programs were stored on cassette tape cartridges. Facilities for printing and documentation were very minimal due to lack of memory capacity. The very oldest PLCs used non-volatile magnetic core memory.

More recently, PLCs are programmed using application software on personal computers. The computer is connected to the PLC through Ethernet, RS-232, RS-485 or RS-422 cabling. The programming software allows entry and editing of the ladder-style logic. Generally the software provides functions for debugging and troubleshooting the PLC software, for example, by highlighting portions of the logic to show current status during operation or via simulation. The software will upload and download the PLC program, for backup and restoration purposes. In some models of programmable controller, the program is transferred from a personal computer to the PLC through a programming board which writes the program into a removable chip such as an EEPROM or EPROM.

Functionality

The functionality of the PLC has evolved over the years to include sequential relay control, motion control, process control, distributed control systems and networking. The data handling, storage, processing power and communication capabilities of some modern PLCs are approximately equivalent to desktop computers. PLC-like programming combined with remote I/O hardware, allow a general-purpose desktop computer to overlap some PLCs in certain applications. Regarding the practicality of these desktop computer based logic controllers, it is important to note that they have not been generally accepted in heavy industry because the desktop computers run on less stable operating systems than do PLCs, and because the desktop computer hardware is typically not designed to the same levels of tolerance to temperature, humidity, vibration, and longevity as the processors used in PLCs. In addition to the hardware limitations of desktop based logic, operating systems such as Windows do not lend themselves to deterministic logic execution, with the result that the logic may not always respond to changes in logic state or input status with the extreme consistency in timing as is expected from PLCs. Still, such desktop logic applications find use in less critical situations, such as laboratory automation and use in small facilities where the application is less demanding and critical, because they are generally much less expensive than PLCs.

In more recent years, small products called PLRs (programmable logic relays), and also by similar names, have become more common and accepted. These are very much like PLCs, and are used in light industry where only a few points of I/O (i.e. a few signals coming in from the real world and a few going out) are involved, and low cost is desired. These small devices are typically made in a common physical size and shape by several manufacturers, and branded by the makers of larger PLCs to fill out their low end product range. Popular names include PICO Controller, NANO PLC, and other names implying very small controllers. Most of these have between 8 and 12 digital inputs, 4 and 8 digital outputs, and up to 2 analog inputs. Size is usually about 4" wide, 3" high, and 3" deep. Most such devices include a tiny postage stamp sized LCD screen for viewing simplified ladder logic (only a very small portion of the program being visible at a given time) and status of I/O points, and typically these screens are accompanied by a 4-way rocker push-button plus four more separate push-buttons, similar to the key buttons on a VCR remote control, and used to navigate and edit the logic. Most have a small plug for connecting via RS-232 or RS-485 to a personal computer so that programmers can use simple Windows applications for programming instead of being forced to use the tiny LCD and push-button set for this purpose. Unlike regular PLCs that are usually modular and greatly expandable, the PLRs are usually not modular or expandable, but their price can be two orders of magnitude less than a PLC and they still offer robust design and deterministic execution of the logic.

PLC topics

Features

Control panel with PLC (grey elements in the center). The unit consists of separate elements, from left to right; power supply, controller, relay units for in- and output

The main difference from other computers is that PLCs are armored for severe conditions (such as dust, moisture, heat, cold) and have the facility for extensive input/output (I/O) arrangements. These connect the PLC to sensors and actuators. PLCs read limit switches, analog process variables (such as temperature and pressure), and the positions of complex positioning systems. Some use machine vision. On the actuator side, PLCs operate electric motors, pneumatic or hydraulic cylinders, magnetic relays, solenoids, or analog outputs. The input/output arrangements may be built into a simple PLC, or the PLC may have external I/O modules attached to a computer network that plugs into the PLC.

Scan time

A PLC program is generally executed repeatedly as long as the controlled system is running. The status of physical input points is copied to an area of memory accessible to the processor, sometimes called the "I/O Image Table". The program is then run from its first instruction rung down to the last rung. It takes some time for the processor of the PLC to evaluate all the rungs and update the I/O image table with the status of outputs. This scan time may be a few milliseconds for a small program or on a fast processor, but older PLCs running very large programs could take much longer (say, up to 100 ms) to execute the program. If the scan time was too long, the response of the PLC to process conditions would be too slow to be useful.

As PLCs became more advanced, methods were developed to change the sequence of ladder execution, and subroutines were implemented. This simplified programming and could also be used to save scan time for high-speed processes; for example, parts of the program used only for setting up the machine could be segregated from those parts required to operate at higher speed.

Special-purpose I/O modules, such as timer modules or counter modules, could be used where the scan time of the processor was too long to reliably pick up, for example, counting pulses from a shaft encoder. The relatively slow PLC could still interpret the counted values to control a machine, but the accumulation of pulses was done by a dedicated module that was unaffected by the speed of the program execution.

System scale

A small PLC will have a fixed number of connections built in for inputs and outputs. Typically, expansions are available if the base model has insufficient I/O.

Modular PLCs have a chassis (also called a rack) into which are placed modules with different functions. The processor and selection of I/O modules are customized for the particular application. Several racks can be administered by a single processor, and may have thousands of inputs and outputs. A special high speed serial I/O link is used so that racks can be distributed away from the processor, reducing the wiring costs for large plants.

User interface

PLCs may need to interact with people for the purpose of configuration, alarm reporting or everyday control. A human-machine interface (HMI) is employed for this purpose. HMIs are also referred to as man-machine interfaces (MMIs) and graphical user interface (GUIs). A simple system may use buttons and lights to interact with the user. Text displays are available as well as graphical touch screens. More complex systems use programming and monitoring software installed on a computer, with the PLC connected via a communication interface.

Communications

PLCs have built in communications ports, usually 9-pin RS-232, but optionally EIA-485 or Ethernet. Modbus, BACnet or DF1 is usually included as one of the communications protocols. Other options include various fieldbuses such as DeviceNet or Profibus. Other communications protocols that may be used are listed in the List of automation protocols.

Most modern PLCs can communicate over a network to some other system, such as a computer running a SCADA (Supervisory Control And Data Acquisition) system or web browser.

PLCs used in larger I/O systems may have peer-to-peer (P2P) communication between processors. This allows separate parts of a complex process to have individual control while allowing the subsystems to co-ordinate over the communication link. These communication links are also often used for HMI devices such as keypads or PC-type workstations.

Programming

Before the advent of solid-state logic circuits, logical control systems were designed and built exclusively around electromechanical relays. Relays are far from obsolete in modern design, but have been replaced in many of their former roles as logic-level control devices, relegated most often to those applications demanding high current and/or high voltage switching.

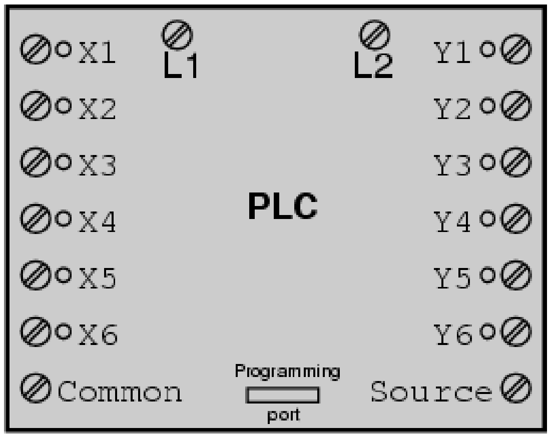
Systems and processes requiring "on/off" control abound in modern commerce and industry, but such control systems are rarely built from either electromechanical relays or discrete logic gates. Instead, digital computers fill the need, which may be *programmed* to do a variety of logical functions.

In the late 1960's an American company named Bedford Associates released a computing device they called the *MODICON*. As an acronym, it meant **Mod**ular **Di**gital **Con**troller, and later became the name of a company division devoted to the design, manufacture, and sale of these special-purpose control computers. Other engineering firms developed their own versions of this device, and it eventually came to be known in non-proprietary terms as a *PLC*, or **P**rogrammable **L**ogic **C**ontroller. The purpose of a PLC was to directly replace electromechanical relays as logic elements, substituting instead a solid-state digital computer with a stored program, able to emulate the interconnection of many relays to perform certain logical tasks.

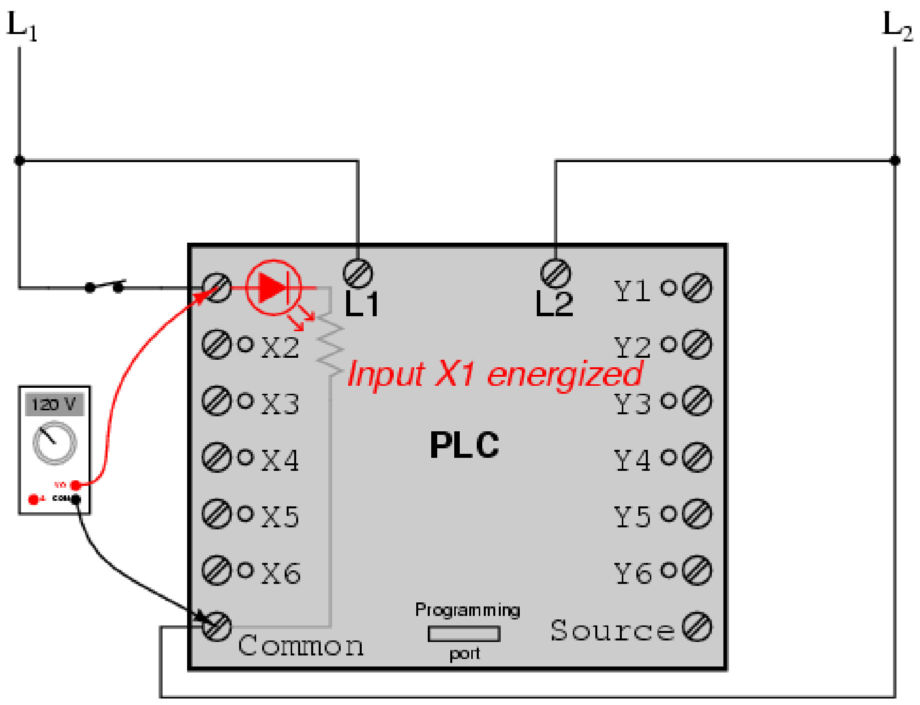
A PLC has many "input" terminals, through which it interprets "high" and "low" logical states from sensors and switches. It also has many output terminals, through which it outputs "high" and "low" signals to power lights, solenoids, contactors, small motors, and other devices lending themselves to on/off control. In an effort to make PLCs easy to program, their programming language was designed to resemble ladder logic diagrams. Thus, an industrial electrician or electrical engineer accustomed to reading ladder logic schematics would feel comfortable programming a PLC to perform the same control functions.

PLCs are industrial computers, and as such their input and output signals are typically 120 volts AC, just like the electromechanical control relays they were designed to replace. Although some PLCs have the ability to input and output low-level DC voltage signals of the magnitude used in logic gate circuits, this is the exception and not the rule.

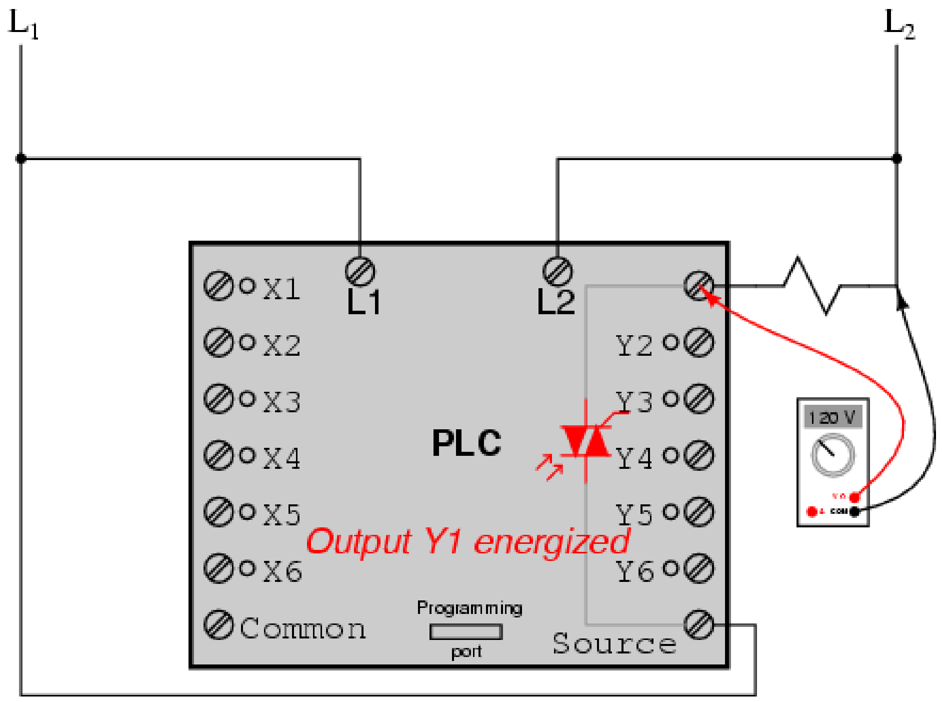
Signal connection and programming standards vary somewhat between different models of PLC, but they are similar enough to allow a "generic" introduction to PLC programming here. The following illustration shows a simple PLC, as it might appear from a front view. Two screw terminals provide connection to 120 volts AC for powering the PLC's internal circuitry, labeled L1 and L2. Six screw terminals on the left-hand side provide connection to input devices, each terminal representing a different input "channel" with its own "X" label. The lower-left screw terminal is a "Common" connection, which is generally connected to L2 (neutral) of the 120 VAC power source.



Inside the PLC housing, connected between each input terminal and the Common terminal, is an opto-isolator device (Light-Emitting Diode) that provides an electrically isolated "high" logic signal to the computer's circuitry (a photo-transistor interprets the LED's light) when there is 120 VAC power applied between the respective input terminal and the Common terminal. An indicating LED on the front panel of the PLC gives visual indication of an "energized" input:



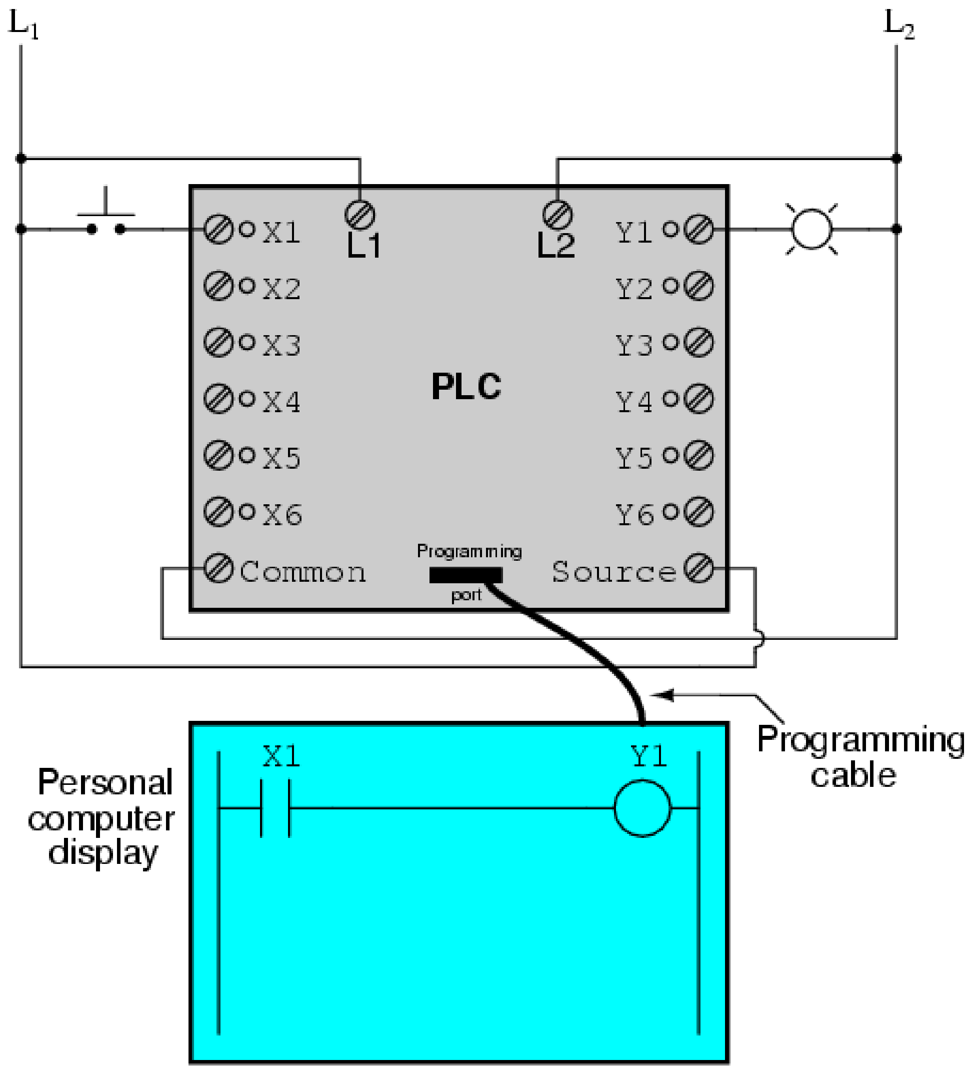
Output signals are generated by the PLC's computer circuitry activating a switching device (transistor, TRIAC, or even an electromechanical relay), connecting the "Source" terminal to any of the "Y-" labeled output terminals. The "Source" terminal, correspondingly, is usually connected to the L1 side of the 120 VAC power source. As with each input, an indicating LED on the front panel of the PLC gives visual indication of an "energized" output:



In this way, the PLC is able to interface with real-world devices such as switches and solenoids.

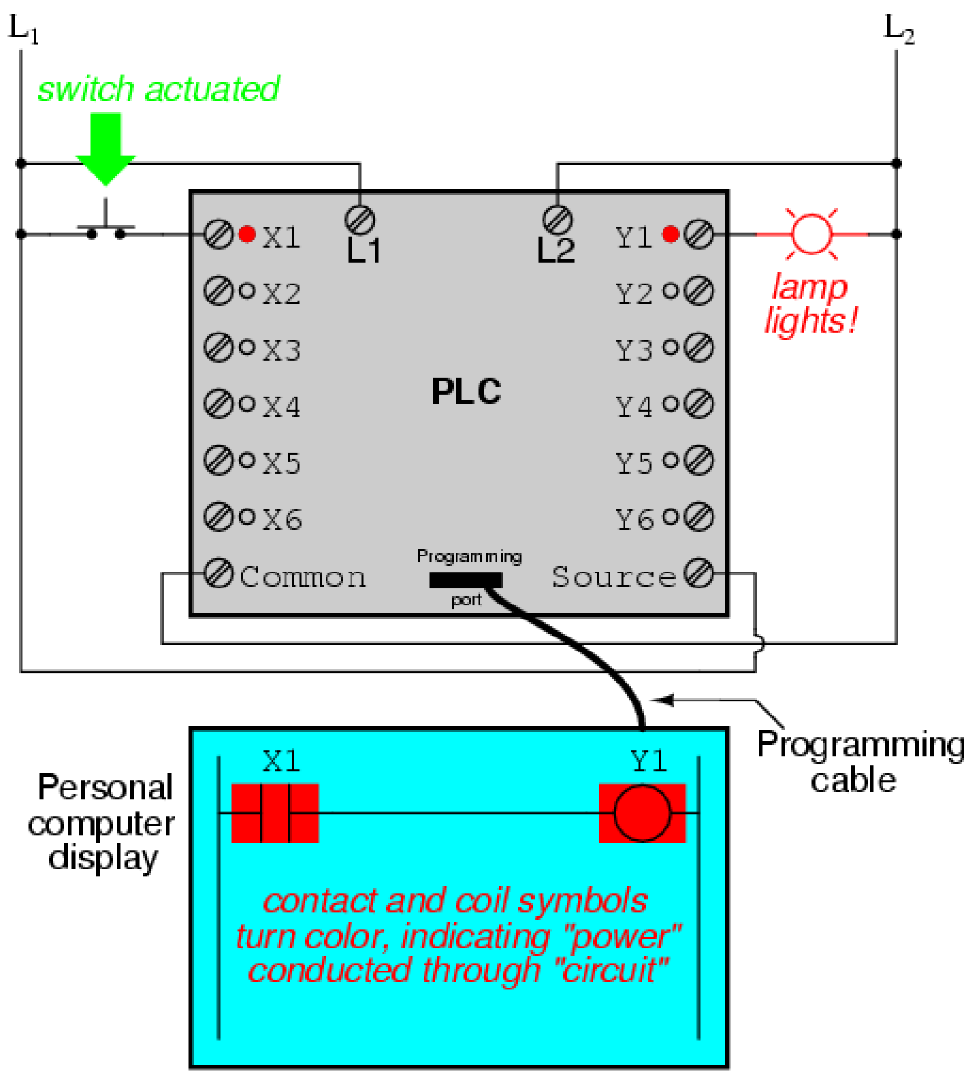
The actual *logic* of the control system is established inside the PLC by means of a computer program. This program dictates which output gets energized under which input conditions. Although the program itself appears to be a ladder logic diagram, with switch and relay symbols, there are no actual switch contacts or relay coils operating inside the PLC to create the logical relationships between input and output. These are *imaginary* contacts and coils, if you will. The program is entered and viewed via a personal computer connected to the PLC's programming port.

Consider the following circuit and PLC program:



When the pushbutton switch is unactuated (unpressed), no power is sent to the X1 input of the PLC. Following the program, which shows a normally-open X1 contact in series with a Y1 coil, no "power" will be sent to the Y1 coil. Thus, the PLC's Y1 output remains de-energized, and the indicator lamp connected to it remains dark.

If the pushbutton switch is pressed, however, power will be sent to the PLC's X1 input. Any and all X1 contacts appearing in the program will assume the actuated (non-normal) state, as though they were relay contacts actuated by the energizing of a relay coil named "X1". In this case, energizing the X1 input will cause the normally-open X1 contact will "close," sending "power" to the Y1 coil. When the Y1 coil of the program "energizes," the real Y1 output will become energized, lighting up the lamp connected to it:

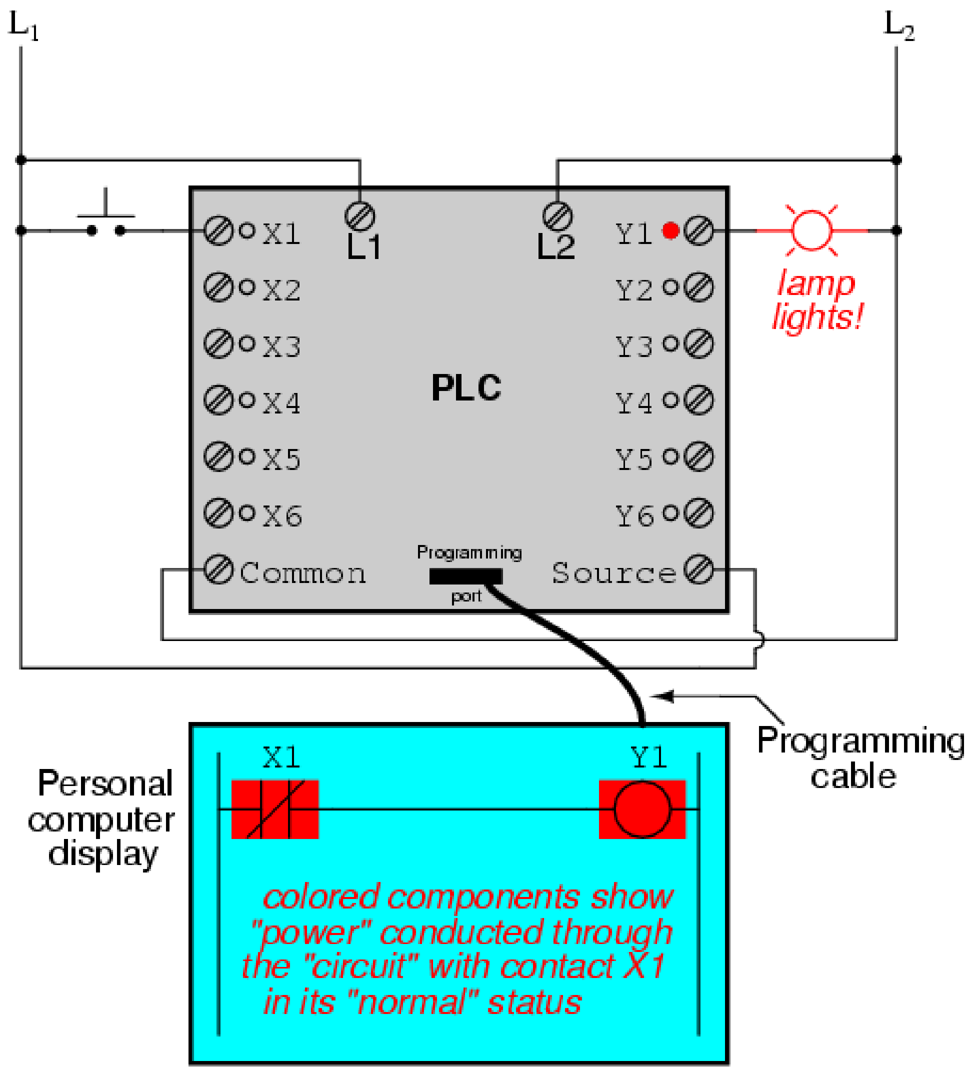


It must be understood that the X1 contact, Y1 coil, connecting wires, and "power" appearing in the personal computer's display are all *virtual*. They do not exist as real electrical components. They exist as commands in a computer program -- a piece of software only -- that just happens to resemble a real relay schematic diagram.

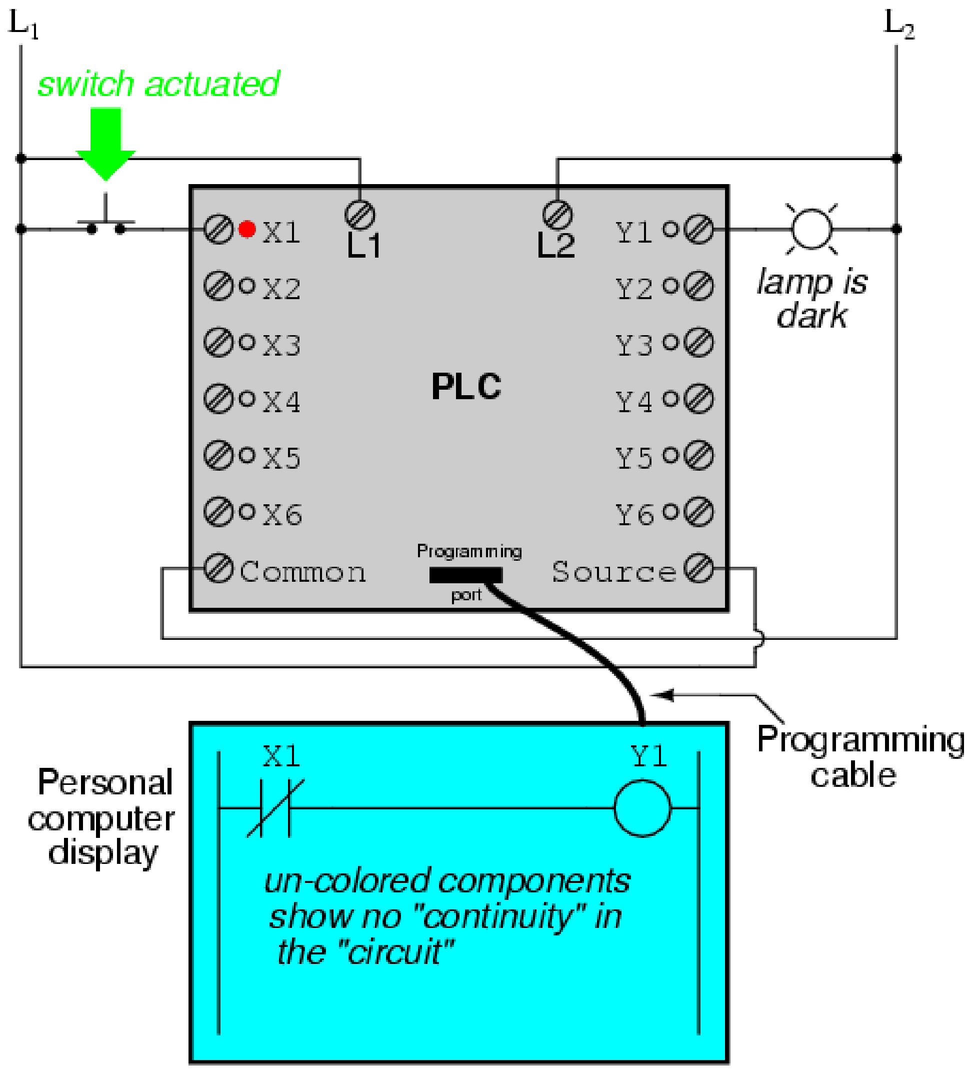
Equally important to understand is that the personal computer used to display and edit the PLC's program is not necessary for the PLC's continued operation. Once a program has been loaded to the PLC from the personal computer, the personal computer may be unplugged from the PLC, and the PLC will continue to follow the programmed commands. I include the personal computer display in these illustrations for your sake only, in aiding to understand the relationship between real-life conditions (switch closure and lamp status) and the program's status ("power" through virtual contacts and virtual coils).

The true power and versatility of a PLC is revealed when we want to alter the behavior of a control system. Since the PLC is a programmable device, we can alter its behavior by changing the commands we give it, without having to reconfigure the electrical components connected to it. For example, suppose we wanted to make this switch-and-lamp circuit function in an inverted fashion: push the button to make the lamp turn *off*, and release it to make it turn *on*. The "hardware" solution would require that a normally-closed pushbutton switch be substituted for the normally-open switch currently in place. The "software" solution is much easier: just alter the program so that contact X1 is normally-closed rather than normally-open.

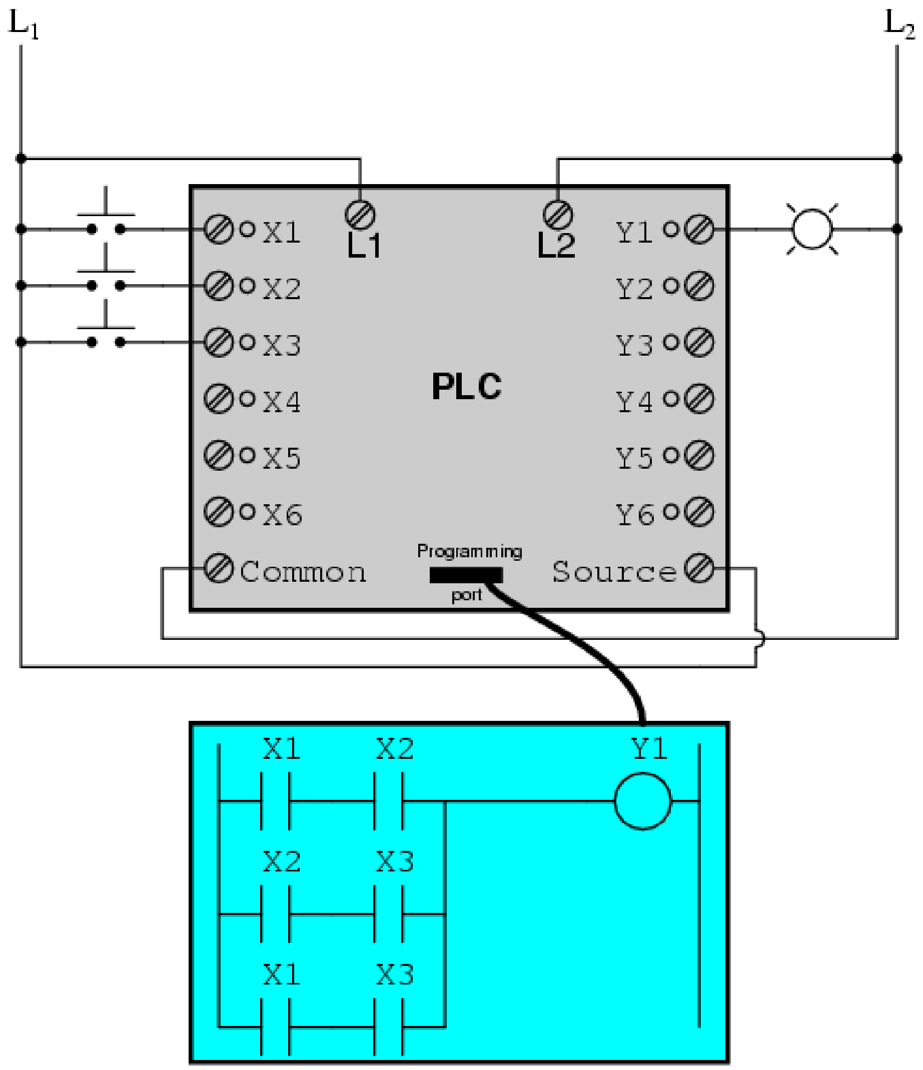
In the following illustration, we have the altered system shown in the state where the pushbutton is unactuated (*not* being pressed):



In this next illustration, the switch is shown actuated (pressed):

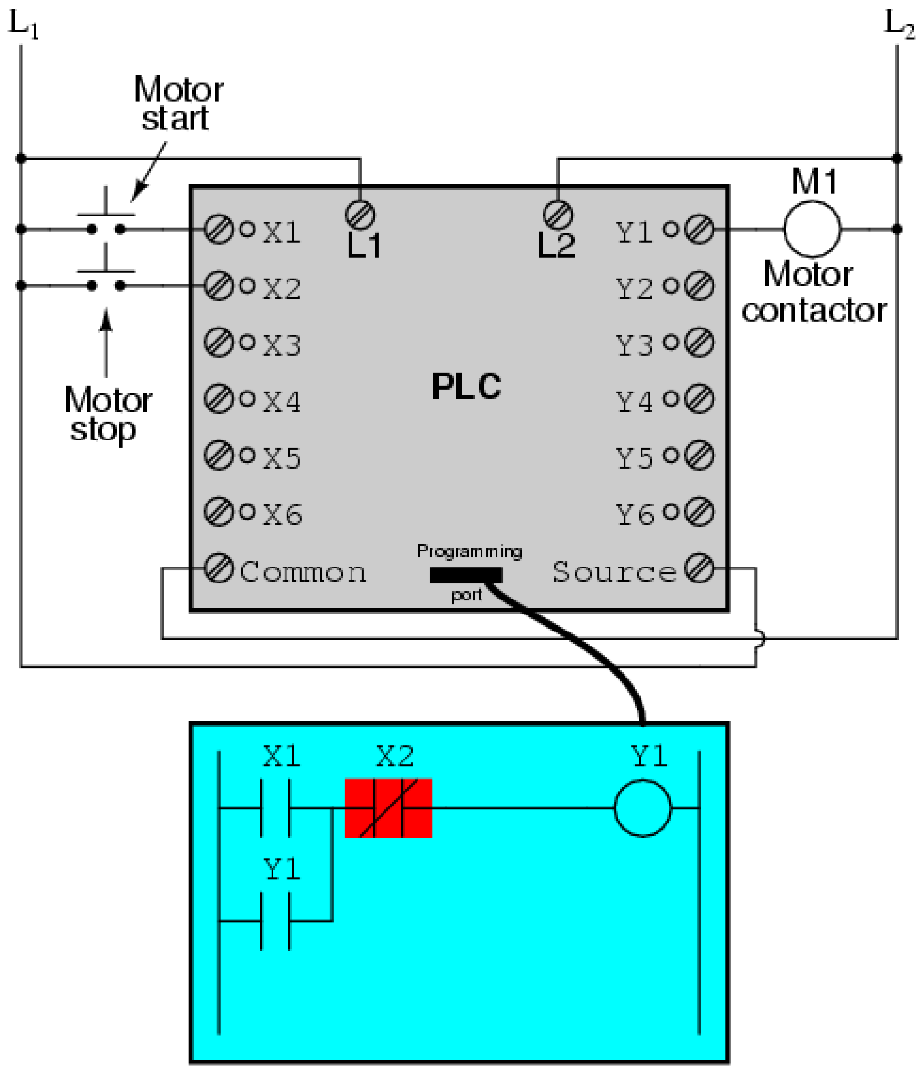


One of the advantages of implementing logical control in software rather than in hardware is that input signals can be re-used as many times in the program as is necessary. For example, take the following circuit and program, designed to energize the lamp if at least two of the three pushbutton switches are simultaneously actuated:



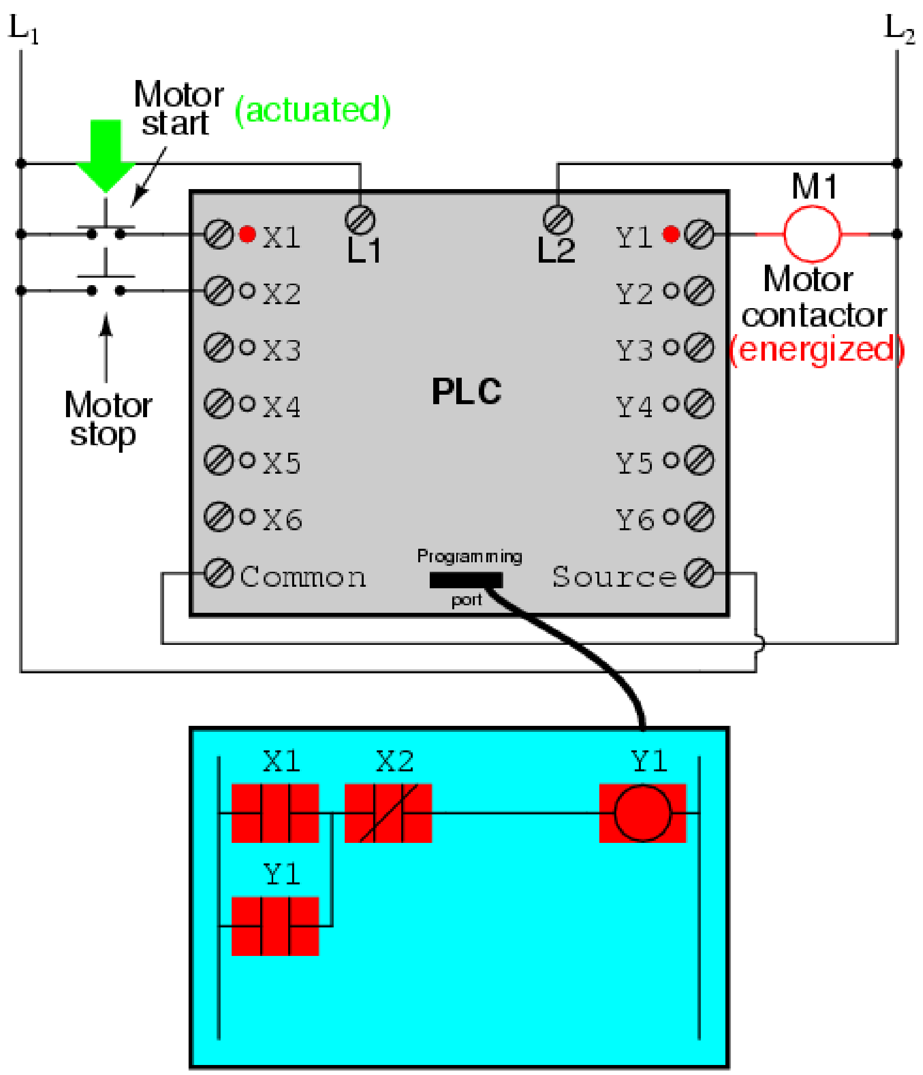
To build an equivalent circuit using electromechanical relays, three relays with two normally-open contacts each would have to be used, to provide two contacts per input switch. Using a PLC, however, we can program as many contacts as we wish for each "X" input without adding additional hardware, since each input and each output is nothing more than a single bit in the PLC's digital memory (either 0 or 1), and can be recalled as many times as necessary.

Furthermore, since each output in the PLC is nothing more than a bit in its memory as well, we can assign contacts in a PLC program "actuated" by an output (Y) status. Take for instance this next system, a motor start-stop control circuit:

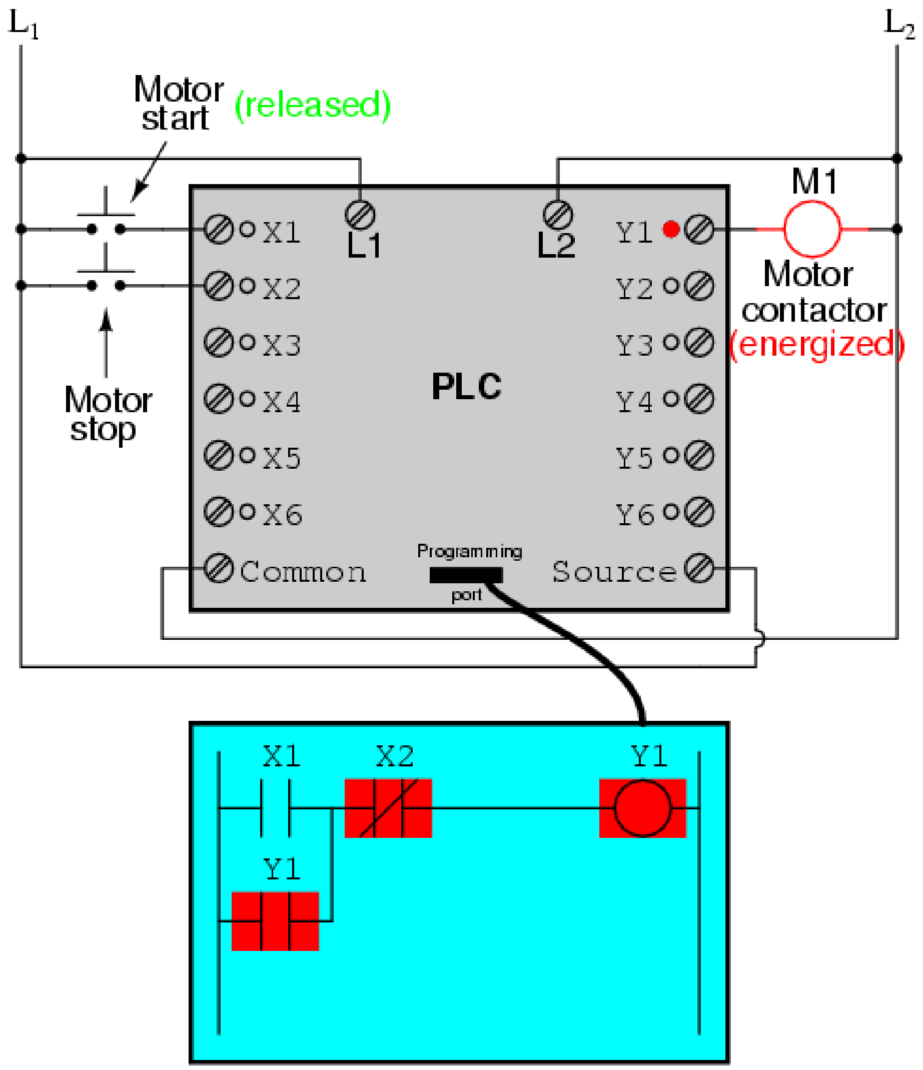


The pushbutton switch connected to input X1 serves as the "Start" switch, while the switch connected to input X2 serves as the "Stop." Another contact in the program, named Y1, uses the output coil status as a seal-in contact, directly, so that the motor contactor will continue to be energized after the "Start" pushbutton switch is released. You can see the normally-closed contact X2 appear in a colored block, showing that it is in a closed ("electrically conducting") state.

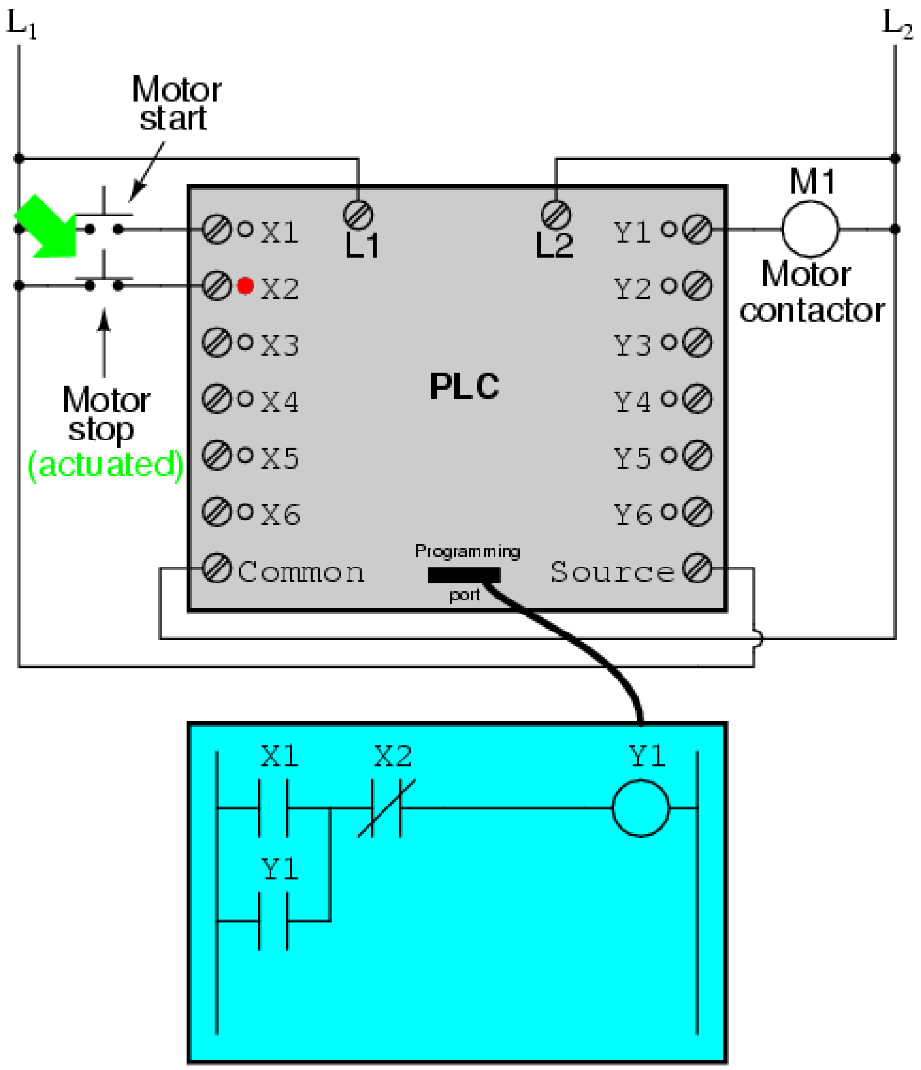
If we were to press the "Start" button, input X1 would energize, thus "closing" the X1 contact in the program, sending "power" to the Y1 "coil," energizing the Y1 output and applying 120 volt AC power to the real motor contactor coil. The parallel Y1 contact will also "close," thus latching the "circuit" in an energized state:



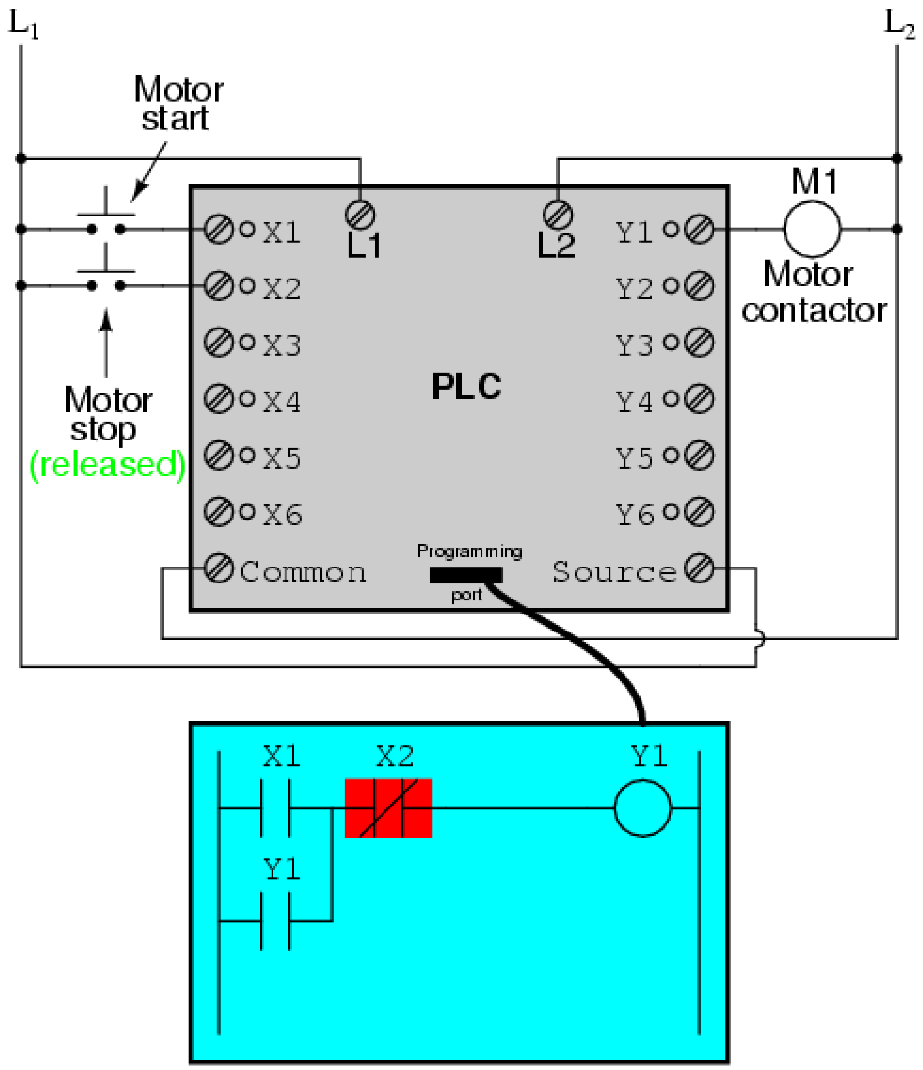
Now, if we release the "Start" pushbutton, the normally-open X1 "contact" will return to its "open" state, but the motor will continue to run because the Y1 seal-in "contact" continues to provide "continuity" to "power" coil Y1, thus keeping the Y1 output energized:



To stop the motor, we must momentarily press the "Stop" pushbutton, which will energize the X2 input and "open" the normally-closed "contact," breaking continuity to the Y1 "coil:"

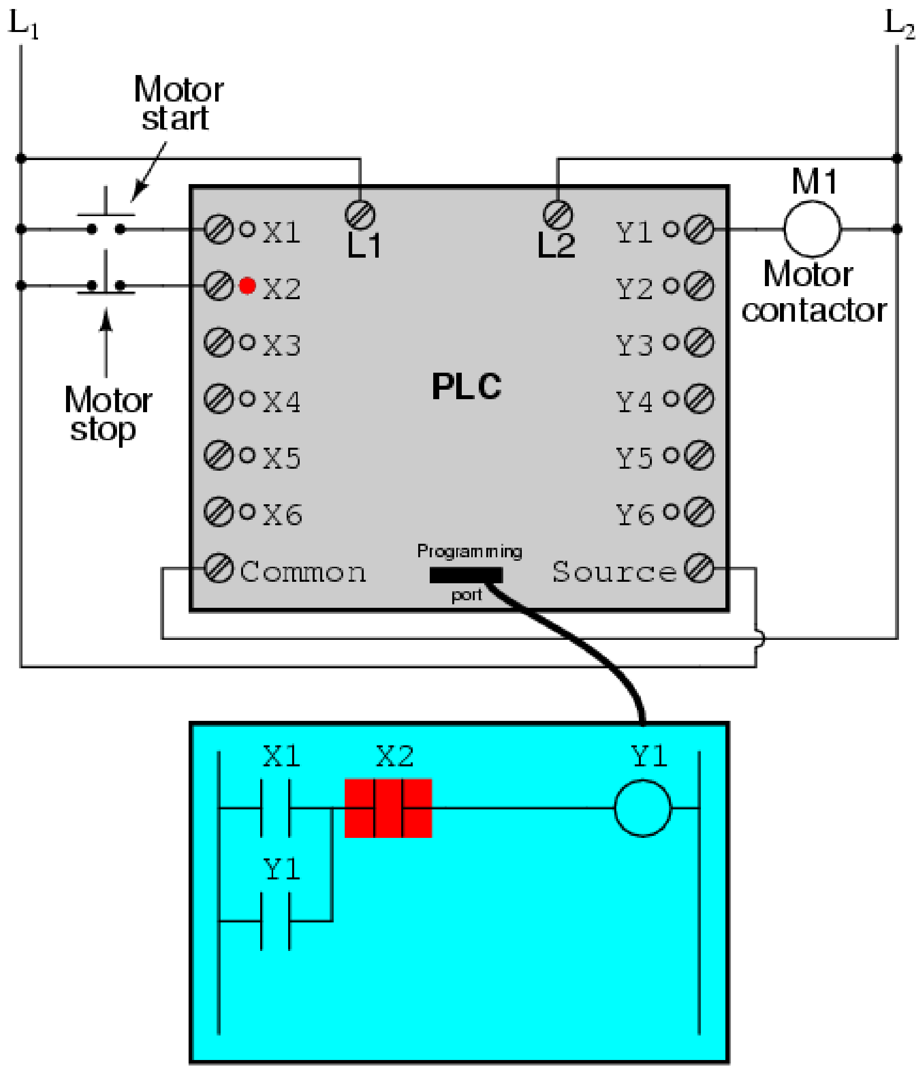


When the "Stop" pushbutton is released, input X2 will de-energize, returning "contact" X2 to its normal, "closed" state. The motor, however, will not start again until the "Start" pushbutton is actuated, because the "seal-in" of Y1 has been lost:



An important point to make here is that *fail-safe* design is just as important in PLC-controlled systems as it is in electromechanical relay-controlled systems. One should always consider the effects of failed (open) wiring on the device or devices being controlled. In this motor control circuit example, we have a problem: if the input wiring for X2 (the "Stop" switch) were to fail open, there would be no way to stop the motor!

The solution to this problem is a reversal of logic between the X2 "contact" inside the PLC program and the actual "Stop" pushbutton switch:



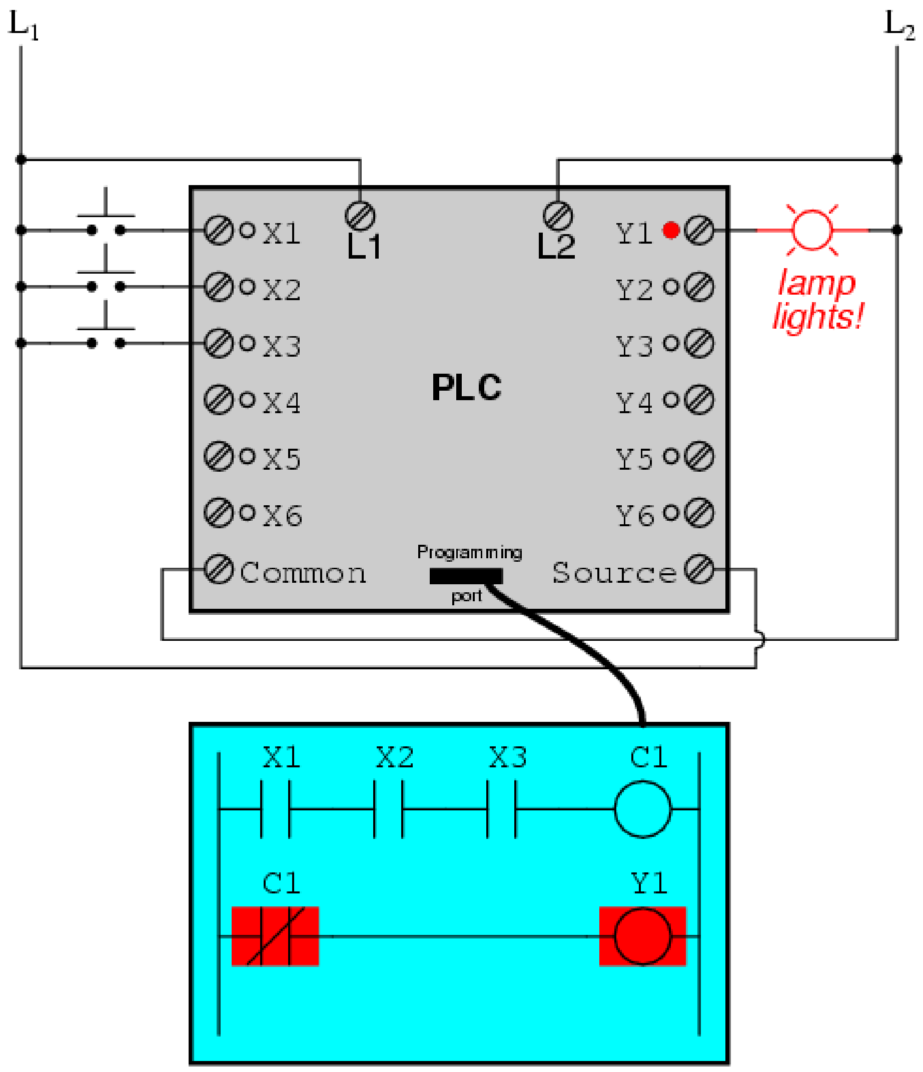
When the normally-closed "Stop" pushbutton switch is unactuated (not pressed), the PLC's X2 input will be energized, thus "closing" the X2 "contact" inside the program. This allows the motor to be started when input X1 is energized, and allows it to continue to run when the "Start" pushbutton is no longer pressed. When the "Stop" pushbutton is actuated, input X2 will de-energize, thus "opening" the X2 "contact" inside the PLC program and shutting off the motor. So, we see there is no operational difference between this new design and the previous design.

However, if the input wiring on input X2 were to fail open, X2 input would de-energize in the same manner as when the "Stop" pushbutton is pressed. The result, then, for a wiring failure on the X2 input is that the motor will immediately shut off. This is a safer design than the one previously shown, where a "Stop" switch wiring failure would have resulted in an *inability* to turn off the motor.

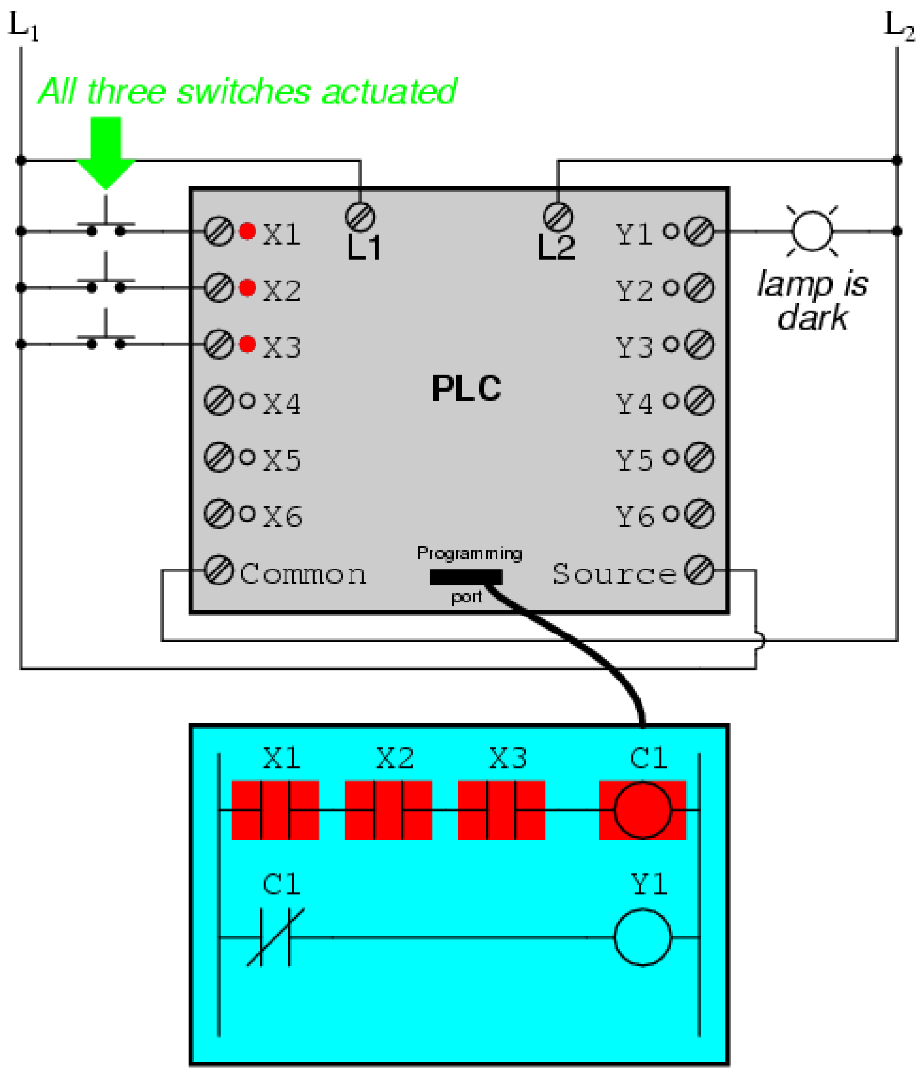
In addition to input (X) and output (Y) program elements, PLCs provide "internal" coils and contacts with no intrinsic connection to the outside world. These are used much the same as "control relays" (CR1, CR2, etc.) are used in standard relay circuits: to provide logic signal inversion when necessary.

To demonstrate how one of these "internal" relays might be used, consider the following example circuit and program, designed to emulate the function of a three-input NAND gate.

Since PLC program elements are typically designed by single letters, I will call the internal control relay "C1" rather than "CR1" as would be customary in a relay control circuit:

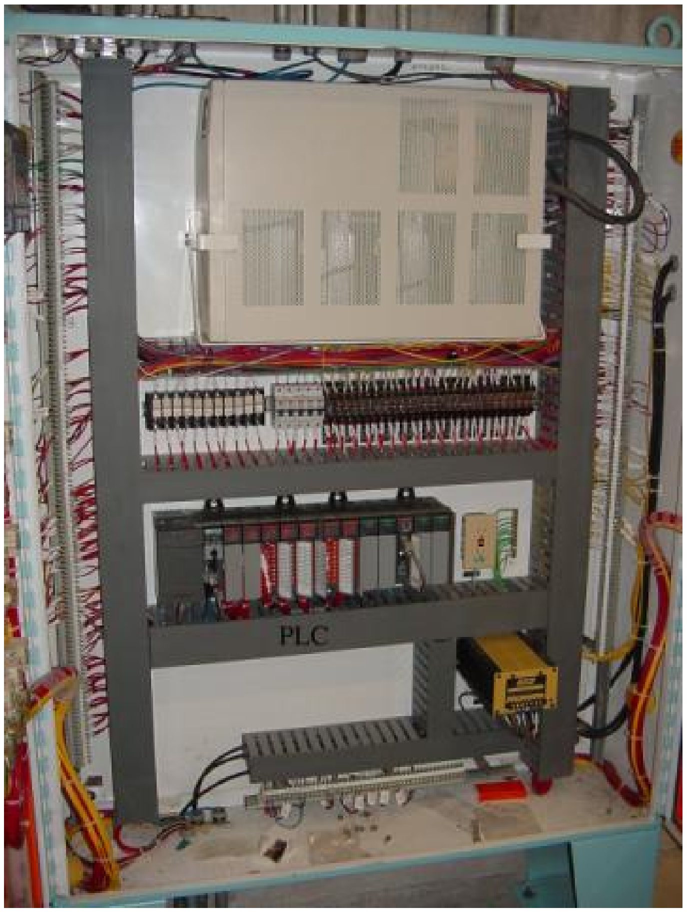


In this circuit, the lamp will remain lit so long as *any* of the pushbuttons remain unactuated (unpressed). To make the lamp turn off, we will have to actuate (press) *all* three switches, like this:



This section on programmable logic controllers illustrates just a small sample of their capabilities. As computers, PLCs can perform timing functions (for the equivalent of time-delay relays), drum sequencing, and other advanced functions with far greater accuracy and reliability than what is possible using electromechanical logic devices. Most PLCs have the capacity for far more than six inputs and six outputs. The following photograph shows several input and output modules of a single Allen-Bradley PLC.

With each module having sixteen "points" of either input or output, this PLC has the ability to monitor and control dozens of devices. Fit into a control cabinet, a PLC takes up little room, especially considering the equivalent space that would be needed by electromechanical relays to perform the same functions:



One advantage of PLCs that simply *cannot* be duplicated by electromechanical relays is remote monitoring and control via digital computer networks. Because a PLC is nothing more than a special-purpose digital computer, it has the ability to communicate with other computers rather easily. The following photograph shows a personal computer displaying a graphic image of a real liquid-level process (a pumping, or "lift," station for a municipal wastewater treatment system) controlled by a PLC. The actual pumping station is located miles away from the personal computer display:

PLC compared with other control systems

Allen-Bradley PLC installed in a control panel

PLCs are well-adapted to a range of automation tasks. These are typically industrial processes in manufacturing where the cost of developing and maintaining the automation system is high relative to the total cost of the automation, and where changes to the system would be expected during its operational life. PLCs contain input and output devices compatible with industrial pilot devices and controls; little electrical design is required, and the design problem centers on expressing the desired sequence of operations. PLC applications are typically highly customized systems so the cost of a packaged PLC is low compared to the cost of a specific custom-built controller design. On the other hand, in the case of mass-produced goods, customized control systems are economic due to the lower cost of the components, which can be optimally chosen instead of a "generic" solution, and where the non-recurring engineering charges are spread over thousands or millions of units.

For high volume or very simple fixed automation tasks, different techniques are used. For example, a consumer dishwasher would be controlled by an electromechanical cam timer costing only a few dollars in production quantities.

A microcontroller-based design would be appropriate where hundreds or thousands of units will be produced and so the development cost (design of power supplies, input/output hardware and necessary testing and certification) can be spread over many sales, and where the end-user would not need to alter the control. Automotive applications are an example; millions of units are built each year, and very few end-users alter the programming of these controllers. However, some specialty vehicles such as transit buses economically use PLCs instead of custom-designed controls, because the volumes are low and the development cost would be uneconomic.[5]

Very complex process control, such as used in the chemical industry, may require algorithms and performance beyond the capability of even high-performance PLCs. Very high-speed or precision controls may also require customized solutions; for example, aircraft flight controls. Single-board computers using semi-customized or fully proprietary hardware may be chosen for very demanding control applications where the high development and maintenance cost can be supported. "Soft PLCs" running on desktop-type computers can interface with industrial I/O hardware while executing programs within a version of commercial operating systems adapted for process control needs.[5]

Programmable controllers are widely used in motion control, positioning control and torque control. Some manufacturers produce motion control units to be integrated with PLC so that G-code (involving a CNC machine) can be used to instruct machine movements.[*citation needed*]

PLCs may include logic for single-variable feedback analog control loop, a "proportional, integral, derivative" or "PID controller". A PID loop could be used to control the temperature of a manufacturing process, for example. Historically PLCs were usually configured with only a few analog control loops; where processes required hundreds or thousands of loops, a distributed control system (DCS) would instead be used. As PLCs have become more powerful, the boundary between DCS and PLC applications has become less distinct.

PLCs have similar functionality as Remote Terminal Units. An RTU, however, usually does not support control algorithms or control loops. As hardware rapidly becomes more powerful and cheaper, RTUs, PLCs and DCSs are increasingly beginning to overlap in responsibilities, and many vendors sell RTUs with PLC-like features and vice versa. The industry has standardized on the IEC 61131-3 functional block language for creating programs to run on RTUs and PLCs, although nearly all vendors also offer proprietary alternatives and associated development environments.

In recent years "Safety" PLCs have started to become popular, either as standalone models (Pilz PNOZ Multi, Sick etc.) or as functionality and safety-rated hardware added to existing controller architectures (Allen Bradley Guardlogix, Siemens F-series etc.). These differ from conventional PLC types as being suitable for use in safety-critical applications for which PLCs have traditionally been supplemented with hard-wired safety relays. For example, a Safety PLC might be used to control access to a robot cell with trapped-key access, or perhaps to manage the shutdown response to an emergency stop on a conveyor production line. Such PLCs typically have a restricted regular instruction set augmented with safety-specific instructions designed to interface with emergency stops, light screens and so forth. The flexibility that such systems offer has resulted in rapid growth of demand for these controllers.

Digital and analog signals

Digital or discrete signals behave as binary switches, yielding simply an On or Off signal (1 or 0, True or False, respectively). Push buttons, limit switches, and photoelectric sensors are examples of devices providing a discrete signal. Discrete signals are sent using either voltage or current, where a specific range is designated as *On* and another as *Off*. For example, a PLC might use 24 V DC I/O, with values above 22 V DC representing *On*, values below 2VDC representing *Off*, and intermediate values undefined. Initially, PLCs had only discrete I/O.

Analog signals are like volume controls, with a range of values between zero and full-scale. These are typically interpreted as integer values (counts) by the PLC, with various ranges of accuracy depending on the device and the number of bits available to store the data. As PLCs typically use 16-bit signed binary processors, the integer values are limited between -32,768 and +32,767. Pressure, temperature, flow, and weight are often represented by analog signals. Analog signals can use voltage or current with a magnitude proportional to the value of the process signal. For example, an analog 0 - 10 V input or 4-20 mA would be converted into an integer value of 0 - 32767.

Current inputs are less sensitive to electrical noise (i.e. from welders or electric motor starts) than voltage inputs.

Example

As an example, say a facility needs to store water in a tank. The water is drawn from the tank by another system, as needed, and our example system must manage the water level in the tank.

Using only digital signals, the PLC has two digital inputs from float switches (Low Level and High Level). When the water level is above the switch it closes a contact and passes a signal to an input. The PLC uses a digital output to open and close the inlet valve into the tank.

When the water level drops enough so that the Low Level float switch is off (down), the PLC will open the valve to let more water in. Once the water level rises enough so that the High Level switch is on (up), the PLC will shut the inlet to stop the water from overflowing. This rung is an example of seal-in (latching) logic. The output is sealed in until some condition breaks the circuit.

```
|                                                             |
|   Low Level      High Level                Fill Valve       |
|------[/]------|------[/]----------------------(OUT)---------|
|               |                                             |
|               |                                             |
|               |                                             |
|   Fill Valve  |                                             |
|------[ ]------|                                             |
|                                                             |
|                                                             |
```

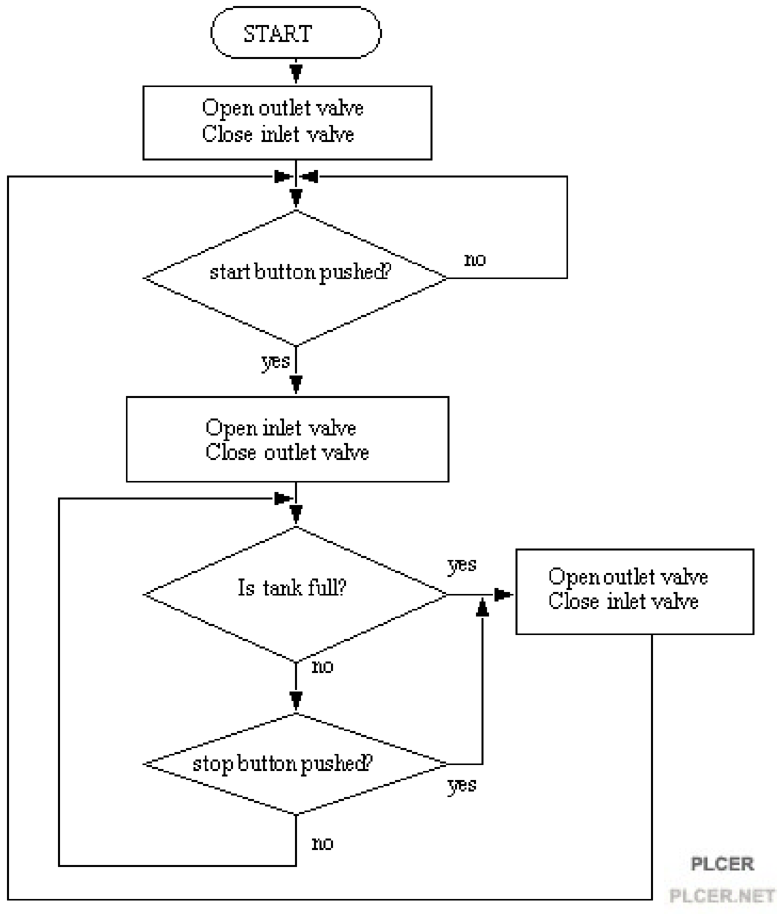
An analog system might use a water pressure sensor or a load cell, and an adjustable (throttled) control (e.g. by a valve) of the fill of the tank.

In this system, to avoid 'flutter' adjustments that can wear out the valve, many PLCs incorporate "hysteresis" which essentially creates a "deadband" of activity. A technician adjusts this dead band so the valve moves only for a significant change in rate. This will in turn minimize the motion of the valve, and reduce its wear.

A real system might combine both approaches, using float switches and simple valves to prevent spills, and a rate sensor and rate valve to optimize refill rates and prevent water hammer. Backup and maintenance methods can make a real system very complicated.

Flowchart for a Tank Filler PLC program

flowchart is shown in See A
Flowchart for a Tank Filler for a control system for a large water tank. When a start button is pushed the tank will start to fill, and the flow out will be stopped. When full, or the stop button is pushed the outlet will open up, and the flow in will be stopped. In the flowchart the general flow of execution starts at the top. The first operation is to open the outlet valve and close the inlet valve. Next, a single decision block is used to wait for a button to be pushed. when the button is pushed the yes branch is followed and the inlet valve is opened, and the outlet valve is closed. Then the flow chart goes into a loop that uses two decision blocks to wait until the tank is full, or the stop button is pushed. If either case occurs the inlet valve is closed and the outlet valve is opened. The system then goes back to wait for the start button to be pushed again. When the controller is on the program should always be running, so only a start block is needed. Many beginners will neglect to put in checks for stop buttons.



A Flowchart for a Tank Filler

References

1. ^ E. A. Parr, *Industrial Control Handbook*, Industrial Press Inc., 1999 ISBN 0831130857
2. ^ ***a b*** M. A. Laughton, D. J. Warne (ed), *Electrical Engineer's Reference book, 16th edition*,Newnes, 2003 Chapter 16 *Programmable Controller*
3. ^ "The father of invention: Dick Morley looks back on the 40th anniversary of the PLC". *Manufacturing Automation*. 12 September 2008. http://www.automationmag.com/programable-control/features/the-father-of-invention-dick-morley-looks-back-on-the-40th-anniversary-of-the-plc.html.
4. ^ ***a b*** W. Bolton, *Programmable Logic Controllers, Fifth Edition*, Newnes, 2009 ISBN 978-1-85617-751-1, Chapter 1
5. ^ ***a b*** Gregory K. McMillan, Douglas M. Considine (ed), *Process/Industrial Instruments and Controls Handbook Fifth Edition*, McGraw-Hill, 1999 ISBN 0-07-012582-1 Section 3 *Controllers*

Further reading

- Daniel Kandray, *Programmable Automation Technologies*, Industrial Press, 2010 ISBN 978-0-8311-3346-7, Chapter 8 *Introduction to Programmable Logic Controllers*