

Programmable logic controllers

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.

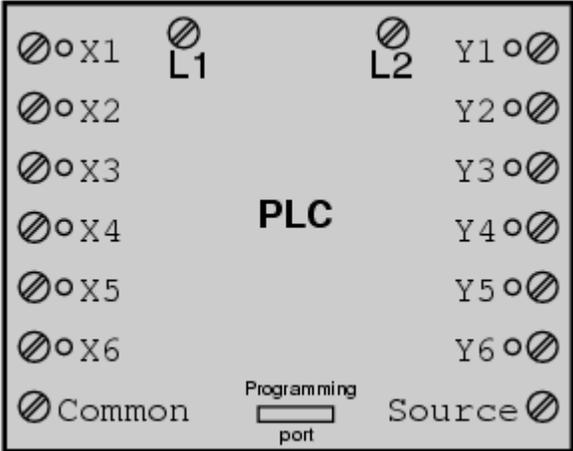
Systems and processes requiring "on/off" control abound in modern commerce and industry, but such control systems are rarely built from either electromechanical relays or discrete logic gates. Instead, digital computers fill the need, which may be *programmed* to do a variety of logical functions.

In the late 1960's an American company named Bedford Associates released a computing device they called the *MODICON*. As an acronym, it meant **ModularDigital Controller**, 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 **Programmable Logic Controller**. 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.

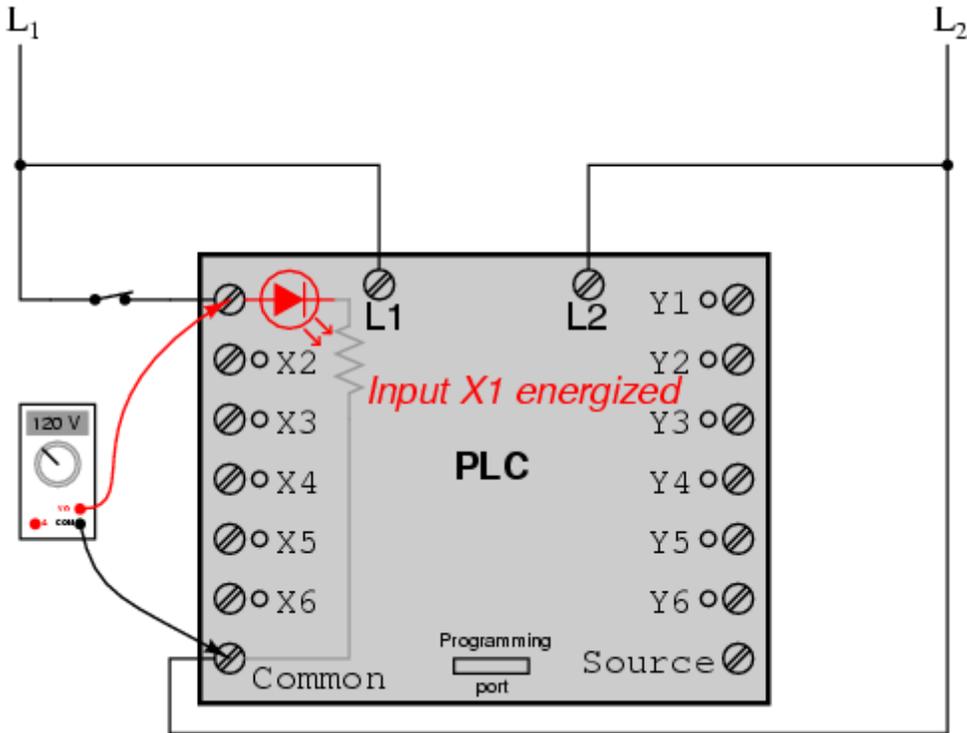
A PLC has many "input" terminals, through which it interprets "high" and "low" logical states from sensors and switches. It also has many output terminals, through which it outputs "high" and "low" signals to power lights, solenoids, contactors, small motors, and other devices lending themselves to on/off 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.

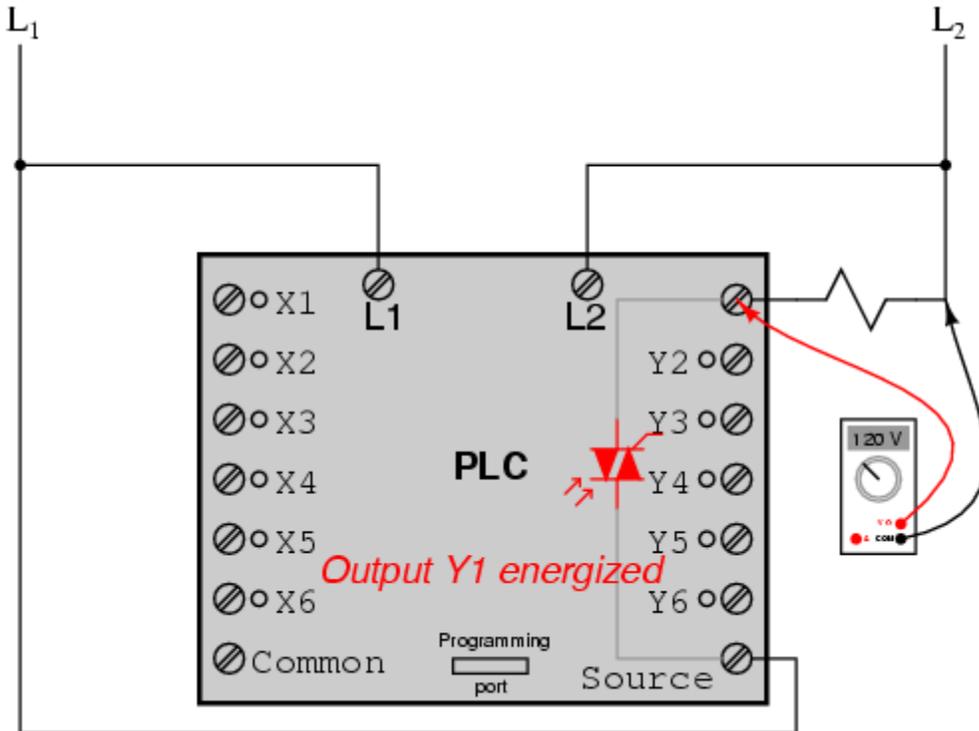
Signal connection and programming standards vary somewhat between different models of PLC, but they are similar enough to allow a "generic" introduction to PLC programming here. The following illustration shows a simple PLC, as it might appear from a front view. Two screw terminals provide connection 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:



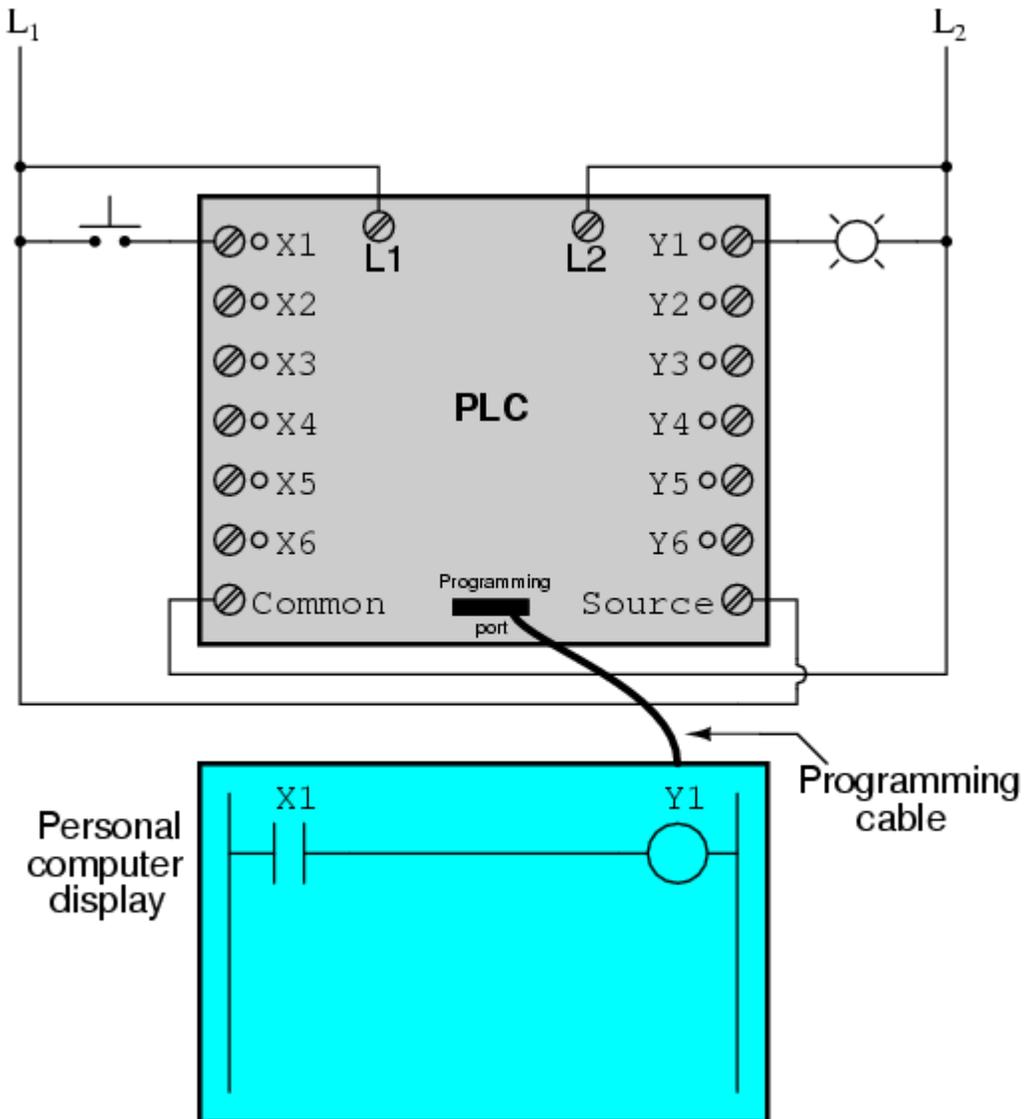
Output signals are generated by the PLC's computer circuitry activating a switching device (transistor, TRIAC, or even an electromechanical relay), connecting the "Source" terminal to any of the "Y-" labeled output terminals. The "Source" terminal, correspondingly, is usually connected to the L1 side 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.

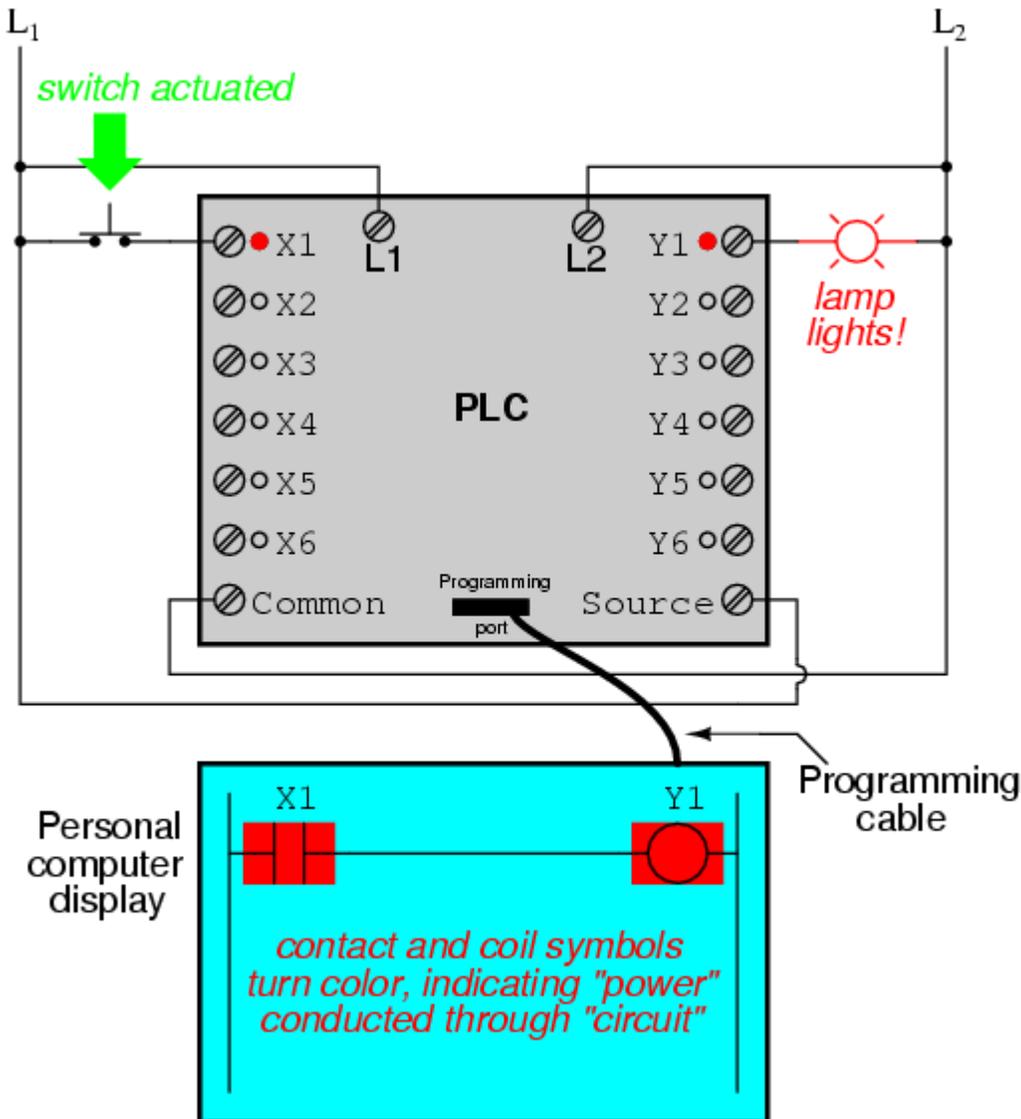
The actual *logic* of the control system is established inside the PLC by means of a computer program. This program dictates which output gets energized under which input conditions. Although the program itself appears to be a ladder logic diagram, with switch and relay symbols, there are no actual switch 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:



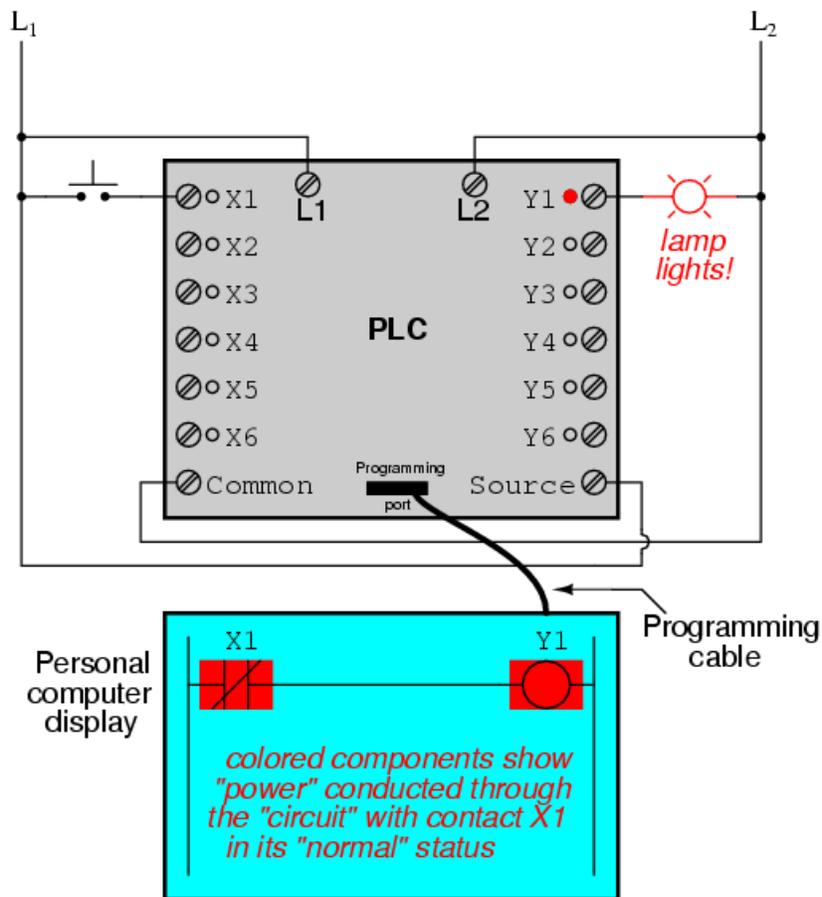
It must be understood that the X1 contact, Y1 coil, connecting wires, and "power" appearing in the personal computer's display are all *virtual*. They do not exist as real electrical components. They exist as commands in a computer program -- a piece of software only -- that just happens to resemble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

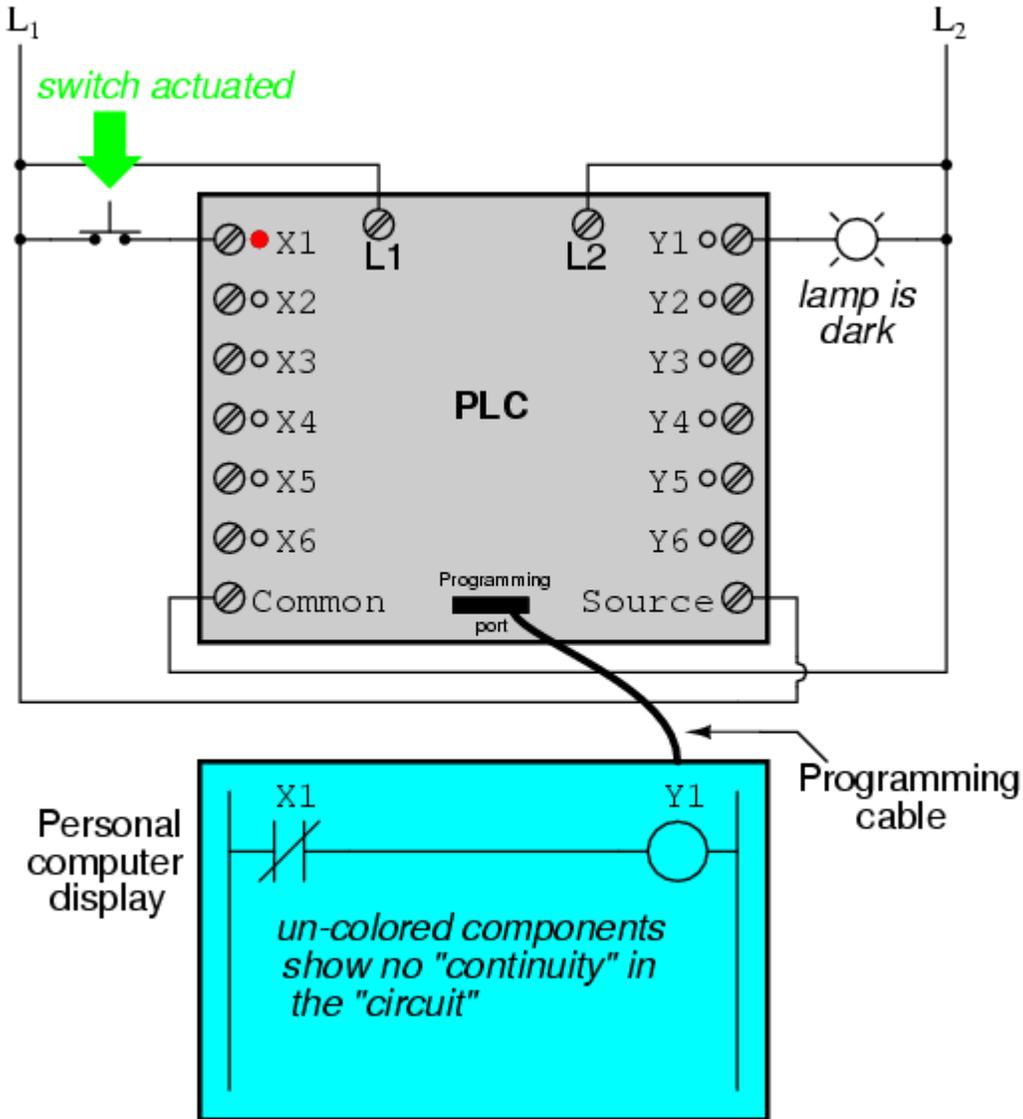
real-life conditions (switch closure and lamp status) and the program's status ("power" through virtual contacts and virtual coils).

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.

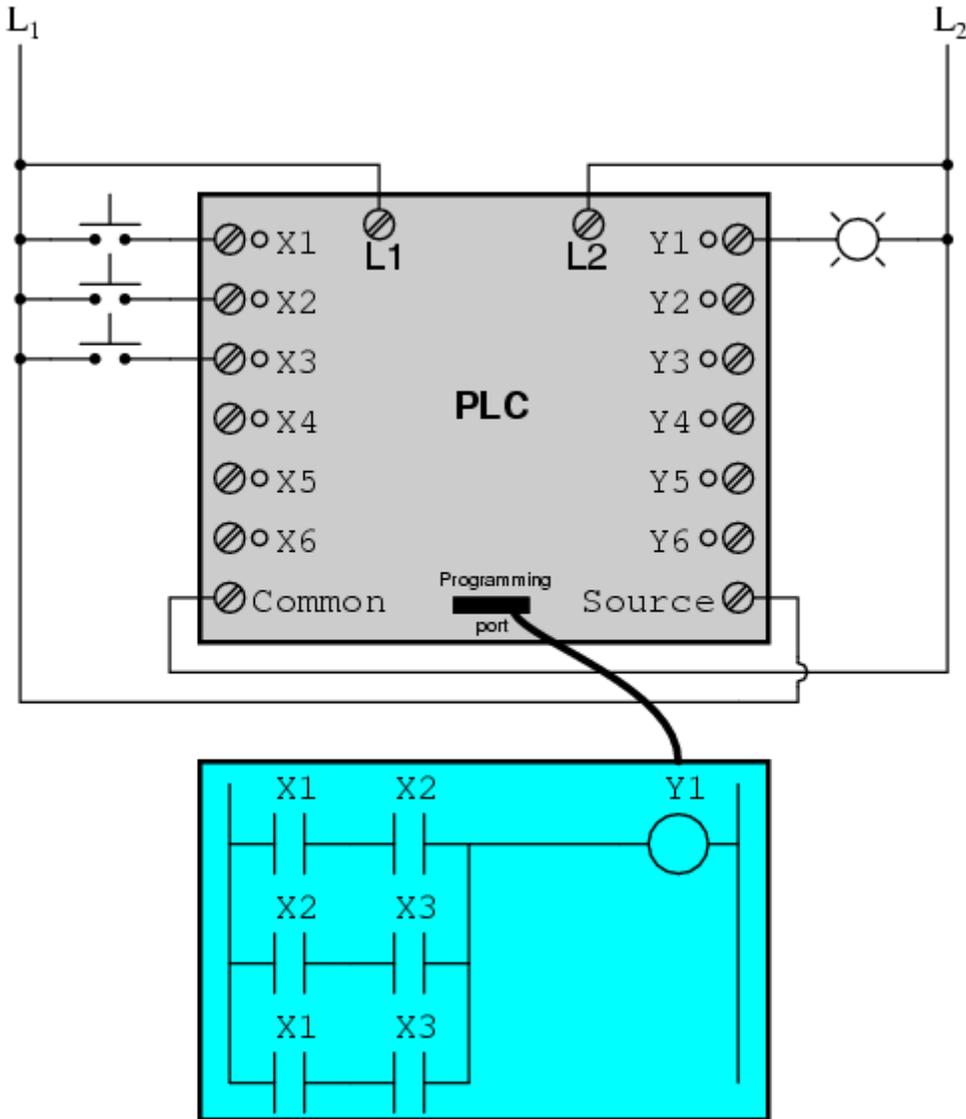
In the following illustration, we have the altered system shown in the state where the pushbutton is unactuated (*not* being pressed):



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

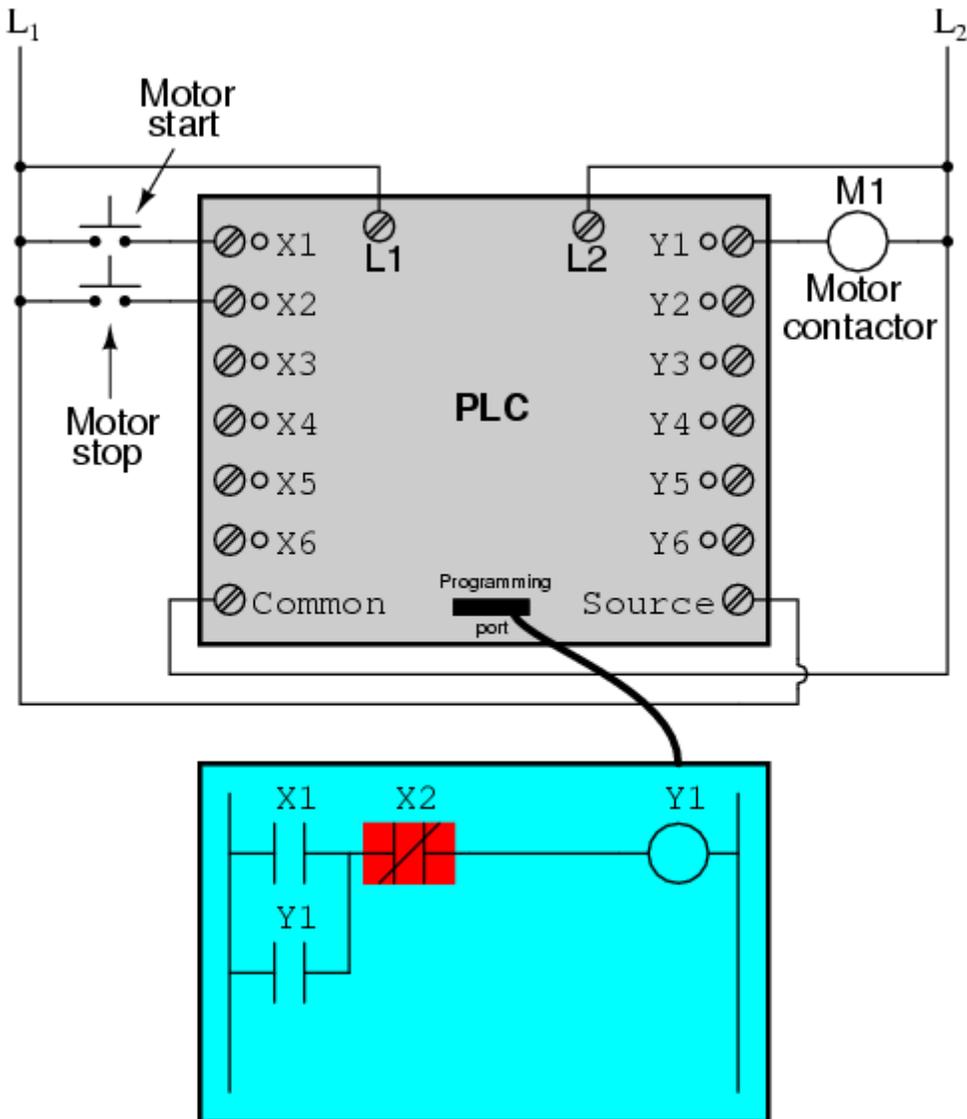


One of the advantages of implementing logical control in software rather than in hardware is that input signals can be re-used as many times in the program as is necessary. For example, take the following circuit and program, designed to energize the lamp if at least two of the three pushbutton switches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



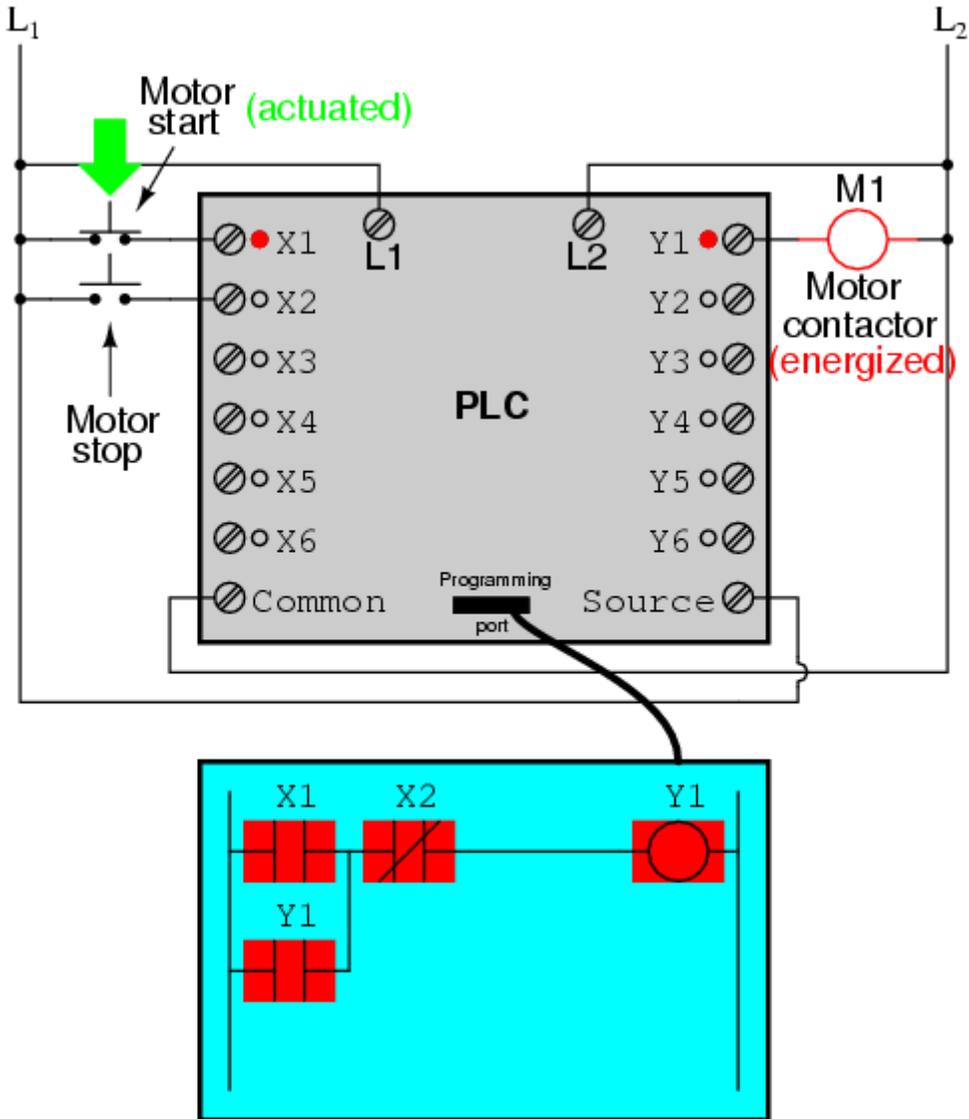
To build an equivalent circuit using electromechanical relays, three relays with two normally-open contacts each would have to be used, to provide two contacts per input switch. Using a PLC, however, we can program as many contacts as we wish for each "X" input without adding additional hardware, since 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:

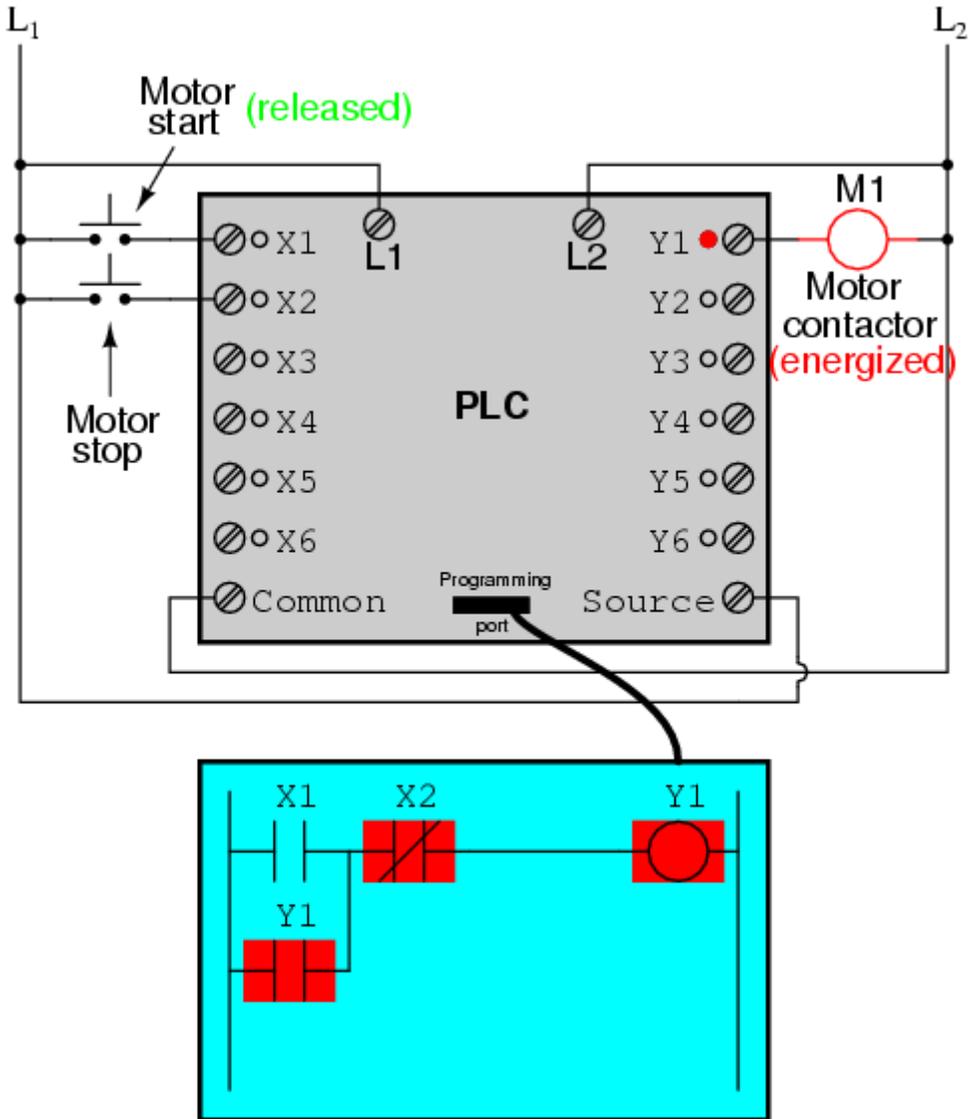


The pushbutton switch connected to input X1 serves as the "Start" switch, while the switch connected to input X2 serves as the "Stop." Another contact in the program, named Y1, uses the output coil status as a seal-in contact, directly, so that the motor contactor will continue to be energized after 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.

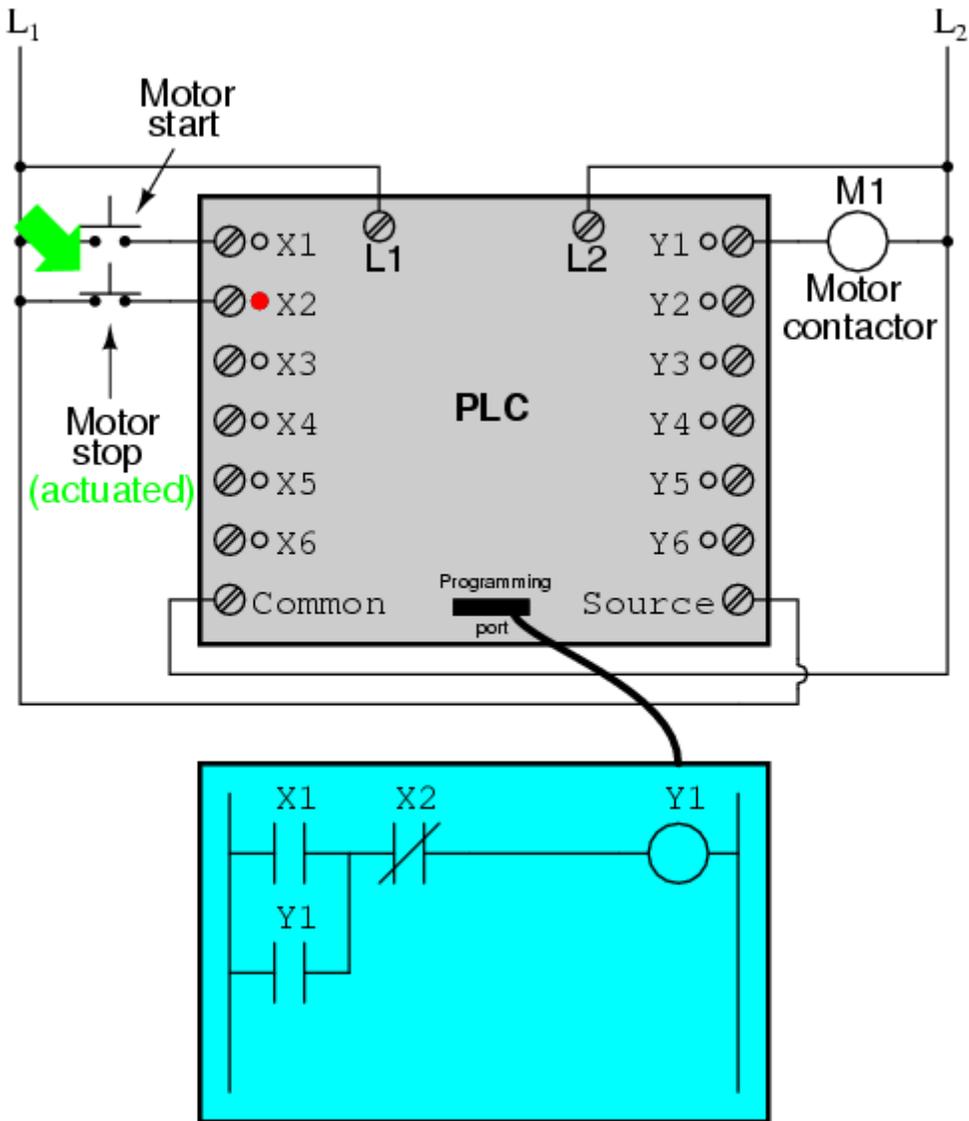
If we were to press the "Start" button, input X1 would energize, thus "closing" the X1 contact in the program, sending "power" to the Y1 "coil," energizing the Y1 output and applying 120 volt AC power to the real motor contactor coil. The parallel Y1 contact will also "close," thus latching the "circuit"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



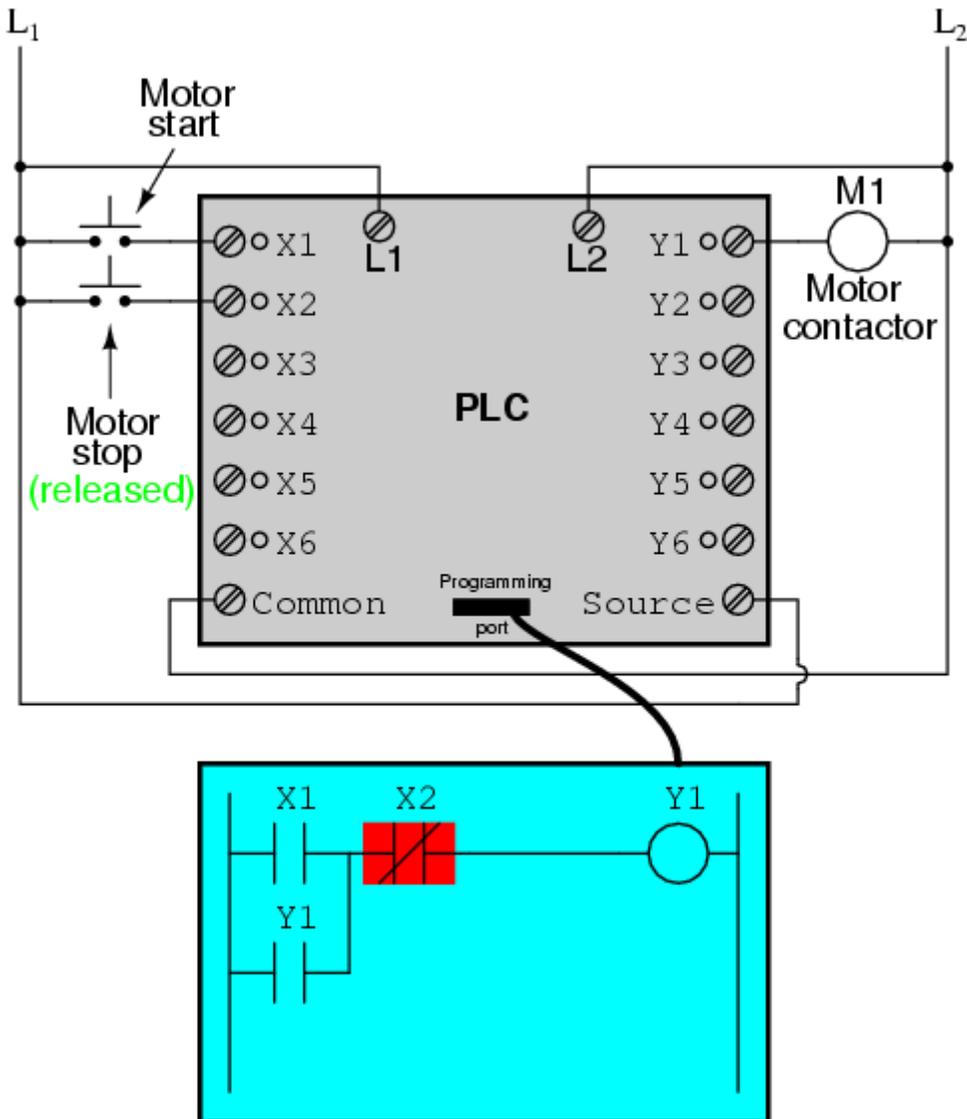
Now, if we release the "Start" pushbutton, the normally-open X1 "contact" will return to its "open" state, but the motor will continue to run because the Y1 seal-in "contact" continues to provide "continuity" to "power" coil Y1, thus keeping the Y1 output energized:



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:"

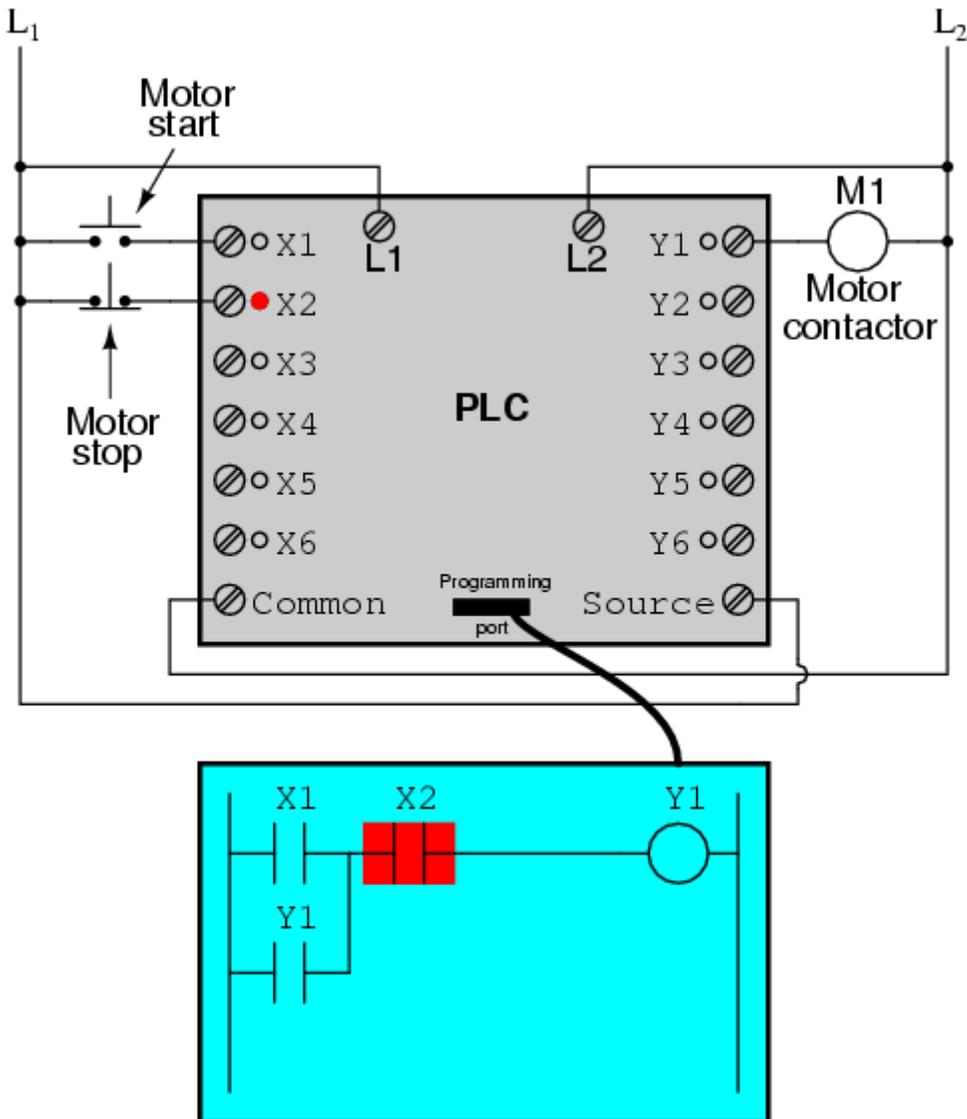


When the "Stop" pushbutton is released, input X2 will de-energize, returning "contact" X2 to its normal, "closed" state. The motor, however, will not start again until the "Start" pushbutton is actuated, because the "seal-in" of Y1 has been lost:



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:



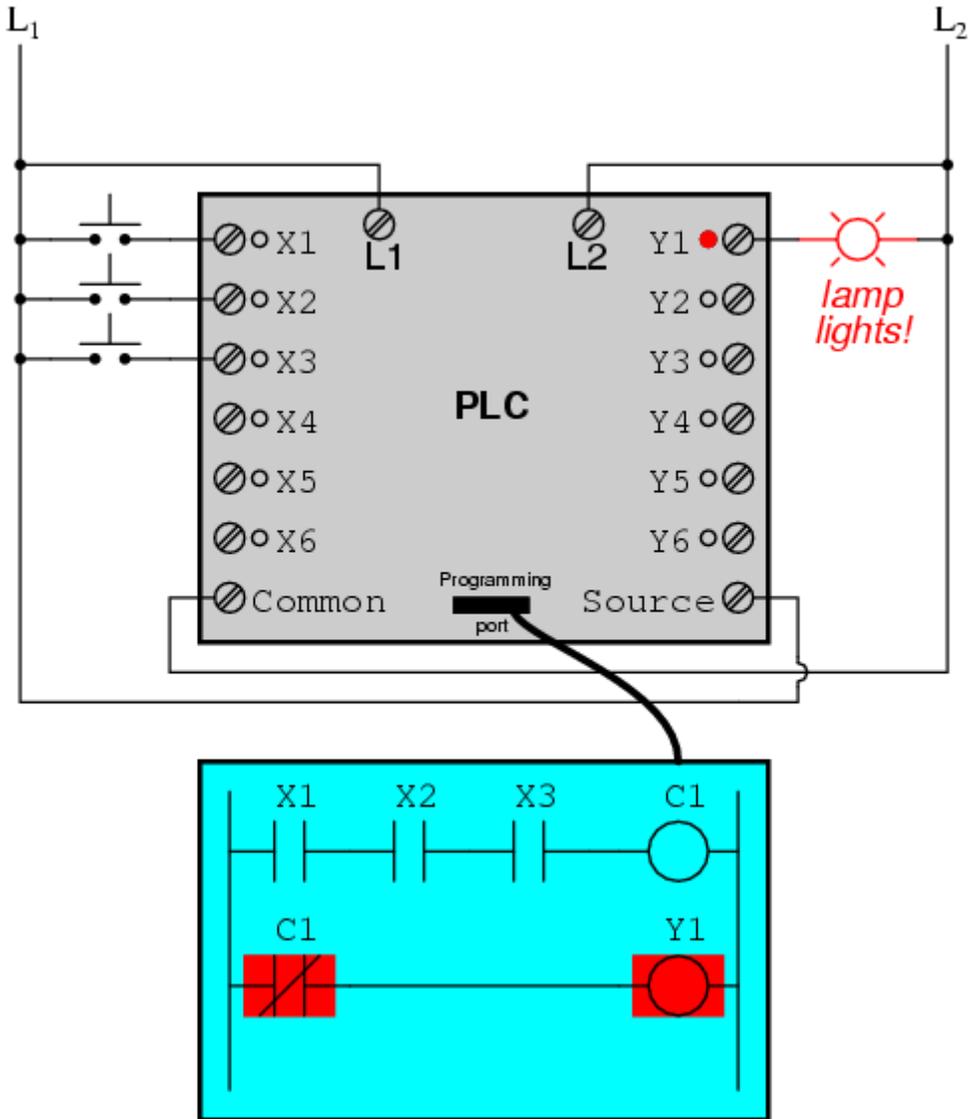
When the normally-closed "Stop" pushbutton switch is unactuated (not pressed), the PLC's X2 input will be energized, thus "closing" the X2 "contact" inside the program. This allows the motor to be started when input X1 is energized, and allows it to continue to run when the "Start" pushbutton is no longer 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.

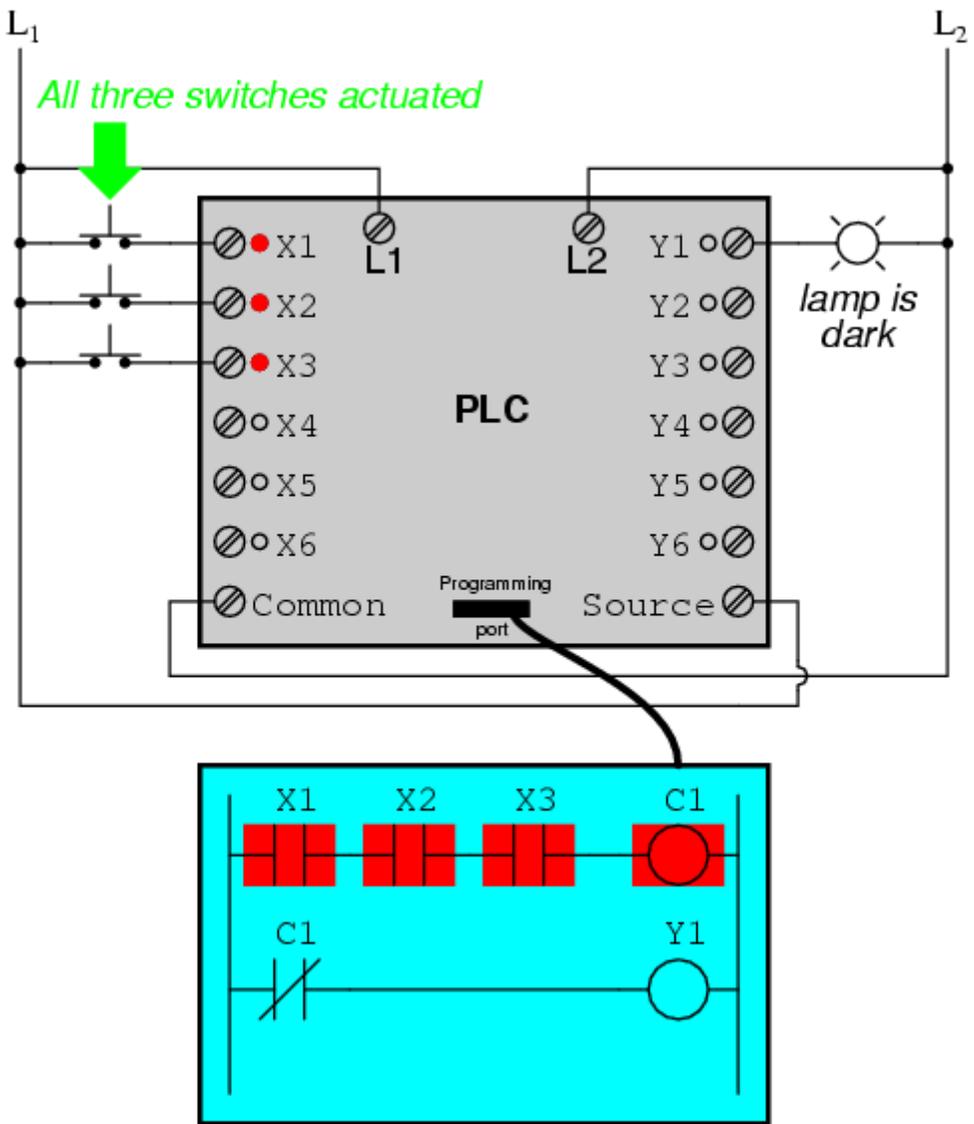
This is a safer design than the one previously shown, where a "Stop" switch wiring failure would have resulted in an *inability* to turn off the motor.

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:



Source: http://www.allaboutcircuits.com/vol_4/chpt_6/6.html