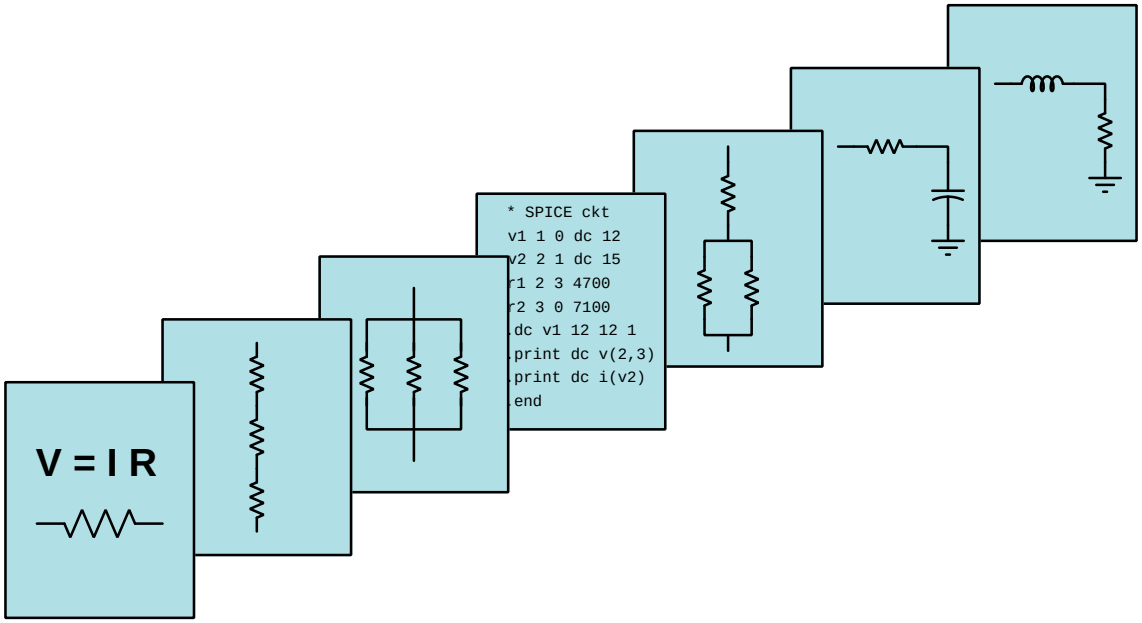
Modular Electronics Learning (ModEL) project



PLC Inputs and Outputs

© 2025-2026 by Tony R. Kuphaldt – under the terms and conditions of the Creative Commons Attribution 4.0 International Public License

Last update = 29 January 2026

This is a copyrighted work, but licensed under the Creative Commons Attribution 4.0 International Public License. A copy of this license is found in the last Appendix of this document. Alternatively, you may visit `http://creativecommons.org/licenses/by/4.0/` or send a letter to Creative Commons: 171 Second Street, Suite 300, San Francisco, California, 94105, USA. The terms and conditions of this license allow for free copying, distribution, and/or modification of all licensed works by the general public.

Contents

1 Introduction **3**

1.1 Recommendations for students . . . 3

1.2 Challenging concepts related to programmable logic controllers (PLCs) . . . 5

1.3 Recommendations for instructors . . . 6

2 Case Tutorial **7**

2.1 Example: NAND function in a PLC . . . 8

2.2 Example: simple PLC comparisons . . . 10

3 Tutorial **13**

3.1 Review of basic PLC functionality . . . 13

3.2 Types and sizes of PLCs . . . 21

3.3 PLC hardware inputs and outputs (I/O) . . . 30

3.4 Discrete PLC I/O . . . 32

3.5 Analog PLC I/O . . . 43

3.6 Network PLC I/O . . . 44

4 Derivations and Technical References **45**

4.1 Feature comparisons between PLC models . . . 46

4.1.1 Viewing live values . . . 46

4.1.2 Forcing live values . . . 47

4.1.3 Single-bit addressing . . . 47

4.1.4 Special “system” values . . . 48

4.1.5 Free-running clock pulses . . . 48

4.1.6 Standard counter instructions . . . 48

4.1.7 High-speed counter instructions . . . 49

4.1.8 Timer instructions . . . 49

4.1.9 ASCII text message instructions . . . 49

4.1.10 Analog signal scaling . . . 50

4.2 Getting started with Rockwell Studio 5000 . . . 51

4.3 Legacy Allen-Bradley memory maps and I/O addressing . . . 57

5 Questions **65**
5.1 Conceptual reasoning . . . 69
5.1.1 Reading outline and reflections . . . 70
5.1.2 Foundational concepts . . . 71
5.1.3 Relay ladder logic analogy for a PLC . . . 72
5.1.4 Sourcing versus sinking PLC I/O . . . 75
5.1.5 Sketching wires to PLC discrete I/O . . . 76
5.2 Diagnostic reasoning . . . 77
5.2.1 Incorrect PLC output wiring . . . 78

A Problem-Solving Strategies **79**

B Instructional philosophy **81**
B.1 First principles of learning . . . 82
B.2 Proven strategies for instructors . . . 83
B.3 Proven strategies for students . . . 85
B.4 Design of these learning modules . . . 86

C Tools used **89**

D Creative Commons License **93**

E Version history **101**

Index **101**

Chapter 1

Introduction

1.1 Recommendations for students

A *programmable logic controller* or *PLC* is a specialized form of industrial computer, designed to be programmed by the end user for many different types of discrete and continuous process control applications. The word "programmable" in its name reveals just why PLCs are so useful: the end-user is able to program, or instruct, the PLC to execute virtually any control function imaginable.

PLCs were introduced to industry as electronic replacements for electromechanical relay controls. In applications where relays typically control the starting and stopping of electric motors and other discrete output devices, the reliability of an electronic PLC meant fewer system failures and longer operating life. The re-programmability of a PLC also meant changes could be implemented to the control system strategy must easier than with relay circuits, where re-wiring was the only way to alter the system's function. Additionally, the computer-based nature of a PLC meant that process control data could now be communicated by the PLC over *networks*, allowing process conditions to be monitored in distant locations, and by multiple operator stations.

The legacy of PLCs as relay-replacements is probably most evident in their traditional programming language: a graphical convention known as a *Ladder Diagram*. Ladder Diagram PLC programs resemble ladder-style electrical schematics, where vertical power "rails" convey control power to a set of parallel "rung" circuits containing switch contacts and relay coils. A human being programming a PLC literally draws the diagram on the screen, using relay-contact symbols to represent instructions to read data bits in the PLC's memory, and relay-coil symbols to represent instructions writing data bits to the PLC's memory. When executing, these graphical elements become colored when "conductive" to virtual electricity, thereby indicating their status to any human observer of the program. This style of programming was developed to make it easier for industrial electricians to adapt to the new technology of PLCs. While Ladder Diagram programming definitely has limitations compared to other computer programming languages, it is relatively easy to learn and diagnose, which is why it remains popular as a PLC programming language today.

While ladder diagram programming was designed to be simple by virtue of its resemblance to relay ladder-logic schematic diagrams, this very same resemblance often creates problems for students encountering it for the very first time. A very common misconception is to think that the contact and

coil symbols shown on the editing screen of the PLC programming software are somehow *identical* or at least *directly representative* of real-world contacts and coils wired to the PLC. This is not true. Contacts and coils shown on the screen of PLC programming software applications are *instructions* for the PLC to follow, and their logical states depend both on how they are drawn in the program *and* upon their related bit states in the PLC's memory.

The relationship between a discrete sensor (e.g. switch) and the colored state of a ladder diagram element inside of a PLC follows a step-by-step chain of causation:

1. Physical closure of the discrete switch causes electricity to flow through the switch's contact.
2. This electrical current flows through the PLC input terminal wired to that switch.
3. This energization causes a corresponding bit in the PLC's memory to become "high" (1).
4. Any Ladder Diagram "contact" instruction associated with that bit will become "actuated". If the contact instruction is normally-open, the "1" bit state will "close" the contact instruction and cause it to be colored. If the contact instruction is normally-closed, the "1" bit state will "open" is and cause it to be un-colored.
5. If all elements in a rung of the Ladder Diagram program are colored, the final instruction (at the far right end of the rung) will become activated and will cause it to fulfill its function.

Here are some good questions to ask of yourself while studying this subject:

- How might an experiment be designed and conducted to test whether a switch is normally-open or normally-closed?
- How might an experiment be designed and conducted to test the "scan time" of a PLC program?
- What are some practical applications of PLCs?
- What is the "normal" status of a switch?
- How does a "two-out-of-three" alarm or shutdown system function?
- What is a "nuisance trip"?
- Are there applications where a hard-wired relay control system might actually be better than a system using a PLC?
- What purpose is served by the color highlighting feature of PLC program editing software?
- What is an HMI panel, and where would one be useful?
- How might you alter one of the example analyses shown in the text, and then determine the behavior of that altered circuit?
- Devise your own question based on the text, suitable for posing to someone encountering this subject for the first time

1.2 Challenging concepts related to programmable logic controllers (PLCs)

The following list cites concepts related to this module's topic that are easily misunderstood, along with suggestions for properly understanding them:

- **Normal status of a switch** – to say that a switch is "normally open" or "normally closed" refers to its electrical state *when at rest*, and not necessarily the state you will typically find the switch in. The root of the confusion here is the word *normal*, which most people take to mean "typical" or "ordinary". In the case of switches, though, it refers to the zero-stimulation status of the switch as it has been manufactured. In the case of PLCs it refers to the status of a "virtual switch" when its controlling bit is 0.
- **Sourcing versus Sinking output currents** – a common misconception is that since PLC outputs are called "outputs" it must mean that current only ever *exits* those terminals. This is untrue. All that "output" actually signifies is the fact that the PLC is outputting *information* consisting of voltage values measured between that terminal and ground. Sometimes the assertion of an "on" state requires that the PLC output channel actually draws current *in* through its "output" terminal!

1.3 Recommendations for instructors

This section lists realistic student learning outcomes supported by the content of the module as well as suggested means of assessing (measuring) student learning. The outcomes state what learners should be able to do, and the assessments are specific challenges to prove students have learned.

- **Outcome** – Demonstrate effective technical reading and writing

  Assessment – Students present their outlines of this module's instructional chapters (e.g. Case Tutorial, Tutorial, Historical References, etc.) ideally as an entry to a larger Journal document chronicling their learning. These outlines should exhibit good-faith effort at summarizing major concepts explained in the text.

  Assessment – Interpret elements of a real PLC ladder-logic program such as those shown in the Tutorial chapter, including identification of virtual coils, virtual contacts (normally-open and normally-closed), color-highlighting, and associated statuses of those elements based on color highlighting.

  Assessment – Students present their "Conceptual Toolbox" Theory Journal entries listing important concepts applied in the Tutorial text(s). A "Conceptual Toolbox" serves as inspiration when solving problems, each concept suggesting a potential problem-solving action. Challenge students to collaboratively generate a Conceptual Toolbox for the day's topic during the class session and then use it when solving problems together, thus fostering the habit of reasoning from principles rather than recalling procedures!

- **Outcome** – Apply the concept of "normal" contact status to real and virtual contacts in PLC systems

  Assessment – Predict the status of every PLC program virtual contact and PLC program coil in a ladder-style diagram for various input switch state combinations. Good starting points include the Case Tutorial section "Example: NAND function in a PLC".

- **Outcome** – Sketch wire connections necessary to interface a PLC to a controlled process

  Assessment – Sketch wire placement in a pictorial diagram showing how switches would connect to the input channels of a PLC and how loads would connect to the output channels of the same PLC; e.g. pose problems in the form of the "Sketching wires to PLC discrete I/O" Conceptual Reasoning question.

- **Outcome** – Independent research

  Assessment – Locate PLC I/O module datasheets and properly interpret some of the information contained in those documents including number of I/O channels, voltage and current limitations, sourcing versus sinking capability, etc.

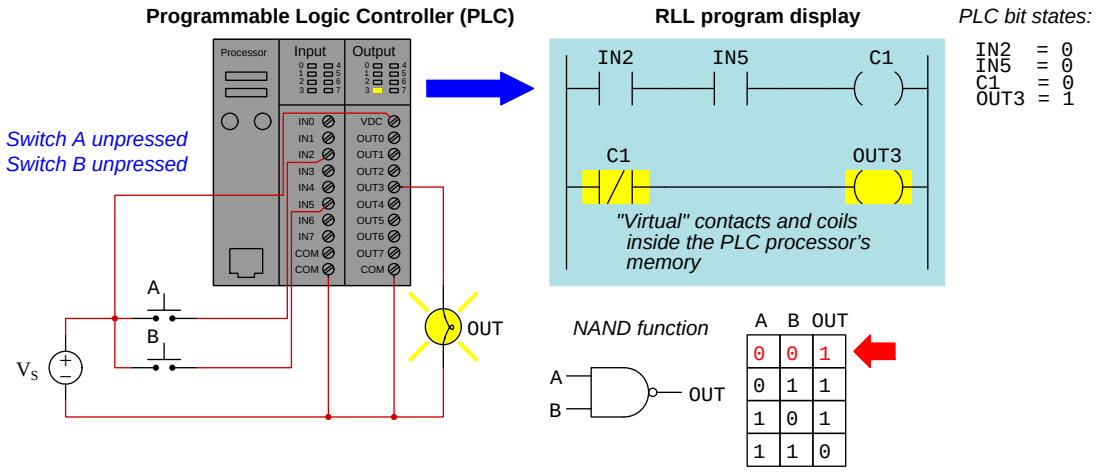
Chapter 2

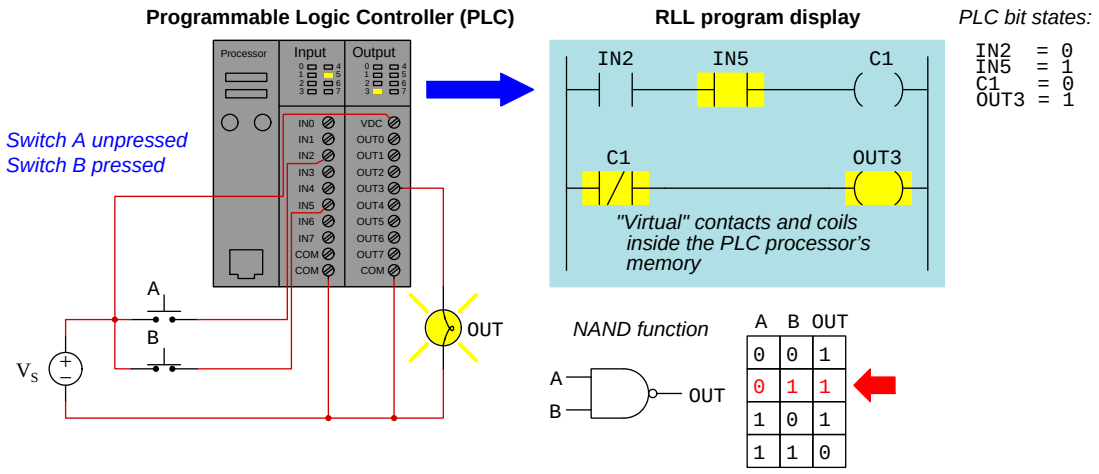
Case Tutorial

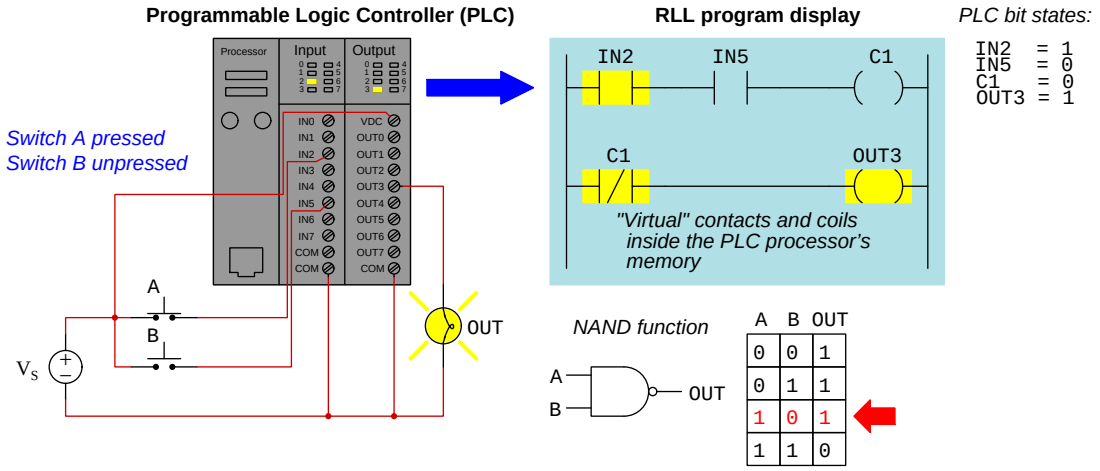
The idea behind a *Case Tutorial* is to explore new concepts by way of example. In this chapter you will read less presentation of theory compared to other Tutorial chapters, but by close observation and comparison of the given examples be able to discern patterns and principles much the same way as a scientific experimenter. Hopefully you will find these cases illuminating, and a good supplement to text-based tutorials.

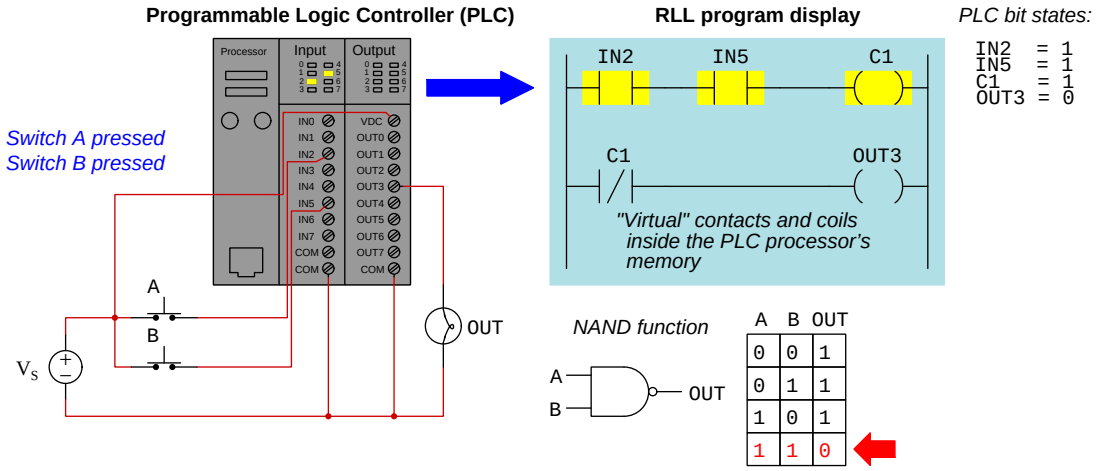
These examples also serve well as challenges following your reading of the other Tutorial(s) in this module – can you explain *why* the circuits behave as they do?

2.1 Example: NAND function in a PLC



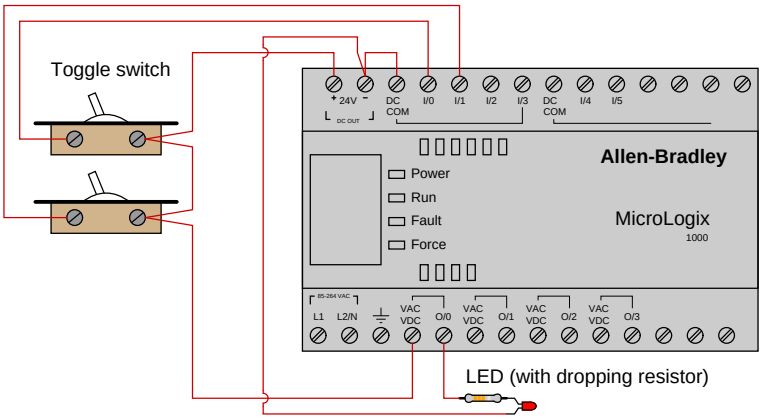



Programmable Logic Controller (PLC)
RLL program display
PLC bit states:
Processor
Input
Output
Switch A pressed
Switch B unpressed
A
B
OUT
IN2
IN5
C1
C1
OUT3
"Virtual" contacts and coils inside the PLC processor's memory
IN2 = 1
IN5 = 0
C1 = 0
OUT3 = 1
NAND function
A B OUT
0 0 1
0 1 1
1 0 1
1 1 0

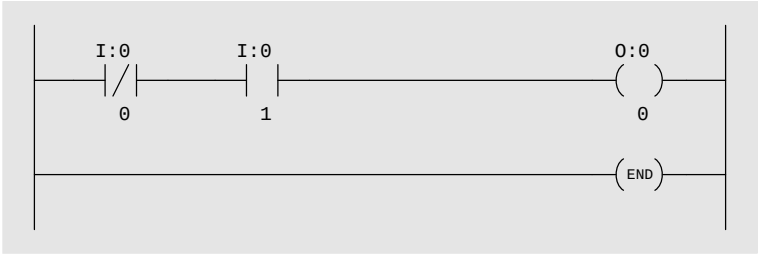

Programmable Logic Controller (PLC)
RLL program display
PLC bit states:
Processor
Input
Output
Switch A pressed
Switch B pressed
A
B
OUT
IN2
IN5
C1
C1
OUT3
"Virtual" contacts and coils inside the PLC processor's memory
IN2 = 1
IN5 = 1
C1 = 1
OUT3 = 0
NAND function
A B OUT
0 0 1
0 1 1
1 0 1
1 1 0

2.2 Example: simple PLC comparisons

The following illustration shows wiring and a sample relay ladder logic (RLL) program for an Allen-Bradley MicroLogix 1000 PLC:



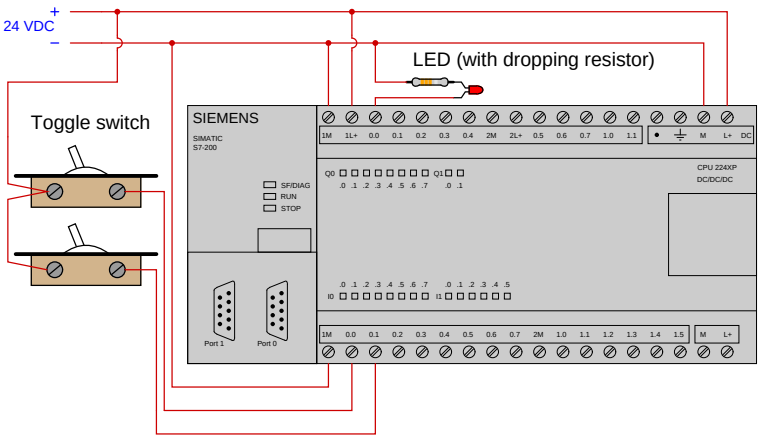
Ladder-Diagram program written to PLC:



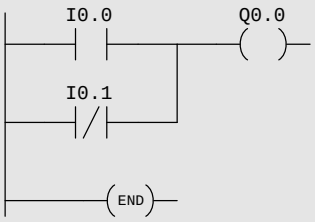
Note how Allen-Bradley I/O is labeled in the program: input bits designated by the letter `I` and output bits designated by the letter `O`.

In order to energize the LED, the switch connected to input terminal 0 must be off (open) and the switch connected to input terminal 1 must be on (closed).

The following illustration shows wiring and a sample relay ladder logic (RLL) program for a Siemens Simatic S7-200 PLC:



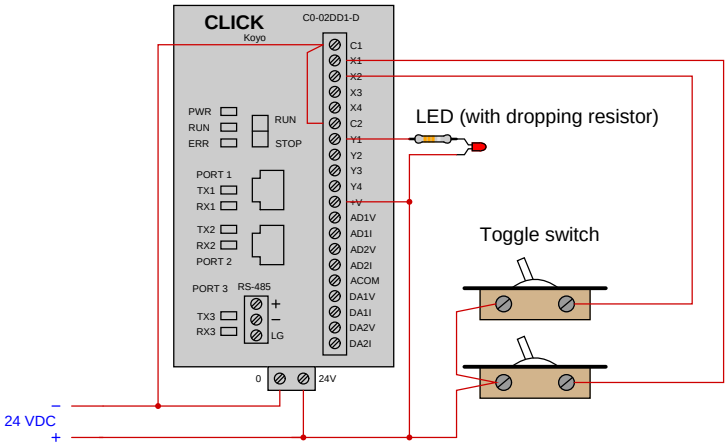
Ladder-Diagram program written to PLC:



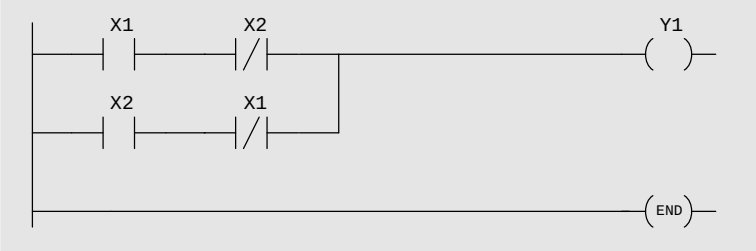
Note how Siemens I/O is labeled in the program: input bits designated by the letter `I` and output bits designated by the letter `Q`.

In order to energize the LED, either the switch connected to input terminal 0.0 must be on (closed) or the switch connected to input terminal 0.1 must be off (open).

The following illustration shows wiring and a sample relay ladder logic (RLL) program for a Koyo CLICK PLC:



Ladder-Diagram program written to PLC:



Note how Koyo I/O is labeled in the program: input bits designated by the letter `X` and output bits designated by the letter `Y`.

In order to energize the LED, at least one of the following conditions must be met:

- X1 switch turned on (closed) and X2 switch turned off (open)
- X2 switch turned on (closed) and X1 switch turned off (open)

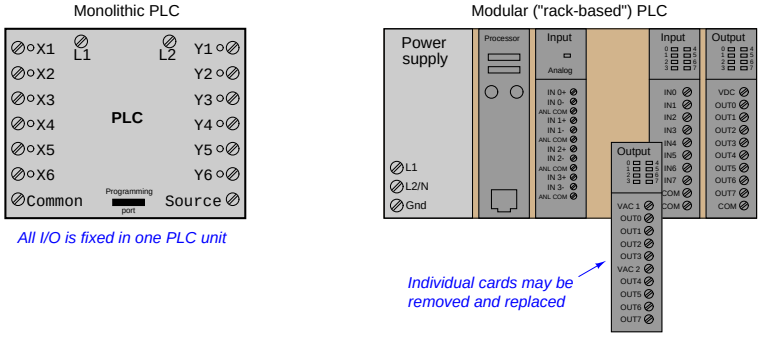
Either the switch connected to input terminal 0.0 must be on (closed) or the switch connected to input terminal 0.1 must be off (open).

Chapter 3

Tutorial

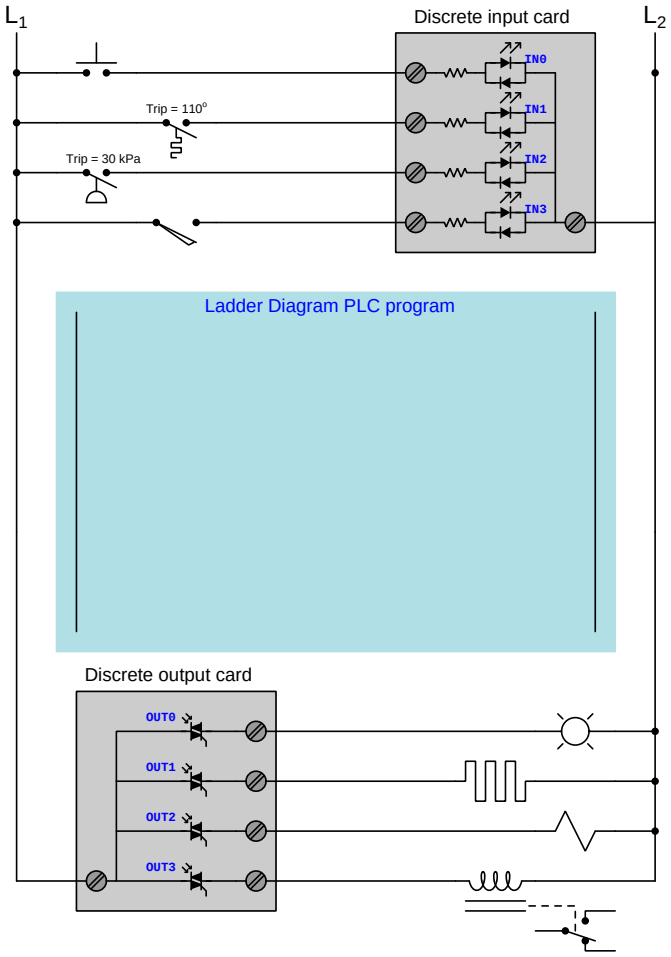
3.1 Review of basic PLC functionality

A *Programmable Logic Controller*, or *PLC*, is a general-purpose industrial computer designed to be easily programmed by end-user maintenance and engineering personnel for specific control functions. PLCs have input and output channels (often hosted on removable "I/O cards") intended to connect to field sensor and control devices such as proximity switches, pushbuttons, solenoids, lamps, sirens, etc. The user-written *program* instructs the PLC how to energize its outputs in accordance with input conditions.



PLCs were originally invented as a replacement for hard-wired relay control systems, and a popular PLC programming language called *Ladder Diagram* was invented to allow personnel familiar with relay ladder logic diagrams to write PLC programs performing the same discrete (on/off) functions as control relays. With a PLC, the discrete functionality for any system could be altered merely by editing the Ladder Diagram program rather than by re-wiring connections between physical relays.

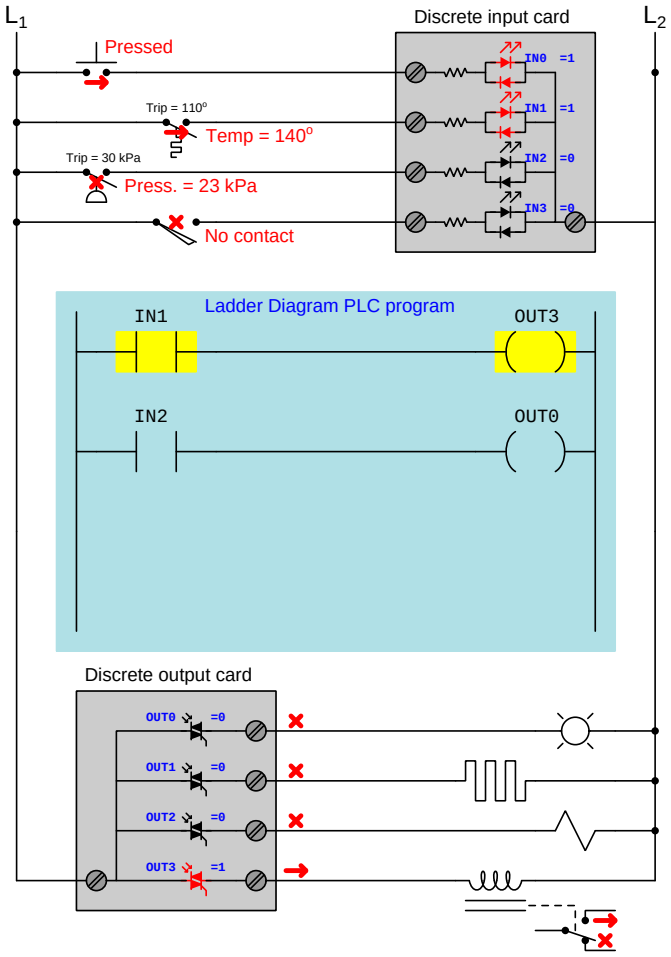
The following diagram shows a PLC separated into three sections: (1) a *discrete input card*, (2) the *program* space in the processor's memory, and (3) a *discrete output card*:



Inputs `IN0` through `IN3` are connected to a pushbutton switch, temperature switch, pressure switch, and limit switch, respectively. Outputs `OUT0` through `OUT3` connect to an indicator lamp, electric heater, solenoid coil, and electromechanical relay coil, respectively. The input card triggers[1] bits in the PLC's memory to switch from 0 to 1 when each respective input is electrically energized, and another set of bits in the PLC's memory control TRIACs inside the output card to turn on when 1 and off when 0. However, with no program installed in the processor, this PLC will not actually *do* anything. As the switch contacts open and close, the only thing the PLC will do is represent their discrete states by the bits `IN0` through `IN3` (0 = de-energized and 1 = energized).

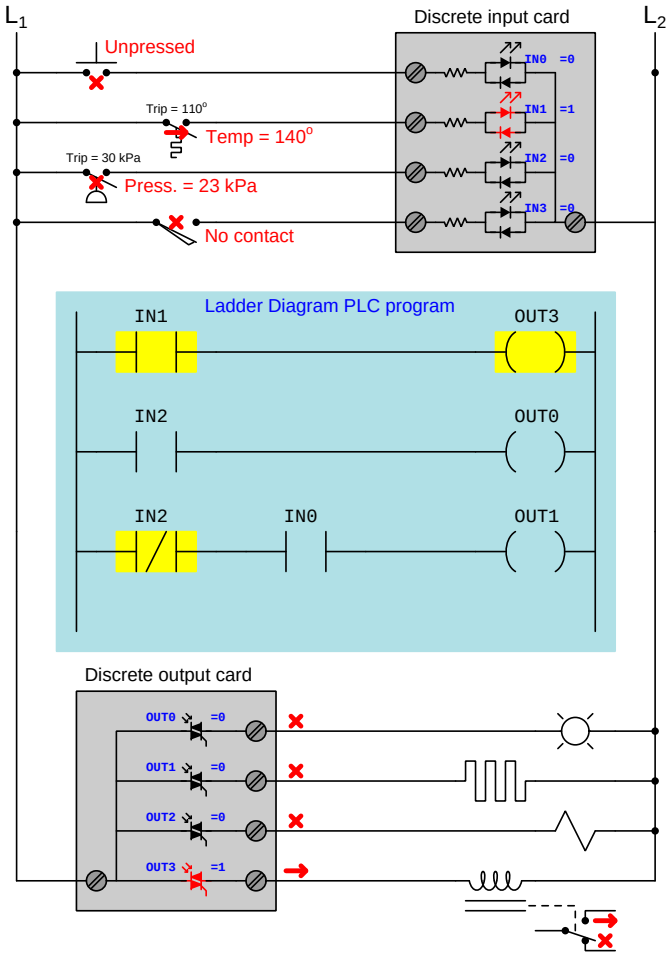
[1] Not shown in this simplified diagram are the optotransistors coupled to the LEDs inside the input card, translating each LED's state to a discrete logic level at the transistor to be interpreted by the PLC's digital processor. Likewise, another set of LEDs driven by the processor's outputs couple to the opto-TRIACs in the output card. Optical isolation of all I/O points is standard design practice for industrial PLCs.

This next diagram shows the same PLC, but this time with a very simple Ladder Diagram program running in the processor, and with stimuli applied to some of the switches:



Inputs `IN0` and `IN1` are energized by their closed switches (pushbutton and temperature), triggering those bits to "1" states in the PLC's memory. The Ladder Diagram program consists of two virtual "contact" instructions and two virtual "coil" instructions, the contact instructions controlled by input bits `IN1` and `IN2` and the coil instructions controlling output bits `OUT3` and `OUT0`. Contact instruction `IN1` "connects" (virtually) to coil instruction `OUT3`, contact `IN2` connecting to coil `OUT0` similarly. Colored highlighting shows the "virtual electricity" status of these instructions, as though they were relays being energized with real electricity. Contact instruction `IN1` is colored because it is a "normally-open" that is being stimulated into its closed state by its "1" bit status. Contact instruction `IN2` is also normally-open, but since its bit is "0" it remains uncolored, and so is the coil it's connected to. The end-result of this program is that the relay's state follows the temperature switch, and the lamp's state follows the pressure switch.

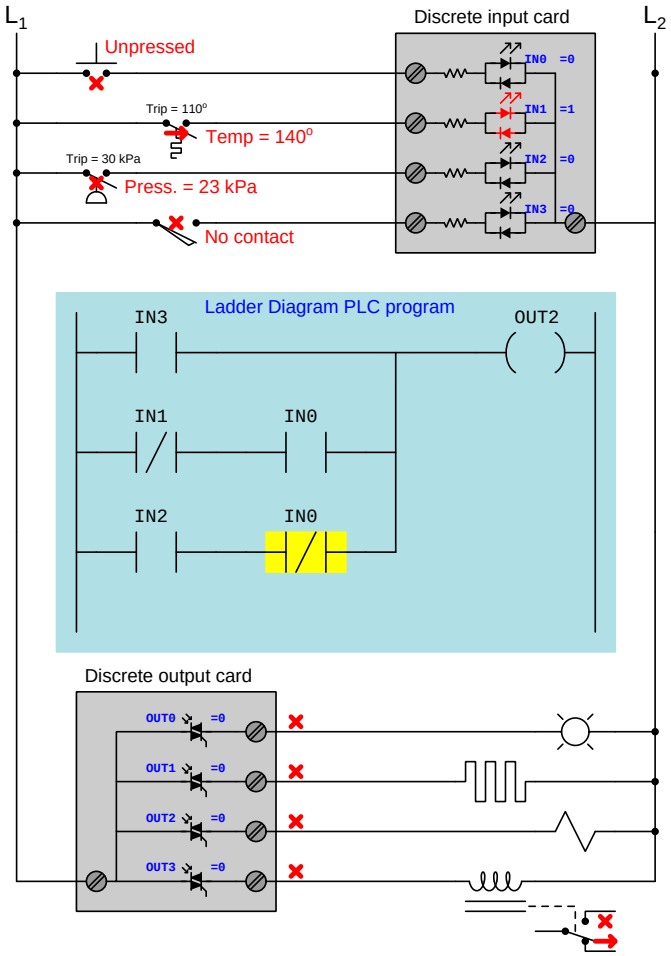
Things get more complex when we begin adding *normally-closed* contact instructions to the program. Consider this next diagram, with updated stimuli and an expanded Ladder Diagram program for the PLC to follow:



The first two rungs of the program are unchanged, as are the temperature and pressure switch statuses, and so outputs `OUT3` and `OUT0` do precisely what they did before. A new rung has been added to the program, with contact instructions linked to bits `IN2` and `IN0`, and the pushbutton switch is no longer being pressed. Both bits `IN0` and `IN2` are currently "0" and so their respective contact instructions are both in their "normal" (i.e. resting) states. The normally-closed contact instruction `IN2` is colored because it is "closed" but the `OUT1` coil in that rung is uncolored because the normally-open contact instruction `IN0` is uncolored and therefore blocks virtual electricity from reaching that coil.

Practically any logic function may be made simply by drawing virtual contact and coil instructions controlling the flow of "virtual electricity". We could describe the above program in Boolean terms: $OUT0 = IN2$; $OUT1 = (\overline{IN2})(IN0)$; $OUT2 = 0$; $OUT3 = IN1$.

This next diagram shows the same PLC with a completely re-written program. The program is now written so that the solenoid coil will energize if the limit switch makes contact, *or* if the temperature is below 110^o *and* the pushbutton is pressed, *or* if the pressure rises above 30 kPa *and* the pushbutton is unpressed:



All switch stimuli are the same as before, resulting in a "0" state for bit `OUT2` and a correspondingly de-energized solenoid. It should be clear to see how this program implements the intended AND and OR functionality by means of series-connected and parallel-connected contact instructions, respectively, with inversion (i.e. the NOT function) implemented by normally-closed rather than normally-open contact instructions.

The logical chain of causality from input to output on a PLC is very important to understand, and will be represented here by a sequence of numbered statements:

1. Energization of input channels controls input bit states (no current = 0 and current = 1)
2. Bit states control the resting/actuated status of contact instructions (0 = resting and 1 = actuated)
3. The resting/actuated status of a contact instruction, combined with its "normal" type determines virtual conductivity (open = uncolored and closed = colored)
4. Continuous color on a rung activates that rung's coil instruction
5. The coil's status controls output bits (uncolored = 0 and colored = 1)
6. Output bits control energization of output channels (0 = off and 1 = on)

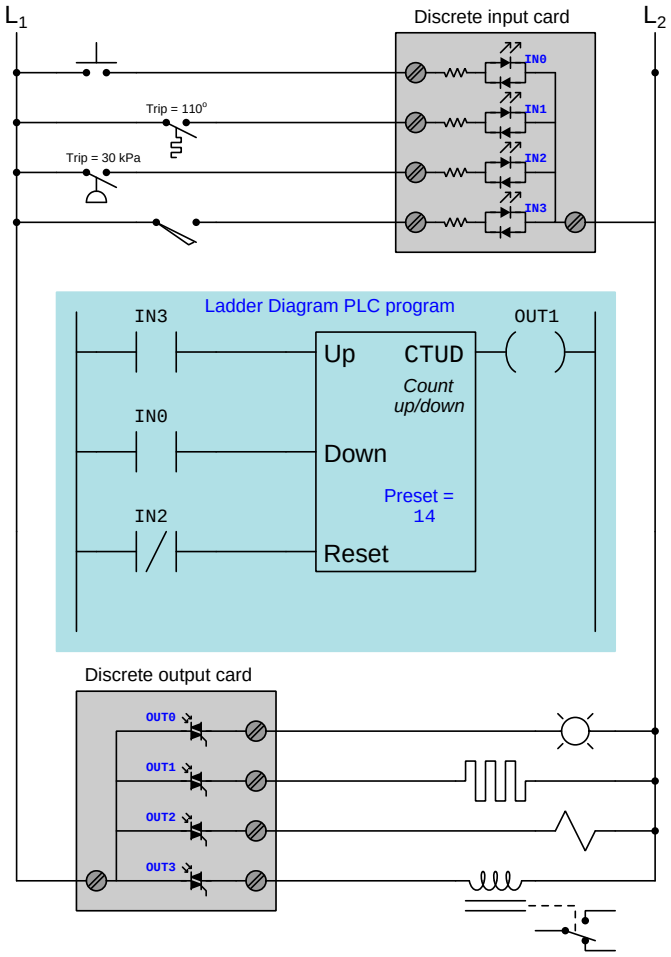
All PLCs follow this chain of logic precisely, and do so in a repeated pattern called the *scan cycle*. Each cycle begins with the PLC's processor reading the electrical states at the input terminals and updating input registers in its memory to reflect those electrical states. The processor then proceeds to evaluate the ladder-diagram program in much the same way we read sentences and paragraphs written in English: from left to right, top rung to bottom rung. Contact instructions are evaluated as either "conductive" or not based on their bit states and their type (e.g. NO or NC), then coil instructions write their states to the PLC's memory, rung by rung through the end of the program. After that the PLC turns on or off its outputs[2] to control real-world devices based on the output register states just updated by the program. Self-diagnostic and other auxiliary evaluations follow that before the entire cycle begins again. This cycle repeats as a continuous loop so long as the PLC is in the "run" mode, each cycle typically taking tens of milliseconds to repeat.

This same step-by-step causality *must* be respected in order to successfully analyze a Ladder Diagram program in a PLC. The most confusing part of this for new students seems to be the relationship of contact instructions to real-world switch inputs. Many students have an unfortunate tendency to want to directly[3] associate real-world switch status with Ladder Diagram color, and/or to believe that the "normal" status of a Ladder Diagram contact instruction must always match the "normal" status of the real-world switch. These and other such misconceptions stem from the same error, namely assuming that contact instructions somehow model or represent real-world contacts, rather than seeing contact instructions as entities *controlled* by the real-world inputs. Only by rigorously following the chain of causation from beginning to end (i.e. input energization → input bit state → contact instruction actuation → color based on normal type *and* bit state → coil color → output bit state → output energization) is one able to properly understand how PLCs function.

[2] At least this is how *most* coil instructions operate. Some PLCs offer a special type of coil instruction called an *immediate* coil that do not wait until the end of the program's scan to drive real-world outputs, but do so immediately upon evaluation mid-way through the program scan.

[3] For a normally-open contact instruction, this association is direct. However, for a normally-closed contact instruction it is inverted!

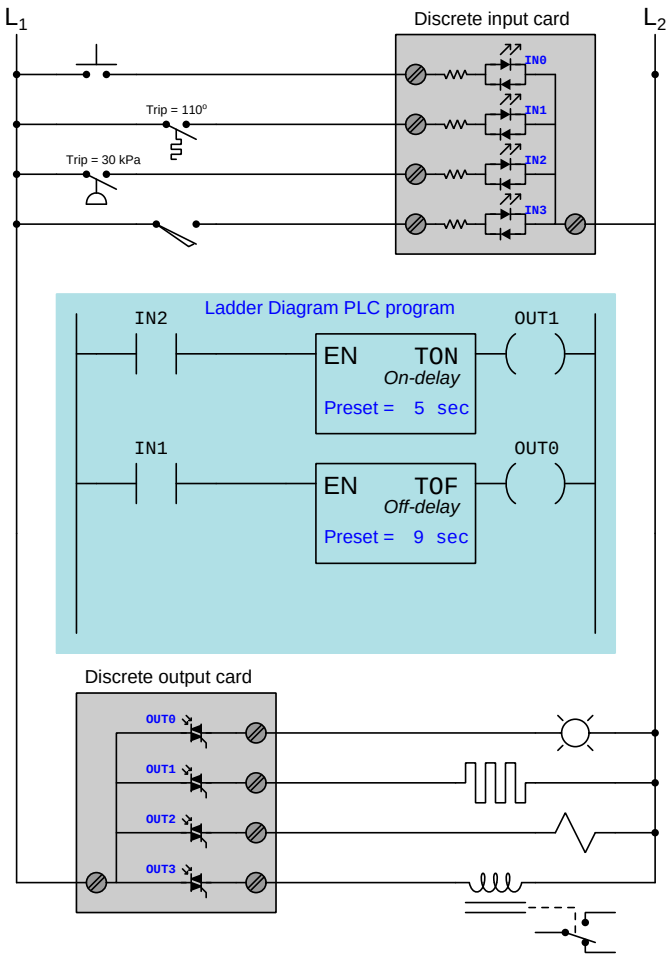
Being fully-fledged digital computers in their own right, PLCs are not limited to executing simple Boolean functions represented by "virtual relay" contacts and coils. Other digital functions include *counters* and *timers*. An example of a counter program is shown here:



The `CTUD` instruction is an *up/down counter* receiving three discrete inputs and generating one discrete output. The program is written so that this counter instruction's count value will increment (i.e. count up) once for every closure of the limit switch, decrement (i.e. count down) once for every closure of the pushbutton switch, and reset to zero if the pressure falls below 30 kPa. The output signal ("wired" to coil `OUT1`) energizes the heating element if this count value reaches or exceeds the "preset" value of 14.

The PLC's scan cycle is exactly the same for this and for all programs as it was for the last: the PLC first scans its inputs and updates those bits in its input memory registers, next scans the program rung-by-rung from left to right and top to bottom as it writes to internal and output registers, next updates all its actual outputs based on those updated output registers, and after some internal "housekeeping" tasks repeats the cycle again.

Next we see an example PLC program showcasing two *timing* instructions, an *on-delay* timer and an *off-delay* timer:



When the pressure exceeds 30 kPa and closes the pressure switch connected to input `IN2`, the `TON` timer instruction begins counting. After 5 seconds of continuous activation, output `OUT1` activates to energize the heating element. When the pressure falls below 30 kPa, the heating element immediately de-energizes.

When the temperature exceeds 110^o and closes the temperature switch connected to input `IN1`, the `TOF` timer instruction immediately activates its output (`OUT0`) to energize the indicator lamp. When the temperature cools down below 110^o, the off-delay timer begins timing and does not de-energize the indicator lamp until 9 seconds after the temperature switch has opened.

Both the utility and versatility of programmable logic controllers should be evident in this brief tutorial. These are digital computers, fully programmable by the end-user in a simple instructional language, designed to implement discrete logic functions, counting functions, timing functions, and a whole host of other useful operations for the purpose of controlling electrically-based systems. Originally designed to replace hard-wired electromechanical relay control circuits, PLCs are designed to mimic the functionality of relays while providing superior reliability and reconfigurability.

PLCs are not limited to contact, coil, counter, and timer instructions, either. A typical PLC literally offers dozens of instruction types in its set, which may be applied and combined in nearly limitless fashion. Other types of PLC programming instructions include *latch instructions* (offering bistable "set" and "reset" capability), *one-shot instructions* (outputting an active state for exactly one "scan" of the PLC's program every time the input transitions from inactive to active), *sequencers* (controlling a pre-determined sequence of discrete states based on a count value), *arithmetic instructions* (e.g. addition, subtraction, multiplication, division, etc.), *comparison instructions* (comparing two numerical values and generating a discrete signal indicating equality, inequality, etc.), *data communication instructions* (sending and receiving digital messages over a communications network), and *clock/calendar functions* (tracking time and date).

One advantage of PLCs over relay circuitry which may not be evident at first inspection is the fact that the number of virtual "contacts" and "coils" and other instructions is limited only by how much memory the PLC's processor has. The example programs shown on the previous pages were extremely short, but a real PLC program may be dozens of pages long! Electromechanical control and timing relays are, of course, limited in the number of physical switch contacts each one offers, which in turn limits how elaborate the control system may be. For the sake of illustration, a PLC with a single discrete input (say, `IN0` wired to a pushbutton switch) may contain a program with *hundreds* of virtual contacts labeled `IN0` triggering all kinds of logical, counting, and timing functions.

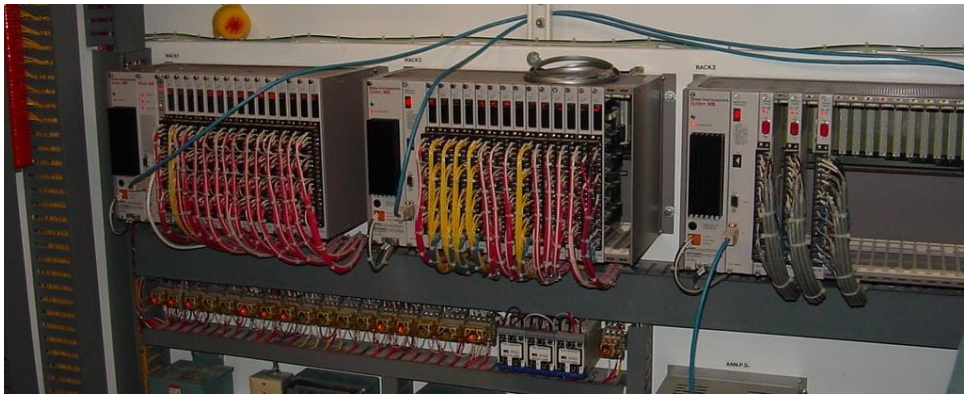
3.2 Types and sizes of PLCs

Large PLC systems consist of a rack into which circuit "cards" are plugged. These cards include processors, input and output (I/O) points, communications ports, and other functions necessary to the operation of a complete PLC system. Such "modular" PLCs may be configured differently according to the specific needs of the application. Individual card failures are also easier to repair in a modular system, since only the failed card need be replaced, not all the cards or the whole card rack.

Small PLC systems consist of a monolithic "brick" containing all processor, I/O, and communication functions. These PLCs are typically far less expensive than their modular cousins, but are also more limited in I/O capability and must be replaced as a whole in the event of failure.

The following photographs show several examples of real PLC systems, some modular and some monolithic. These selections are not comprehensive by any means, as there are many more manufacturers and models of PLC than those I have photographed.

The first photograph is of a Siemens (Texas Instruments) 505 series PLC, installed in a control panel of a municipal wastewater treatment plant. This is an example of a modular PLC, with individual processor, I/O, and communication cards plugged into a rack. Three racks appear in this photograph (two completely filled with cards, and the third only partially filled):

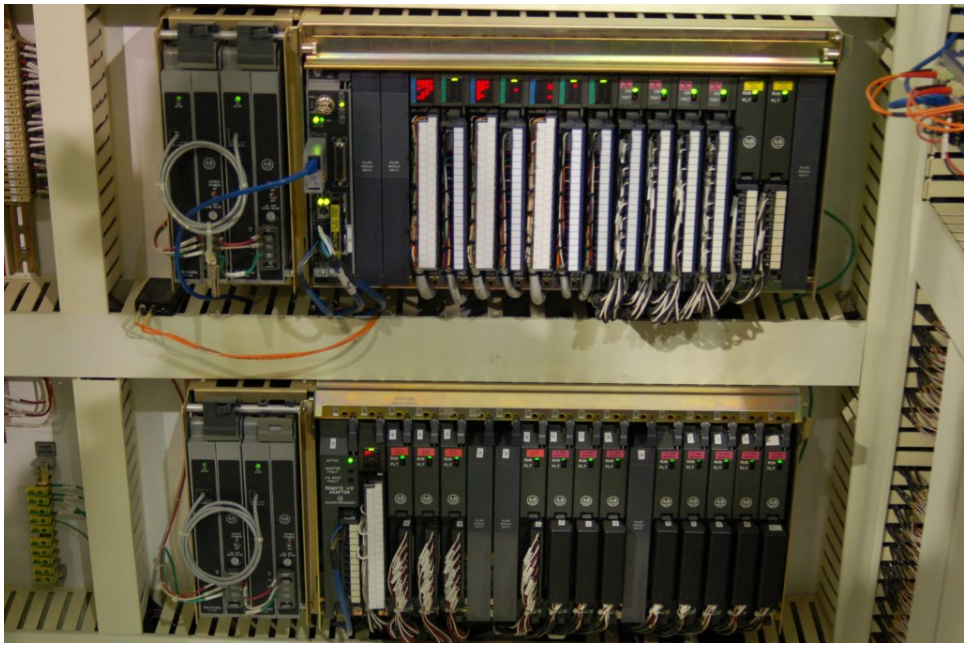


The power supply and processor card for each rack is located on the left-hand end, with I/O cards plugged into slots in the rest of the rack. Input devices such as switches and sensors connect by wire to terminals on *input* cards, while output devices such as lamps, solenoids, and motor contactor coils connect by wire to terminals on *output* cards.

One of the benefits of modular PLC construction is that I/O cards may be changed out as desired, altering the I/O configuration of the PLC as needed. If, for example, the PLC needs to be configured to monitor a greater number of sensors, more input cards may be plugged into the rack and subsequently wired to those sensors. Or, if the *type* of sensor needs to be changed – perhaps from a 24 volt DC sensor to one operating on 120 Volts AC – a different type of input card may be substituted to match the new sensor(s).

In this particular application, the PLC is used to sequence the operation of self-cleaning "trash racks" used to screen large debris such as rags, sticks, and other non-degradable items from municipal wastewater prior to treatment. These trash racks are actuated by electric motors, the captured debris scraped off and transported to a solid waste handling system. The motion of the trash racks, the sensing of wastewater levels and pressures, and the monitoring of any human-operated override controls are all managed by these PLCs. The programming of these PLCs involves timers, counters, sequencers, and other functions to properly manage the continuous operation of the trash racks.

The next photograph shows an Allen-Bradley (Rockwell) PLC-5 system, used to monitor and control the operation of a large natural gas compressor. Two racks appear in this first photograph, with different types of I/O cards plugged into each rack:

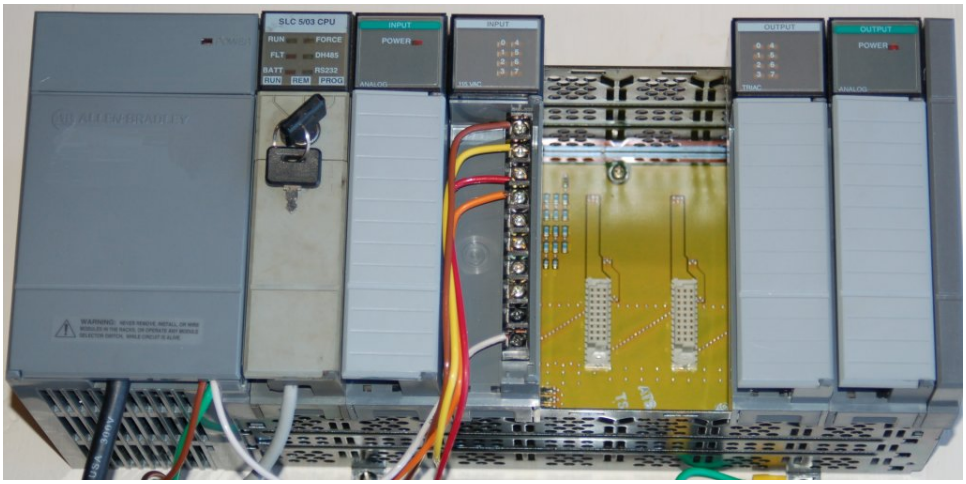


Like the Siemens 505 PLC seen previously, this Allen-Bradley PLC-5 system is fully modular and configurable. The types and locations of the I/O cards inserted into the rack may be altered by appropriately skilled technicians to suit any desired application. The programming of the PLC's processor card may also be altered if a change in the control strategy is desired for any reason.

In this particular application, the PLC is tasked with monitoring certain variables on the gas compressor unit, and taking corrective action if needed to keep the machine productive and safe. The automatic control afforded by the PLC ensures safe and efficient start-ups, shut-downs, and handling of emergency events. The networking and data-logging capability of the PLC ensures that critical data on the compressor unit may be viewed by the appropriate personnel. For this particular compressor station, the data gets communicated from Washington state where the compressor is located all the way to Utah state where the main operations center is located. Human operators in Utah are able to monitor the compressor's operating conditions and issue commands to the compressor over digital networks.

Both the Siemens (formerly Texas Instruments) 505 and Allen-Bradley (Rockwell) PLC-5 systems are considered "legacy" PLC systems by modern standards, the two systems in the previous photographs being over 20 years old each. It is not uncommon to find "obsolete" PLCs still in operation, though. Given their extremely rugged construction and reliable design, these control systems may continue to operate without significant trouble for decades.

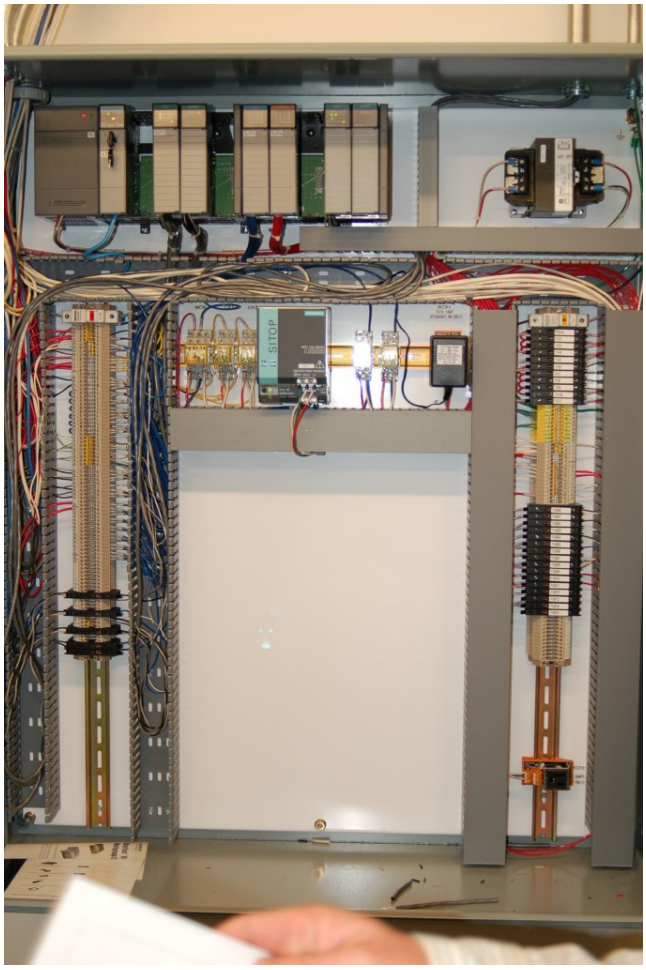
A later model of PLC manufactured by Allen-Bradley was the SLC 500 series (often verbally referred to as the "Slick 500"), also modular in design like the older PLC-5 system, although the racks and modules of the SLC 500 design were more compact than the PLC-5. The SLC 500 rack shown in the next photograph has 7 "slots" for processor and I/O cards to plug in to, numbered 0 through 6 (left to right):



The first three slots of this particular SLC 500 rack (0, 1, and 2) are occupied by the processor card, an analog input card, and a discrete input card, respectively. The slots 3 and 4 are empty (revealing the backplane circuit board and connectors for accepting new cards). The slots 5 and 6 hold discrete output and analog output cards, respectively.

A feature visible on all cards in this system are numerous LED indicators, designed to show the status of each card. The processor card has LED indicators for "Run" mode, "Fault" conditions, "Force" conditions (when either input or output bits have been forced into certain states by the human programmer for testing purposes), and communication network indicators. Each discrete I/O card has indicator LEDs showing the on/off status of each I/O bit, and the analog card has a single LED showing that the card is powered.

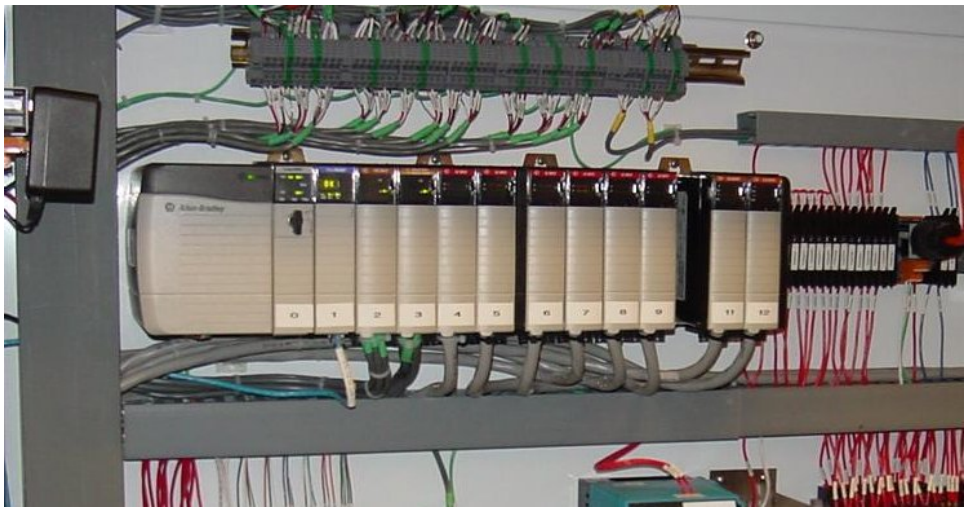
A nine-slot SLC 500 system is shown in the next photograph, controlling a high-purity water treatment system for a biopharmaceuticals manufacturing facility. As you can see in this photograph, not all slots in this particular rack are occupied by I/O cards either:



Some of the inputs to this PLC include water level switches, pressure switches, water flow meters, and conductivity meters (to measure the purity of the water, greater electrical conductivity indicating the presence of more dissolved minerals, which is undesirable in this particular process application). In turn, the PLC controls the starting and stopping of water pumps and the switching of water valves to manage the water purification and storage processes.

A modern PLC manufactured by Siemens appears in this next photograph, an S7-300, which is a different design of modular PLC. Instead of individual cards plugging into a rack, this modular PLC design uses individual modules plugging into each other on their sides to form a wider unit:

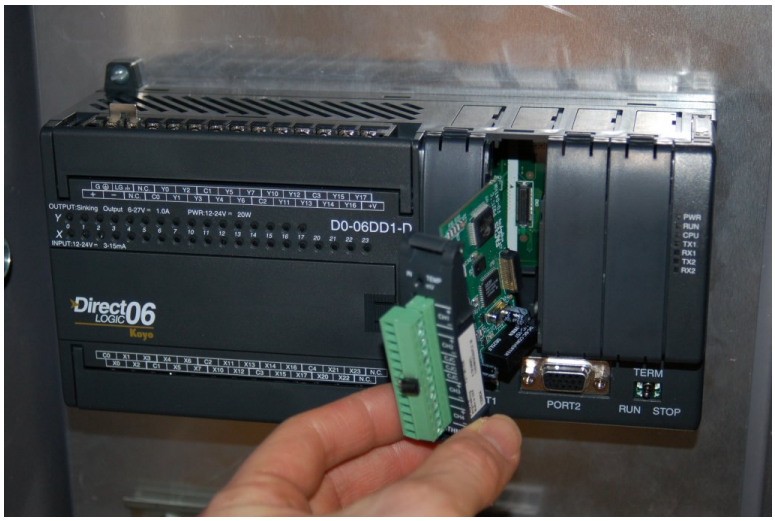
A modern PLC manufactured by Allen-Bradley (Rockwell) is this ControlLogix 5000 system, shown in this photograph used to control a cereal manufacturing process. The modular design of the ControlLogix 5000 system follows the more traditional scheme of individual cards plugged into a rack of fixed size:



While the Siemens S7 and Rockwell ControlLogix PLC platforms represent large-scale, modular PLC systems, there exist much smaller PLCs available for a fraction of the cost. Perhaps the least expensive PLC on the market at this time of writing is the Koyo "CLICK" PLC series, the processor module (with eight discrete input and six discrete output channels built in) shown in my hand (sold for 69 US dollars in the year 2010, and with free programming software!):

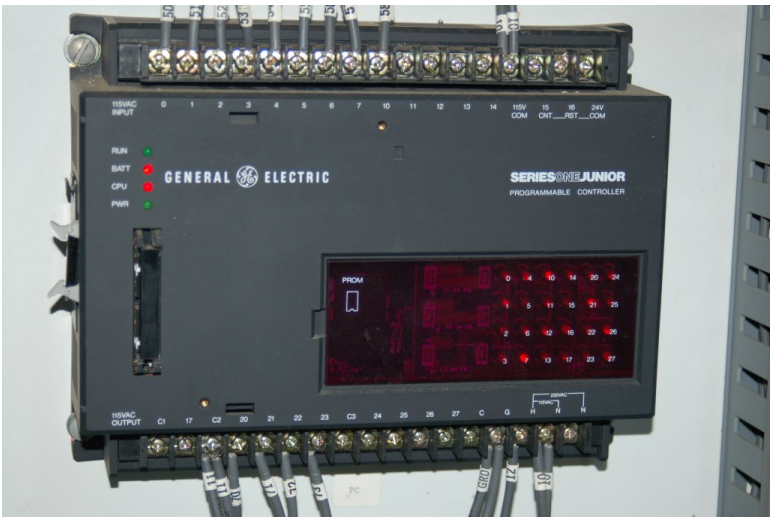
This is a semi-modular PLC design, with a minimum of input/output (I/O) channels built into the processor module, but having the capacity to accept multiple I/O modules plugged in to the side, much like the Siemens S7-300 PLC.

Other semi-modular PLCs expand using I/O cards that plug in to the base unit not unlike traditional rack-based PLC systems. The Koyo DirectLogic DL06 is a good example of this type of semi-modular PLC, the following photograph showing a model DL06 accepting a thermocouple input card in one of its four available card slots:



This photograph shows the PLC base unit with 20 discrete input channels and 16 discrete output channels, accepting an analog input card (this particular card is designed to input signals from thermocouples to measure up to four channels of temperature).

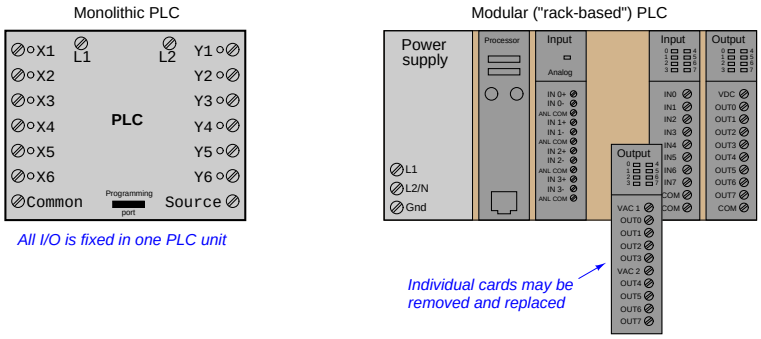
Some low-end PLCs are strictly monolithic, with no ability to accept additional I/O modules. This General Electric Series One PLC (used to monitor a small-scale hydroelectric power generating station) is an example of a purely monolithic design, having no "expansion" slots to accept I/O cards:



A disadvantage of monolithic PLC construction is that damaged I/O cannot be independently replaced. If an I/O channel on one of these PLCs becomes damaged, the entire PLC must be replaced to fix the problem. In a modular system, the damaged I/O card may simply be unplugged from the rack and replaced with a new I/O card. Another disadvantage of monolithic PLCs is the inherently fixed nature of the I/O: the end-user cannot customize the I/O configuration to match the application. For these reasons, monolithic PLCs are usually found on small-scale processes with few I/O channels and limited potential for expansion.

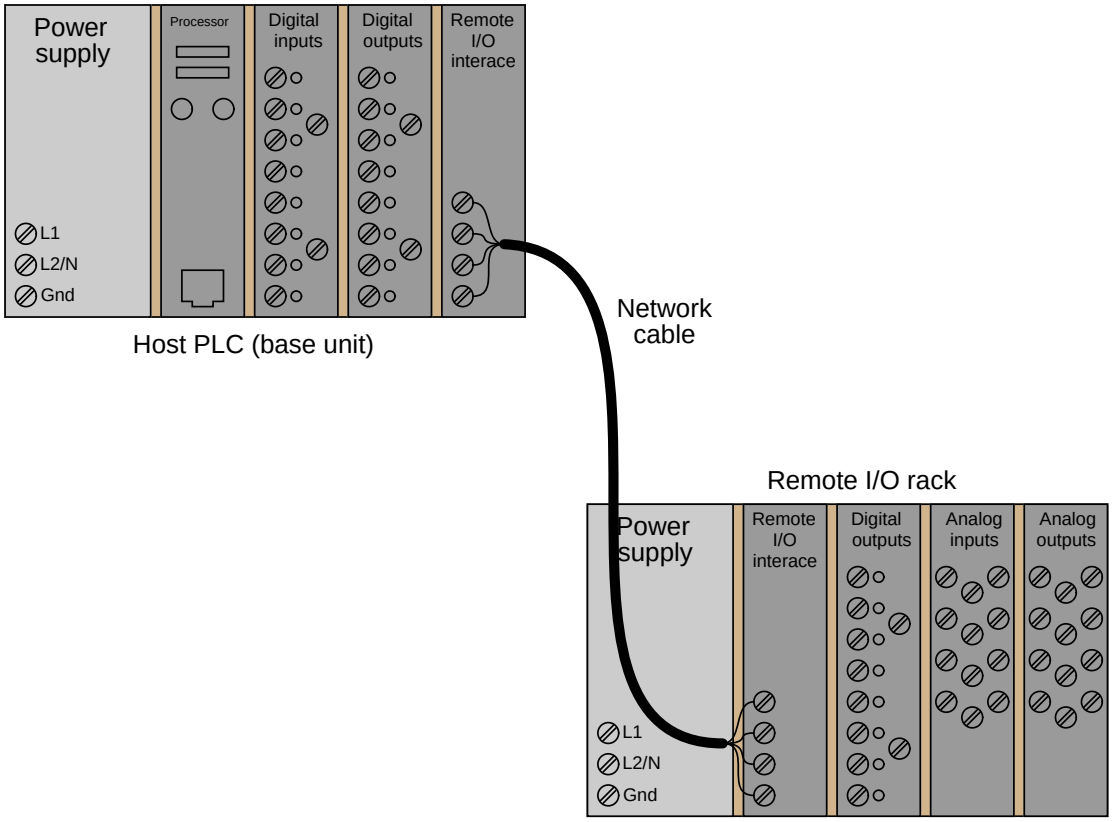
3.3 PLC hardware inputs and outputs (I/O)

Every programmable logic controller must have some means of receiving and interpreting signals from real-world sensors such as switches, and encoders, and also be able to effect control over real-world control elements such as solenoids, valves, and motors. This is generally known as *input/output*, or *I/O*, capability. Monolithic ("brick") PLCs have a fixed amount of I/O capability built into the unit, while modular ("rack") PLCs use individual circuit board "cards" to provide customized I/O capability.



The advantages of using replaceable I/O cards instead of a monolithic PLC design are numerous. First, and most obvious, is the fact that individual I/O cards may be easily replaced in the event of failure without having to replace the entire PLC. Specific I/O cards may be chosen for custom applications, biasing toward discrete cards for applications using many on/off inputs and outputs, or biasing toward analog cards for applications using many 4-20 mA and similar signals. Some PLCs even offer the feature of *hot-swappable* cards, meaning each card may be removed and a new one inserted without de-energizing power to the PLC processor and rack. Please note that one should not assume any system has hot-swappable cards, because if you attempt to change out a card "live" in a system without this feature, you run the risk of damaging the card and/or the rest of the unit it is plugged in to!

Some PLCs have the ability to connect to processor-less remote racks filled with additional I/O cards or modules, thus providing a way to increase the number of I/O channels beyond the capacity of the base unit. The connection from host PLC to remote I/O racks usually takes the form of a special digital network, which may span a great physical distance:

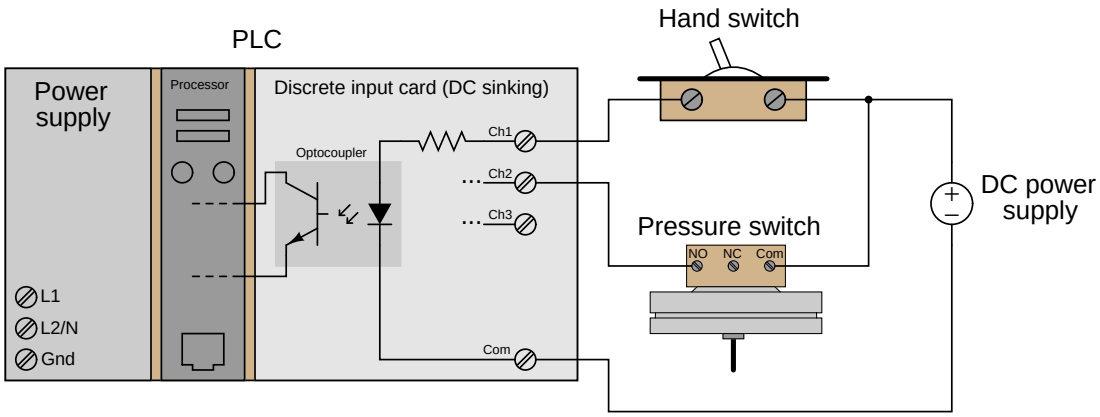


An alternative scheme for system expansion is to network multiple PLCs together, where each PLC has its own dedicated rack and processor. Through the use of communication instructions, one PLC may be programmed to read data from and/or write data to another PLC, effectively using the other PLC as an extension of its own I/O. Although this method is more expensive than remote I/O (where the remote racks lack their own dedicated processors), it provides the capability of stand-alone control in the event the network connection between PLC processors becomes severed.

Input/output capability for programmable logic controllers comes in three basic varieties: *discrete*, *analog*, and *network*; each type discussed in a following section.

3.4 Discrete PLC I/O

A "discrete" data point is one with only two states *on* and *off*. Process switches, pushbutton switches, limit switches, and proximity switches are all examples of discrete sensing devices. In order for a PLC to be aware of a discrete sensor's state, it must receive a signal from the sensor through a *discrete input* channel. Inside each discrete input module is (typically) a set of light-emitting diodes (LEDs) which will be energized when the corresponding sensing device turns on. Light from each LED shines on a photo-sensitive device such as a phototransistor inside the module, which in turn activates a *bit* (a single element of digital data) inside the PLC's memory. This opto-coupled arrangement makes each input channel of a PLC rather rugged, capable of isolating the sensitive computer circuitry of the PLC from transient voltage "spikes" and other electrical phenomena capable of causing damage:

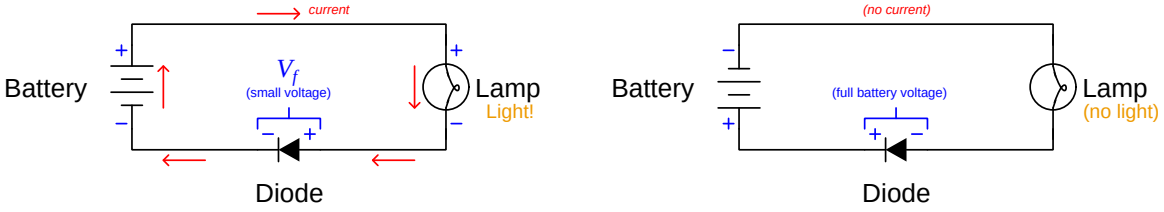


Energizing an input channel lights the LED inside the optocoupler, turning on the phototransistor, sending a "high" signal to the PLC's microprocessor, setting (1) that bit in the PLC's input register.

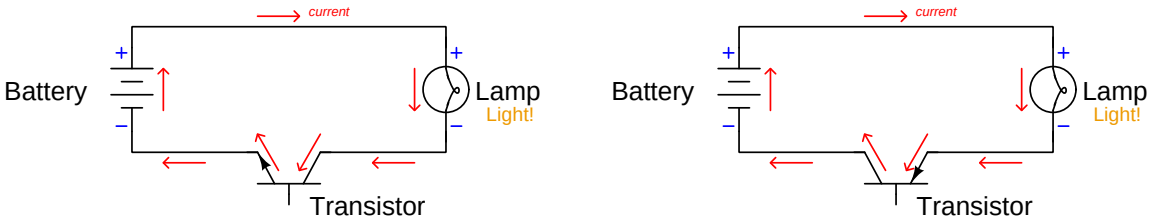
The internal schematic diagram for a discrete input module ("card") shown above reveals the componentry typical for a single input channel on that card. Each input channel has its own optocoupler, writing to its own unique memory register bit inside the PLC's memory. Discrete input cards for PLCs typically have 4, 8, 16, or 32 channels.

Note the "arrow-head" portions of the component symbols shown within the PLC module. These arrow-heads always point in the direction of *conventional flow notation*. This is the convention electrical engineers use to show direct-current electricity flow, that direction exiting the positive (+) terminal of any electrical source and entering the positive terminal of any electrical load. When PLC I/O modules contain such directional components, the direction of current really matters. Wiring such an I/O module incorrectly such that electric current will attempt to pass through it in the wrong direction will result in the module not working at all.

Below we see a simple circuit diagram showing a DC voltage source (battery), an electric lamp, and a diode with current direction sketched according to conventional flow notation. In the left-hand version of this circuit current flows freely through the lamp to energize it, but in the right-hand version the battery is connected backwards and the diode refuses to let current pass:

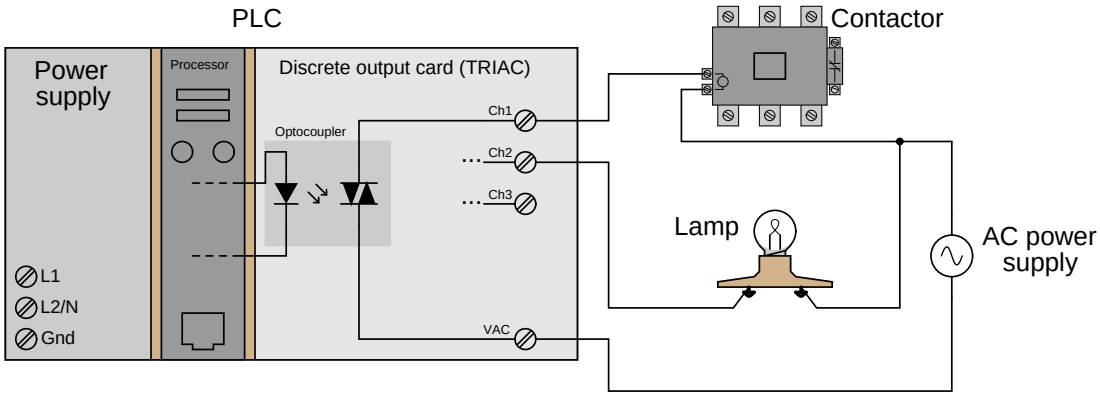


Bipolar junction transistors or *BJTs* are also semiconductor devices like diodes in that they only function with one direction of current. Below we see two circuits, each one controlling a lamp using a transistor as an on/off switch, the transistors assumed[4] to be on in each case:



[4]To properly turn on a bipolar junction transistor we must either shine light onto its semiconductor PNP or NPN junctions, or activate it electrically by making appropriate connections to its "base" terminal. These details are beyond the scope of our discussion on PLC I/O modules.

Indicator lamps, solenoid valves, and motor starters (assemblies consisting of contactors and overload protection devices) are all examples of discrete control devices. In a manner similar to discrete inputs, a PLC connects to any number of different discrete final control devices through a *discrete output channel*[5]. Discrete output modules typically use the same form of opto-isolation to allow the PLC's computer circuitry to send electrical power to loads: the internal PLC circuitry driving an LED which then activates some form of photosensitive switching device.



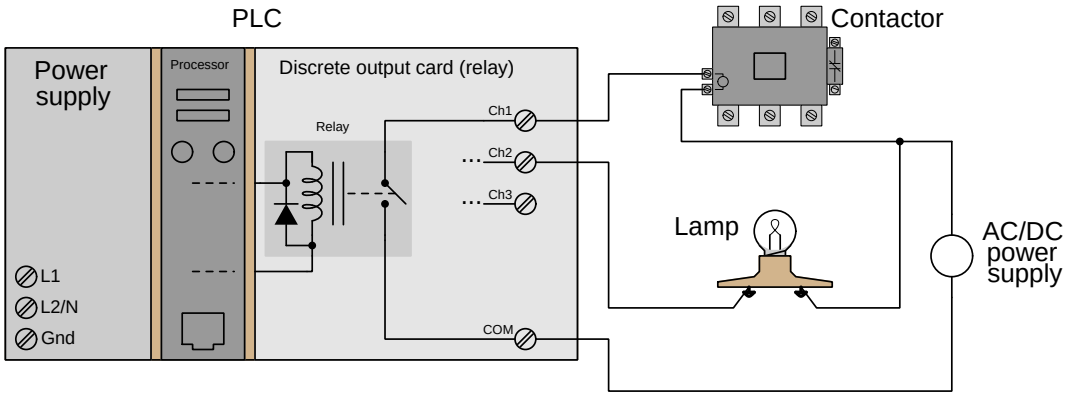
Setting a bit (1) in the PLC's output register sends a "high" signal to the LED inside the optocoupler, turning on the photo-TRIAC, sending AC power to the output channel to energize the load.

As with the schematic diagram for a discrete input module shown previously, the schematic diagram shown here for a discrete output module reveals the componentry typical for a single channel on that card. Each output channel has its own optocoupler, driven by its own unique memory register bit inside the PLC's memory. Discrete output cards for PLCs also typically have 4, 8, 16, or 32 channels.

The current switching device in this particular output module is called a *TRIAC*, and it functions much like a transistor for AC: able to switch on and off but for current passing in either direction. The dual arrow-heads pointing in opposite directions suggest the TRIAC's AC-switching capability.

[5] I/O "channels" are often referred to as "points" in industry lingo. Thus, a "32-point input card" refers to an input circuit with 32 separate channels through which to receive signals from on/off switches and sensors.

A common alternative to opto-isolated semiconductor switching elements such as transistors (DC) or TRIACs (AC) are miniature electromechanical relays. Mechanical relay contacts are capable of switching DC or AC, and provide similar levels of electrical isolation between the PLC's internal logic circuitry and external control circuits compared with opto-isolated transistors and TRIACs:



Setting a bit (1) in the PLC's output register sends a "high" signal to the relay coil, closing the relay contact, completing the circuit for power to energize the load.

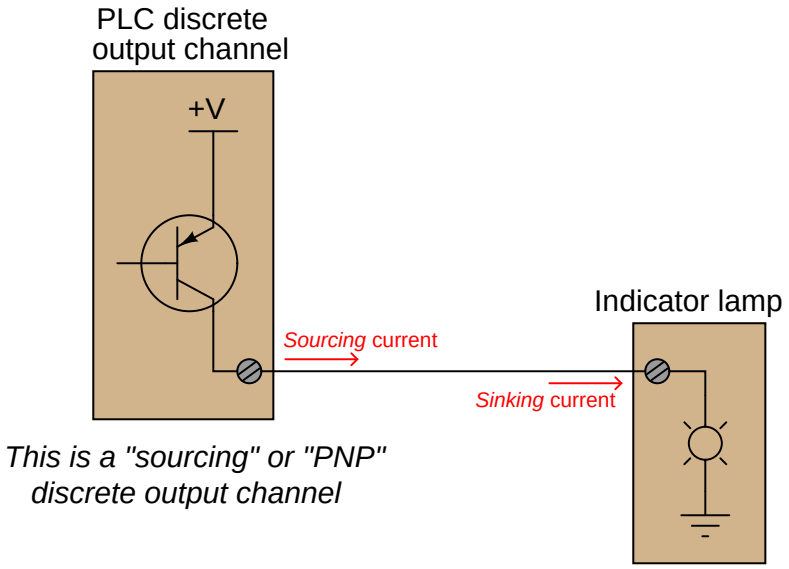
The relay-based discrete PLC output shown here happens to share a "Common" terminal with the other discrete output channels of the PLC. Isolated relay outputs, however, are a popular option. In an isolated relay output array, each relay contact has its own pair of dedicated screw terminals. This is useful when multiple loads requiring separate power supplies must be controlled by the same PLC.

Just like transistor and TRIAC discrete outputs, relay outputs typically allow the end-user to connect their power supply of choice to the switching element. Given that relay contacts are capable of DC and AC current switching, this provides a greater level of versatility than semiconductor elements such as transistors and TRIACs. Relays, of course, have their disadvantages as well. Relay contacts are much slower-responding than semiconductors which have no moving parts. The moving components of relays are subject to wear and failure in ways that semiconductor devices cannot fail. Relay contacts also "bounce" as they move, which may be problematic if the PLC output needs to drive the input of a high-speed counting device because each "bounce" will count as a separate contact closure event.

A term you will frequently encounter regarding PLC output relay contacts is *dry contact*. This simply means the relay contact inside the PLC's output channel has no pre-connected power source, and that the end-user must wire in their own (external) power source in order for the contact to send electrical power to any load. Most semiconductor-based discrete I/O channels are the same way, but interestingly the term "dry" is rarely used to describe semiconductor switching elements, only electromechanical relay contacts. By contrast, a *wetted contact* is internally connected to an electrical source, with terminals provided to directly connect to a load with no external power supply required.

An important concept to master when working with DC discrete I/O is the distinction between *current-sourcing* and *current-sinking* devices. The terms "sourcing" and "sinking" refer to the direction of current (as denoted by conventional flow notation) into or out of a device's control wire[6]. A device sending (conventional flow) current out of its control terminal to some other device(s) is said to be *sourcing* current, while a device accepting (conventional flow) current into its control terminal is said to be *sinking* current.

To illustrate, the following illustration shows a PLC output channel is *sourcing* current to an indicator lamp, which is *sinking* current to ground:

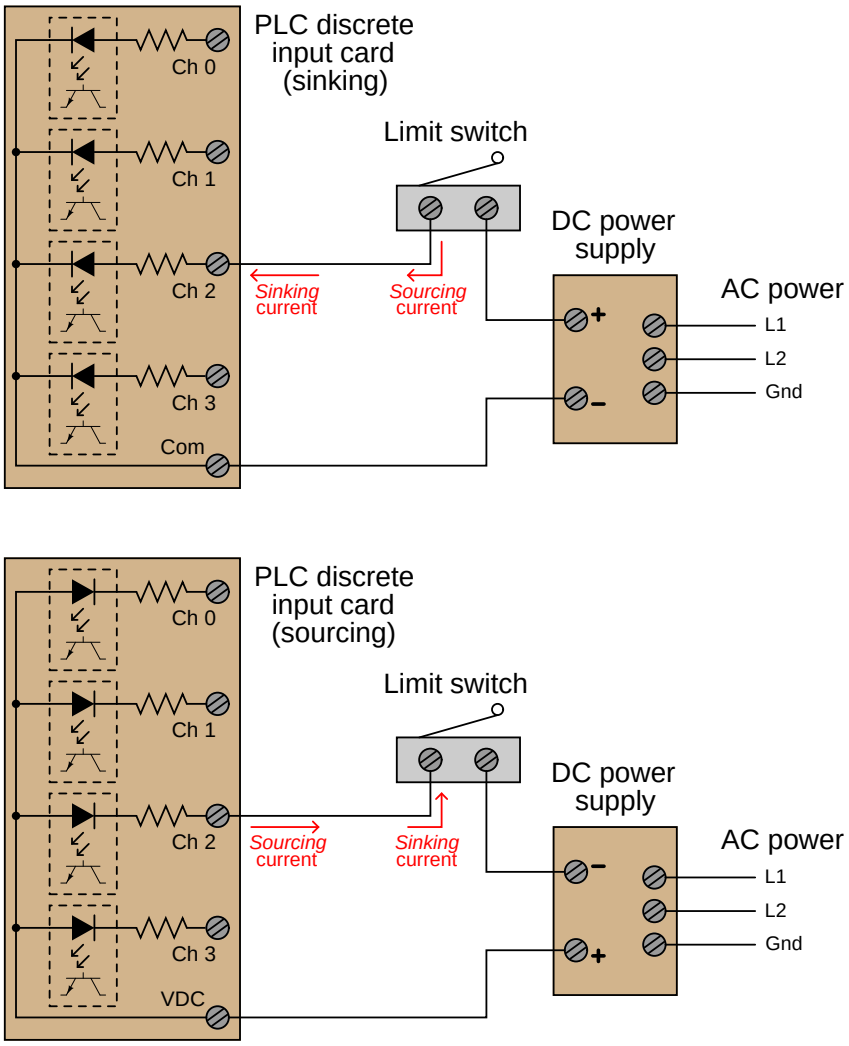


These terms really only make sense when electric current is viewed from the perspective of conventional flow, where the positive terminal of the DC power supply is envisioned to be the "source" of the current, with current finding its way "down" to ground (the negative terminal of the DC power supply). In every circuit formed by the output channel of a PLC driving a discrete control device, or by a discrete sensing device driving an input channel on a PLC, one element in the circuit must be sourcing current while the other is sinking current.

An engineering colleague of mine has a charming way to describe sourcing and sinking: *blowing* and *sucking*. A device that sources current to another "blows" current toward the other device. A device that sinks current "sucks" current from the other device. Many students seem to find these terms helpful in first mastering the distinction between sourcing and sinking despite (or perhaps because of!) their informal nature.

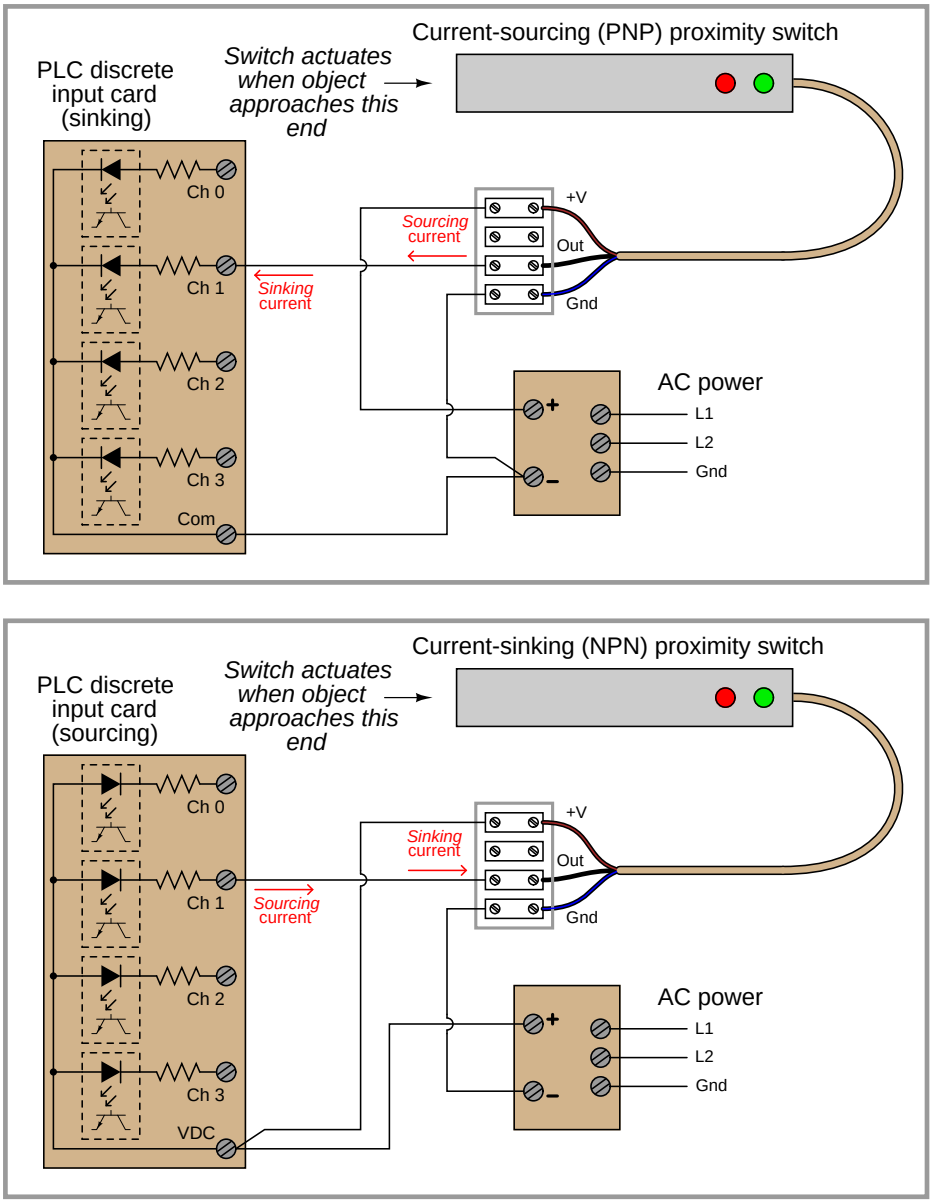
[6] By "control wire," I mean the single conductor connecting the I/O card channel to the field device, as opposed to conductors directly common with either the positive or negative lead of the voltage source. If you focus your attention on this one wire, noting the direction of conventional-flow current through it, the task of determining whether a device is sourcing or sinking current becomes much simpler.

If the discrete device connecting to the PLC is not polarity-sensitive, either type of PLC I/O module will suffice. For example, the following diagrams show a mechanical limit switch connecting to a sinking PLC input and to a sourcing PLC input:



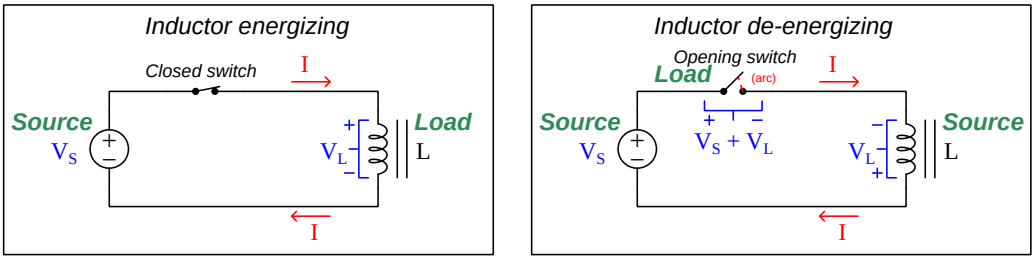
Note the differences in polarity and labeling between the sinking card's common terminal and the sourcing card's common terminal. On the "sinking" card, the input channel terminal is positive while the common ("Com") terminal is negative. On the "sourcing" card, the input channel terminal is negative while the common ("VDC") terminal is positive.

Some discrete sensing devices *are* polarity-sensitive, such as electronic proximity sensors containing transistor outputs. A "sourcing" proximity switch can only interface with a "sinking" PLC input channel, and vice-versa:



In all cases, the "sourcing" device sends current *out of* its signal terminal while the "sinking" device takes current *into* its signal terminal.

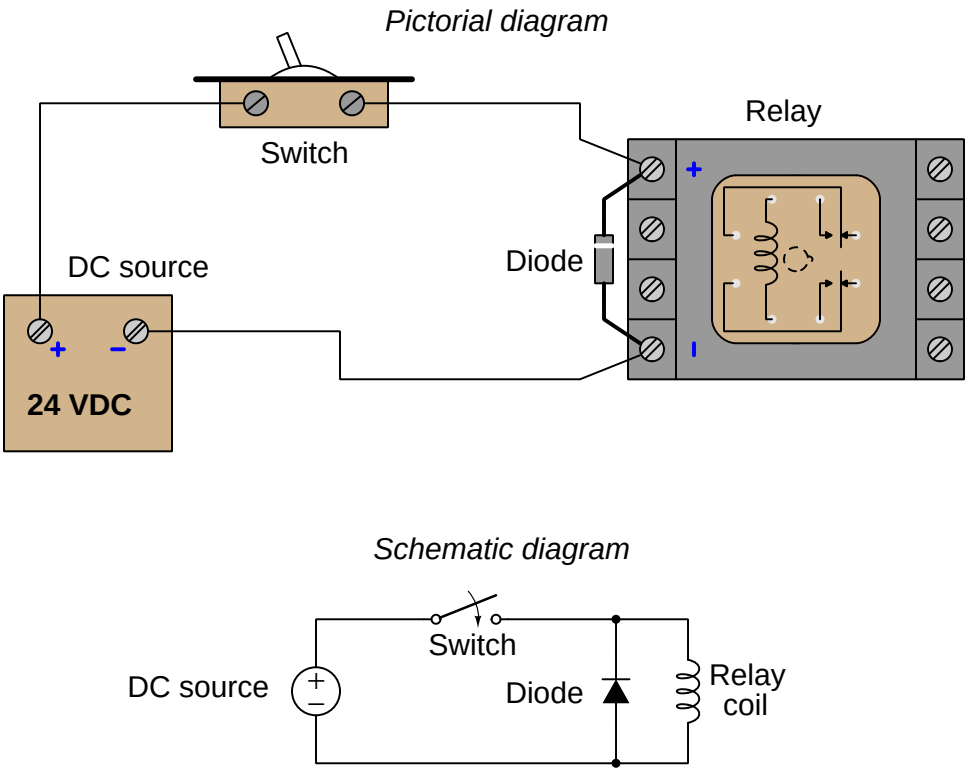
A concern when connecting inductive DC loads to either sinking or sourcing PLC outputs is a phenomenon called *inductive kickback* where the inductive device generates high voltage upon de-energization. Inductance by definition is any electrical structure capable of storing energy in a magnetic field, that magnetic field a result of current passing through the device. Relay coils, solenoid valve coils, electric motors, and all other electromagnetic devices contain inductance. When something external to such a device attempts to break the circuit to de-energize that device, the magnetic field accumulated within that device will begin to decay in strength, and as it does so the device will begin to output energy (i.e. act as an electrical *source*), its DC voltage polarity suddenly reversing while current continues to pass through it in the same direction as when it was functioning as a load. We see this phenomenon illustrated below:



Note how when the switch opens, it functions as an electrical load with *both* the DC source (V_S) and the de-energizing inductance (L) acting as electrical sources contributing energy to the circuit. Normally we don't think of switches as loads, but in this case the high voltage dropped across the air gap of the opening switch causes the air to ionize and conduct electricity, the resulting arc dissipating energy in the form of heat with intensely-high temperatures damaging the metal switch contact points over time.

Mechanical switches can bear this abuse for many on/off cycles before completely failing, but semiconductor switching components such as transistors cannot. If the switching element for an inductive circuit is a transistor such as what we find within a PLC's DC output channel, the high voltage created by the collapsing magnetic field (V_L) *will immediately damage that transistor* unless mitigated somehow. A popular way to prevent high-voltage impulses at the inductive device is to take advantage of the voltage polarity reversal by connecting a diode across the device's terminals such that current will pass through it rather than through the opening switch, and by having a low-resistance pathway that current will create only modest voltage drops rather than destructively high voltage drops.

A demonstration circuit below shows a 24 Volt DC source energizing a relay through a toggle switch, with a diode connected across the relay's coil terminals to provide that non-destructive pathway for current when the switch opens (turns off) and the coil must release its stored magnetic energy:

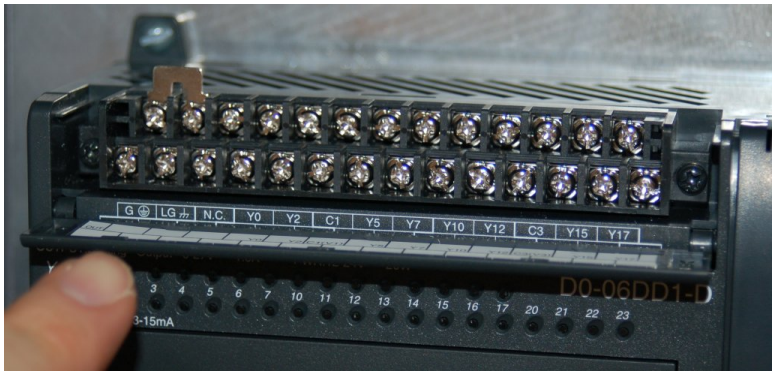


This is called a *commutating diode* because it serves its purpose when the inductance's voltage polarity reverses (commutates), but no current passes through the diode when the inductive load is being energized by the "on" toggle switch. The polarity shown in the illustration by the blue-colored + and − symbols drawn on the relay's base represents the polarity while the relay is being energized and is acting as a load (i.e. when the diode is non-conducting).

Two photographs of a DC (sinking) discrete input module for an Allen-Bradley model SLC 500 PLC are shown here: one with the plastic cover closed over the connection terminals, and the other with the plastic cover opened up for viewing the terminals. A legend on the inside of the cover shows the purpose of each screw terminal: eight input channels (numbered 0 through 7) and two redundant "DC Com" terminals for the negative pole of the DC power supply to connect:

A standard feature found on practically every PLC discrete I/O module is a set of LED indicators visually indicating the status of each bit (discrete channel). On the SLC 500 module, the LEDs appear as a cluster of eight numbered squares near the top of the module.

A photograph of discrete output terminals on another brand of PLC (a Koyo model DL06) shows somewhat different labeling:



Here, each output channel terminal is designated with a letter/number code beginning with the letter "Y". Several "common" terminals labeled with "C" codes service clusters of output channels. In this particular case, each "common" terminal is common only to four output channels. With sixteen total output channels on this PLC, this means there are four different "common" terminals. While this may seem somewhat strange (why not just have one "common" terminal for all sixteen

output channels?), it more readily permits different DC power supplies to service different sets of output channels.

Electrical polarity is not an issue with AC discrete I/O, since the polarity of AC reverses periodically anyway. However, there is still the matter of whether the "common" terminal on a discrete PLC module will connect to the *neutral* (grounded) or *hot* (ungrounded) AC power conductor.

The next photograph shows a discrete AC output module for an Allen-Bradley SLC 500 PLC, using TRIACs as power switching devices rather than transistors as is customary with DC discrete output modules:

This particular eight-channel module provides two sets of TRIACs for switching power to AC loads, each set of four TRIACs receiving AC power from a "hot" terminal (VAC 1 or VAC 2), the other side of the load device being connected to the "neutral" (grounded) conductor of the AC power source.

Fortunately, the hardware reference manual supplied by the manufacturer of every PLC shows diagrams illustrating how to connect discrete input and output channels to field devices. One should always consult these diagrams before connecting devices to the I/O points of a PLC!

3.5 Analog PLC I/O

In the early days of programmable logic controllers, processor speed and memory were too limited to support anything but discrete (on/off) control functions. Consequently, the only I/O capability found on early PLCs were discrete in nature[7]. Modern PLC technology, though, is powerful enough to support the measurement, processing, and output of analog (continuously variable) signals.

All PLCs are digital devices at heart. Thus, in order to interface with an analog sensor or control device, some "translation" is necessary between the analog and digital worlds. Inside every analog input module is an *ADC*, or *Analog-to-Digital Converter*, circuit designed to convert an analog electrical signal into a multi-bit binary word. Conversely, every analog output module contains a *DAC*, or *Digital-to-Analog Converter*, circuit to convert the PLC's digital command words into analog electrical quantities.

Analog I/O is commonly available for modular PLCs for many different analog signal types, including:

- Voltage (0 to 10 Volt, 0 to 5 Volt)
- Current (0 to 20 mA, 4 to 20 mA)
- Thermocouple (millivoltage)
- RTD (millivoltage)
- Strain gauge (millivoltage)

[7] Some modern PLCs such as the Koyo "CLICK" are also discrete-only. Analog I/O and processing is significantly more complex to engineer and more expensive to manufacture than discrete control, and so low-end PLCs are more likely to lack analog capability.

The following photographs show two analog I/O cards for an Allen-Bradley SLC 500 modular PLC system, an analog input card and an analog output card. Labels on the terminal cover doors indicate screw terminal assignments:

3.6 Network PLC I/O

Many different digital network standards exist for PLCs to communicate with, from PLC to PLC and between PLCs and field devices. One of the earliest digital protocols developed for PLC communication was *Modbus*, originally for the Modicon brand of PLC. Modbus was adopted by other PLC and industrial device manufacturers as a *de facto* standard[8], and remains perhaps the most universal digital protocol available for industrial digital devices today.

Another digital network standard developed by a particular manufacturer and later adopted as a *de facto* standard is *Profibus*, originally developed by Siemens.

[8] A "de facto" standard is one arising naturally out of legacy rather than by an premeditated agreement between parties. Modbus and Profibus networks are considered "de facto" standards because those networks were designed, built, and marketed by pioneering firms prior to their acceptance as standards for others to conform to. In Latin, *de facto* means "from the fact," which in this case refers to the fact of pre-existence: a standard agreed upon to conform to something already in popular use. By contrast, a standard intentionally agreed upon before its physical realization is a *de jure* standard (Latin for "from the law"). FOUNDATION Fieldbus is an example of a *de jure* standard, where a committee arrives at a consensus for a network design and specifications prior to that network being built and marketed by any firm.

Chapter 4

Derivations and Technical References

This chapter is where you will find mathematical derivations too detailed to include in the tutorial, and/or tables and other technical reference material.

4.1 Feature comparisons between PLC models

In most cases, similarities are far greater for different models of PLC than differences. However, differences do exist, and it is worth exploring the differences in basic features offered by an array of PLC models.

4.1.1 Viewing live values

- Allen-Bradley Logix 5000: the *Controller Tags* folder (typically on the left-hand pane of the programming window set) lists all the tag names defined for the PLC project, allowing you to view the real-time status of them all. Discrete inputs do not have specific input channel tag names, as tag names are user-defined in the Logix5000 PLC series.

- Allen-Bradley PLC-5, SLC 500, and MicroLogix: the *Data Files* listing (typically on the left-hand pane of the programming window set) lists all the data files within that PLC's memory. Opening a data file window allows you to view the real-time status of these data points. Discrete inputs are the `I` file addresses (e.g. `I:0/2`, `I:3/5`, etc.). The letter "`I`" represents "input," the first number represents the slot in which the input card is plugged, and the last number represents the bit within that data element (a 16-bit word) corresponding to the input card.

- Siemens S7-200: the *Status Chart* window allows the user to custom-configure a table showing the real-time values of multiple addresses within the PLC's memory. Discrete inputs are the `I` memory addresses (e.g. `I0.1`, `I1.5`, etc.).

- Koyo (Automation Direct) DirectLogic and CLICK: the *Data View* window allows the user to custom-configure a table showing the real-time values of multiple addresses within the PLC's memory. Discrete inputs are the `X` memory addresses (e.g. `X1`, `X2`, etc.).

4.1.2 Forcing live values

- Allen-Bradley Logix 5000: forces may be applied to specific tag names by right-clicking on the tag (in the program listing) and selecting the "Monitor" option. Discrete outputs do not have specific output channel tag names, as tag names are user-defined in the Logix5000 PLC series.
- Allen-Bradley PLC-5, SLC 500, and MicroLogix: the *Force Files* listing (typically on the left-hand pane of the programming window set) lists those data files within the PLC's memory liable to forcing by the user. Opening a force file window allows you to view and set the real-time status of these bits. Discrete outputs are the `O` file addresses (e.g. `O:0/7`, `O:6/2`, etc.). The letter "`O`" represents "output," the first number represents the slot in which the output card is plugged, and the last number represents the bit within that data element (a 16-bit word) corresponding to the output card.
- Siemens S7-200: the *Status Chart* window allows the user to custom-configure a table showing the real-time values of multiple addresses within the PLC's memory, and enabling the user to force the values of those addresses. Discrete outputs are the `Q` memory addresses (e.g. `Q0.4`, `Q1.2`, etc.).
- Koyo (Automation Direct) DirectLogic and CLICK: the *Override View* window allows the user to force variables within the PLC's memory. Discrete outputs are the `Y` memory addresses (e.g. `Y1`, `Y2`, etc.).

4.1.3 Single-bit addressing

- Allen-Bradley Logix 5000: a period serves as the bit delimiter. For example, `ThisRegister.0` is the least-significant bit (bit #0) of the multi-bit word `ThisRegister`.
- Allen-Bradley PLC-5, SLC 500, and MicroLogix: a forward-slash symbol serves as the bit delimiter. For example, `N7:1/15` is the most-significant bit (bit #15) the 16-bit word `N7:1`.
- Siemens S7-200: a period serves as the bit delimiter. For example, `SM0.5` is the #5 bit of the word `SM0`.

4.1.4 Special "system" values

Every PLC has special registers holding data relevant to its operation, such as error flags, processor scan time, etc.

- Allen-Bradley Logix 5000: various "system" values are accessed via `GSV` (Get System Value) and `SSV` (Save System Value) instructions.
- Allen-Bradley PLC-5, SLC 500, and MicroLogix: the `Data Files` listing (typically on the left-hand pane of the programming window set) shows file number 2 as the "Status" file, in which you will find various system-related bits and registers.
- Siemens S7-200: the *Special Memory* registers contain various system-related bits and registers. These are the `SM` memory addresses (e.g. `SM0.1`, `SMB8`, `SMW22`, etc.).
- Koyo (Automation Direct) DirectLogic and CLICK: the *Special* registers contain various system-related bits and registers. These are the `SP` memory addresses (e.g. `SP1`, `SP2`, `SP3`, etc.) in the DirectLogic PLC series, and the `SC` and `SD` memory addresses in the CLICK PLC series.

4.1.5 Free-running clock pulses

- Allen-Bradley SLC 500: status bit `S:4/0` is a free-running clock pulse with a period of 20 milliseconds, which may be used to clock a counter instruction up to 50 to make a 1-second pulse (because 50 times 20 ms = 1000 ms = 1 second).
- Siemens S7-200: Special Memory bit `SM0.5` is a free-running clock pulse with a period of 1 second.
- Koyo (Automation Direct) DirectLogic: Special relay `SP4` is a free-running clock pulse with a period of 1 second.

4.1.6 Standard counter instructions

- Allen-Bradley Logix 5000: `CTU` count-up, `CTD` count-down, and `CTUD` count-up/down instructions. A `RES` reset instruction exists to reset the accumulator value of the specified counter.
- Allen-Bradley SLC 500: `CTU` and `CTD` instructions. A `RES` reset instruction exists to reset the accumulator value of the specified counter.
- Siemens S7-200: `CTU` count-up, `CTD` count-down, and `CTUD` count-up/down instructions.
- Koyo (Automation Direct) DirectLogic: `UDC` counter instruction.

4.1.7 High-speed counter instructions

- Allen-Bradley SLC 500: `HSU` high-speed count-up instruction.
- Siemens S7-200: `HSC` high-speed counter instruction, used in conjunction with the `HDEF` high-speed counter definition instruction.

4.1.8 Timer instructions

- Allen-Bradley Logix 5000: `TOF` off-delay timer, `TON` on-delay timer, `RTO` retentive on-delay timer, `TOFR` off-delay timer with reset, `TONR` on-delay timer with reset, and `RTOR` retentive on-delay timer with reset instructions.
- Allen-Bradley SLC 500: `TOF` off-delay timer, `TON` on-delay timer, and `RTO` retentive on-delay timer instructions.
- Siemens S7-200: `TOF` off-delay timer, `TON` on-delay timer, and `TONR` retentive on-delay timer instructions.

4.1.9 ASCII text message instructions

- Allen-Bradley Logix 5000: the "ASCII Write" instructions `AWT` and `AWA` may be used to do this. The "ASCII Write Append" instruction (`AWA`) is convenient to use because it may be programmed to automatically insert linefeed and carriage-return commands at the end of a message string.
- Allen-Bradley SLC 500: the "ASCII Write" instructions `AWT` and `AWA` may be used to do this. The "ASCII Write Append" instruction (`AWA`) is convenient to use because it may be programmed to automatically insert linefeed and carriage-return commands at the end of a message string.
- Siemens S7-200: the "Transmit" instruction (`XMT`) is useful for this task when used in Freeport mode.
- Koyo (Automation Direct) DirectLogic: the "Print Message" instruction (`PRINT`) is useful for this task.

4.1.10 Analog signal scaling

- Allen-Bradley Logix 5000: the I/O configuration menu (specifically, the *Module Properties* window) allows you to directly and easily scale analog input signal ranges into any arbitrary numerical range desired. Floating-point ("REAL") format is standard, but integer format may be chosen for faster processing of the analog signal.

- Allen-Bradley PLC-5, SLC 500, and MicroLogix: raw analog input values are 16-bit signed integers. The `SCL` and `SCP` instructions are custom-made for scaling these raw integer ADC count values into ranges of your choosing.

- Siemens S7-200: raw analog input values are 16-bit signed integers. Interestingly, the S7-200 PLC provides built-in potentiometers assigned to special word registers (`SMB28` and `SMB29`) with an 8-bit (0-255 count) range. These values may be used for any suitable purpose, including combination with the raw analog input register values in order to provide mechanical calibration adjustments for the analog input(s).

- Koyo (Automation Direct) DirectLogic: you must use standard math instructions (e.g. `ADD`, `MUL`) to implement a $y = mx + b$ linear equation for scaling purposes.

- Koyo (Automation Direct) CLICK: the I/O configuration menu allows you to directly and easily scale analog input signal ranges into any arbitrary numerical range desired.

4.2 Getting started with Rockwell Studio 5000

These instructions were developed using version 38.01.00 of Rockwell Automation's Studio 5000 Logix Designer software, communicating with a model 1756-L902TS ControlLogix PLC processor module.

Starting a new project online:

1. Open *Logix Designer* software.
2. Click on `File` → `New` to create a new project.
3. Click on `Communications` → `Network Browser`.
4. If you are connecting to the PLC processor using USB, expand on that item in the list; otherwise, choose the network connection you prefer.
5. Drill down on the processor icon (for ControlLogix this will be the one beginning with the 1756 part number).
6. Select `Backplane` and then click on the `Set Project Path` button.
7. Click on the `Go Online` button to begin working while connected to the PLC processor.
8. A `Connected to Go Online` window will appear, in which you will click on the `Download` button.
9. Looking in the `Controller Organizer` list on the left-hand side of the software window, drill down into `Tasks` → `MainTask` → `MainProgram` to raise the ladder-diagram programming window.
10. Right-click on `I/O Configuration` → `1756 Backplane` and choose `Discover Modules`.
11. Under the `Action` column, click on the `Create` button for each I/O module you wish to include in this project.
12. A `Device Definition` window will pop up for each module as you "create" it, where you confirm proper rack slot number and Revision number; here you may also write custom names and descriptions for each module if you prefer.
13. When prompted with a `Create new module online` dialogue box, choose `Create and Uninhibit` if you wish that I/O module to be active. Note that the default is `Create and Inhibit` which will render the module unworkable for your project!
14. Note that some modules will open up additional dialogue windows.
15. Go to the `MainProgram -- MainRoutine` window and start building your ladder-diagram program for the PLC to run, dragging and dropping new instructions into rungs.

16. To create tags as you write your program, right-click on the tagname placeholder (`?`) appearing at the top of each new placed instruction. If the new tagname must map to a real I/O point on the controller, be sure to choose `Alias` rather than `Base`, and then under `Alias For` you may choose the specific I/O point. Note that for basic inputs and outputs you will find the bits and words under "Data". For example, `Local:1:I.Data.0` represents channel 0 on a discrete input module plugged into slot 1 of the PLC chassis, and `Local:3:O.Data.5` represents channel 5 of a discrete output module plugged into slot 3.

17. When online-editing, all new lines and edits will show themselves as new on the program editing view by a dashed vertical line over the left-hand rail. Right-clicking on the rung number of that new/edited rung will allow you to `Accept Pending Rung Edits`, or there is also a green check-mark icon at the top of the editing window to do the same. This compiles the ladder code and displays any syntax errors found.

18. An icon with a fat right-pointing green arrow called `Finalize All Edits in Program` (hot-key sequence *Ctrl + Shift + F*) updates these edits in the running PLC processor. This is analogous to downloading a program to the PLC processor when the PLC is already in Run mode.

Offline programming differences:

1. After making any edits to the ladder-diagram program while offline, right-click on the rung number and select `Verify Rung`. This, like `Accept Pending Rung Edits` for online editing, compiles your code and lets you know if any syntax errors exist.

2. Click on `Communications` → `Download` to send this code to the PLC.

Rockwell PLC tags:

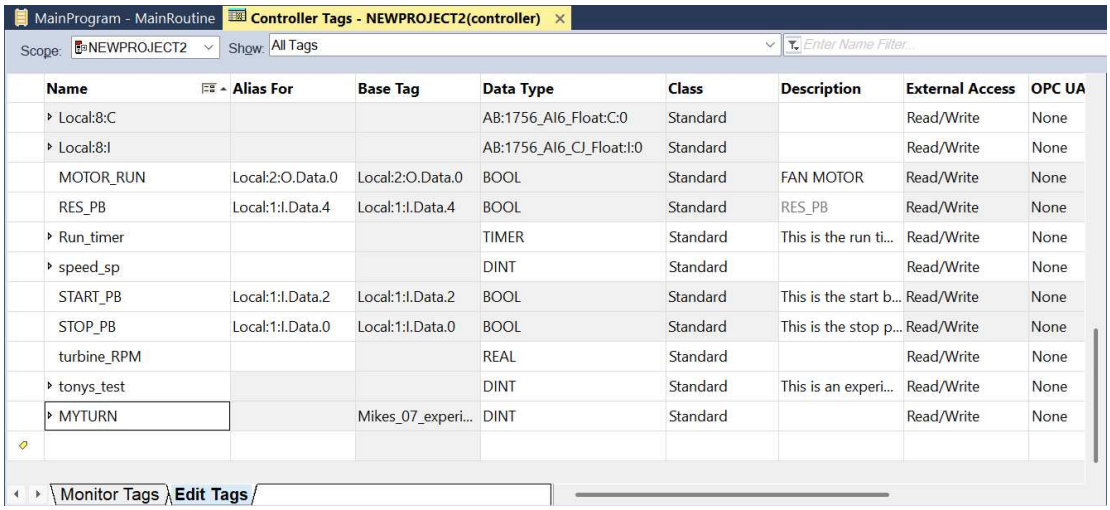
Rockwell Logix5000 programming is designed around the concept of alphanumerically-named *tags* rather than the fixed "file" designations used for bits and words in legacy Rockwell/Allen-Bradley controller products. Much about Logix5000 programming borrows from text-based programming languages such as C and C++, where tagnames serve the same purpose as *variable names* in those typed languages. For example, tags may be of a certain data type such as `BOOL` (bit) or `REAL` (floating-point) or `DINT` (double-width signed integer), or they may be comprised of multiple data elements (arrays) or multiple data types (structures) just like in C or C++, and just like variable names in C or C++ nearly any tagname is permitted using alphanumeric characters (with no spaces). Also, every tagname has a *scope* defining which levels of the PLC's programming it may be accessed. A tagname with scope set for `MainProgram` will only be readable and writable to that section of the program, just like a variable in C is local to a function if declared within that function. By contrast, a tagname with scope set at the project level is considered *global* and may be referenced by any portion of that PLC's program just like a variable in C declared at the start of the program before the `main` function.

Whenever possible, begin any new Rockwell Logix5000 PLC programming project by building out a comprehensive list of tags before writing any ladder-diagram code, making sure each tagname is clearly descriptive and not ambiguous. This is analogous to declaring all variables in C or C++ before writing any functions. Doing so helps organize your thoughts and helps make the writing of logic more self-documenting as each and every tag you use has a sensible name.

Four major categories of tags exist within Logix5000 systems:

- **Base tags** – these are tags belonging directly to a particular project on a particular PLC, and this is the default category for all new tags you as the programmer will create.
- **Alias tags** – these are tags linked to other existing tags with the sole purpose of accessing that data via a different tagname. Alias tags are commonly linked to hardware I/O addresses (e.g. `Local:3:I.Data.5` may be aliased to `Start_PB_switch` so when it appears near a contact instruction its purpose (i.e. a discrete bit controlled by a pushbutton switch wired to an input module) is obvious to anyone reading that ladder-diagram code.)
- **Produced tags** – a category of tags used in cyclic controller-to-controller communication, a produced tag residing in a controller for the purpose of broadcasting that data onto a digital network where it may be read into one or more tags within other controllers.
- **Consumed tags** – a category of tags used in cyclic controller-to-controller communication, a consumed tag residing in a controller for the purpose of receiving data broadcast (published) by another controller. Every consumed tag reads from *one* produced tag in some other controller, and this produced $\rightarrow$ consumed communication occurs at regular intervals without requiring any ladder-diagram code execution. The regular interval time is known as the *Requested Packet Interval* or *RPI*.

The following screenshot shows a partial tagname list configured into one particular Rockwell PLC used to monitor and control a turbine engine:



Name	Alias For	Base Tag	Data Type	Class	Description	External Access	OPC UA
Local:8:C			AB:1756_AI6_Float:C:0	Standard		Read/Write	None
Local:8:I			AB:1756_AI6_CJ_Float:I:0	Standard		Read/Write	None
MOTOR_RUN	Local:2:O.Data.0	Local:2:O.Data.0	BOOL	Standard	FAN MOTOR	Read/Write	None
RES_PB	Local:1:I.Data.4	Local:1:I.Data.4	BOOL	Standard	RES_PB	Read/Write	None
Run_timer			TIMER	Standard	This is the run ti...	Read/Write	None
speed_sp			DINT	Standard		Read/Write	None
START_PB	Local:1:I.Data.2	Local:1:I.Data.2	BOOL	Standard	This is the start b...	Read/Write	None
STOP_PB	Local:1:I.Data.0	Local:1:I.Data.0	BOOL	Standard	This is the stop p...	Read/Write	None
turbine_RPM			REAL	Standard		Read/Write	None
tonys_test			DINT	Standard	This is an experi...	Read/Write	None
MYTURN		Mikes_07_experi...	DINT	Standard		Read/Write	None

In this list we see multiple tags with empty `Alias For` and `Base Tag` fields. These are actually *base* tags, but they were arbitrarily created during the programming process: `turbine_RPM` is one of these, receiving the a calculated result performed by a math instruction in the program; `speed_sp` is another. Some of the other tags in this list contain hardware I/O addresses in both the `Alias For` and `Base Tag` fields, and these are *alias* tags: `RES_PB` is one of these, being the user-named tag for a "reset" pushbutton switch wired to channel 4 of a discrete input card inserted into slot 1 of the local PLC chassis. The tag named `MYTURN` is actually a *consumed* tag, receiving data at regular intervals from a produced tag residing inside a remote PLC named `Mikes_07_experiment`, which we know by noting that PLC's name in the `Base Tag` field. A single produced tag exists here but unfortunately bears no hint of being such: `tonys_test`. This tag may be identified as a produced tag only by viewing its properties in a different window.

Configuring produced and consumed tags:

- *Produced* and *Consumed* tags are special tags in the PLC where the PLC broadcasts data (Produce) or subscribes to broadcasted data (Consume) data. So, the data contained in a Produced tag in one PLC gets sent over a network to become a Consumed tag in one or more other PLCs. With Rockwell PLCs, this exchange happens automatically without need for additional rungs of logic in the program.
- Go offline. Creating a new Produced tag or a new Consumed tag requires you be offline!
- An easy way to create a new tag is to right-click on `Controller Tags` in the Controller Organizer list. This opens up a pop-up window with *all* the tag's parameters in one place which is generally more convenient than scrolling horizontally in the tag browser window and looking for hidden drop-down menu options.
- Under the `Type` option in the `New Tag` window, you will find the option to make the tag `Produced` or `Consumed`. If the tag is `Produced`, there isn't much more to specify except for tag `Name` and `Data Type`. If the tag is `Consumed`, you must also specify which PLC is producing it, and you do that by clicking the `Connection` button to the right of the `Type` field.
- When the `Consumed Tag Connection` pop-up window appears, there will be a `Producer` field where you specify the name of the PLC producing that tag. You will also specify the Produced tag name inside of that producer PLC in the `Remote Data` field.
- To be able to specify that producer PLC by name within the consumer PLC, it must already be listed in the consumer PLC's `Controller Organizer` list. This is done by going to the `I/O Configuration` category within that list, then drill down to the specific communications port (e.g. the `A1 Ethernet` port), and then right-click on that port to create a `New Module`. A `Select Module Type` pop-up window will appear, with a list of all Rockwell product numbers, one for each PLC module in the Logix5000 family. Choose the module number for the producer PLC's processor card (e.g. `1756-L902TS`) and then another window pops up called `New Module` where you specify a name for this PLC (which will be the producer parameter from the previous step) as well as its communication parameters (e.g. IP address). Once the name of this producer PLC is set and its IP address also set, then you can download all of these changes to your PLC, and it will begin looking for data to put into the new consumed tag.

Analog input module configuration:

- Ensure that the I/O module is not inhibited. You can tell by checking that the "OK" status light is solid green, rather than blinking.
- Go offline. You cannot change I/O module parameters while online!
- Right-click on the particular I/O module under `I/O Configuration` in the left-hand pane of Studio 5000 and select `Properties`. This will open up a new tab/window where you can tell the module how to function.
- Under `Overview`, click on `Device definition` and start selecting your desired options for the analog module. For example, here you would select either single-ended or differential channel type if both are supported.
- Download/accept all changes, then with the controller in Remote Run mode, view values in Controller Tags. Note that under the `View` menu you must have the `Value` display option checked to see any live values.
- Under `Local:X:I.ChYData` you should see the live analog input value (where `X` is the slot number into which the module has been plugged and `Y` is the channel number on the module).
- Sometimes Studio 5000 software does strange and unexpected things. For example, sometimes you have have to go offline then online again just to see live analog data for the channel (otherwise it may appear frozen at a fixed value).

Modbus communication:

As popular as Modbus is and has been for many years as an industrial communication standard, the lack of convenient support within Logix5000 PLCs for Modbus suggests the Rockwell philosophy is biased against its use. Nevertheless, one can download special Modbus instructions from Rockwell (called *Add-On Instructions* or *AOI*s) for Modbus/TCP communication. Alternatively, Rockwell manufactures *gateway* devices to map Modbus/TCP registers to EtherNet/IP points.

General tips:

- When placing instructions on the ladder-diagram MainRoutine area, you will need to drop that new instruction where the green dot appears on the run, otherwise it won't stick.
- Click on `Windows` → `Reset Default Layout` to restore the editor layout to its factory-default state, in case you do something that makes the viewing areas and toolbars unusable.
- Click on `View` → `Toolbars` → `Language Element` to toggle the quick-reference toolbox for instructions on and off.
- Right-clicking on any tagname appearing in the ladder logic program gives you the option to `Edit` properties of that tagname, opening up the same dialogue window used to create it.

4.3 Legacy Allen-Bradley memory maps and I/O addressing

A wise PLC programmer once told me that the first thing any aspiring programmer should learn about the PLC they intend to program is how the digital memory of that PLC is organized. This is sage advice for any programmer, especially on systems where memory is limited, and/or where I/O has a fixed association with certain locations in the system's memory. Virtually every microprocessor-based control system comes with a published *memory map* showing the organization of its limited memory: how much is available for certain functions, which addresses are linked to which I/O points, how different locations in memory are to be referenced by the programmer.

Discrete input and output channels on a PLC correspond to individual *bits* in the PLC's memory array. Similarly, analog input and output channels on a PLC correspond to multi-bit *words* (contiguous blocks of bits) in the PLC's memory. The association between I/O points and memory locations is by no means standardized between different PLC manufacturers, or even between different PLC models designed by the same manufacturer. This makes it difficult to write a general tutorial on PLC addressing, and so my ultimate advice is to consult the engineering references for the PLC system you intend to program.

The most common brand of PLC in use in the United States at the time of this writing (2019) is Allen-Bradley (Rockwell), and a great many of these Allen-Bradley PLCs still in service happen to use a unique form of I/O addressing[1] students tend to find confusing.

[1] The most modern Allen-Bradley PLCs have all but done away with fixed-location I/O addressing, opting instead for *tag name* based I/O addressing. However, enough legacy Allen-Bradley PLC systems still exist in industry to warrant coverage of these addressing conventions.

The following table shows a partial memory map for an Allen-Bradley SLC 500 PLC[2]:

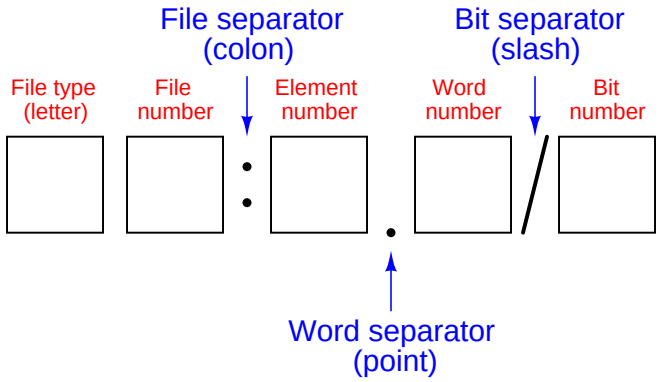
File number	File type	Logical address range
0	Output image	`O:0` to `O:30`
1	Input image	`I:0` to `I:30`
2	Status	`S:0` to `S:`n
3	Binary	`B3:0` to `B3:255`
4	Timers	`T4:0` to `T4:255`
5	Counters	`C5:0` to `C5:255`
6	Control	`R6:0` to `R6:255`
7	Integer	`N7:0` to `N7:255`
8	Floating-point	`F8:0` to `F8:255`
9	Network	`x9:0` to `x9:255`
10 through 255	User defined	`x10:0` to `x255:255`

Note that Allen-Bradley's use of the word "file" differs from personal computer parlance. In the SLC 500 controller, a "file" is a block of random-access memory used to store a particular type of data. By contrast, a "file" in a personal computer is a contiguous collection of data bits with collective meaning (e.g. a word processing file or a spreadsheet file), usually stored on the computer's hard disk drive. Within each of the Allen-Bradley PLC's "files" are multiple "elements," each element consisting of a set of bits (8, 16, 24, or 32) representing data. Elements are addressed by number following the colon after the file designator, and individual bits within each element addressed by a number following a slash mark. For example, the first bit (bit 0) of the second element in file 3 (Binary) would be addressed as `B3:2/0`.

In Allen-Bradley PLCs such as the SLC 500 and PLC-5 models, files 0, 1, and 2 are exclusively reserved for discrete outputs, discrete inputs, and status bits, respectively. Thus, the letter designators O, I, and S (file types) are redundant to the numbers 0, 1, and 2 (file numbers). Other file types such as B (binary), T (timers), C (counters), and others have their own default file numbers (3, 4, and 5, respectively), but may also be used in some of the user-defined file numbers (10 and above). For example, file 7 in an Allen-Bradley controller is reserved for data of the "integer" type (N), but integer data may also be stored in any file numbered 10 or greater at the user's discretion. Thus, file numbers and file type letters for data types other than output (O), input (I), and status (S) always appear together. You would not typically see an integer word addressed as `N:30` (integer word 30 in the PLC's memory) for example, but rather as `N7:30` (integer word 30 *in file 7* of the PLC's memory) to distinguish it from other integer word 30's that may exist in other files of the PLC's memory.

[2]Also called the *data table*, this map shows the addressing of memory areas reserved for programs entered by the user. Other areas of memory exist within the SLC 500 processor, but these other areas are inaccessible to the technician writing PLC programs.

This file-based addressing notation bears further explanation. When an address appears in a PLC program, special characters are used to separate (or "delimit") different fields from each other. The general scheme for Allen-Bradley SLC 500 PLCs is shown here:



Not all file types need to distinguish individual words and bits. Integer files (N), for example, consist of one 16-bit word for each element. For instance, `N7:5` would be the 16-bit integer word number five held in file seven. A discrete input file type (I), though, needs to be addressed as individual bits because each separate I/O point refers to a single bit. Thus, `I:3/7` would be bit number seven residing in input element three. The "slash" symbol is necessary when addressing discrete I/O bits because we do not wish to refer to all sixteen bits in a word when we just mean a single input or output point on the PLC. Integer numbers, by contrast, are collections of 16 bits each in the SLC 500 memory map, and so are usually addressed as entire words rather than bit-by-bit[3].

Certain file types such as timers are more complex. Each timer "element[4]" consists of *two* different 16-bit words (one for the timer's accumulated value, the other for the timer's target value) in addition to no less than *three* bits declaring the status of the timer (an "Enabled" bit, a "Timing" bit, and a "Done" bit). Thus, we must make use of both the decimal-point and slash separator symbols when referring to data within a timer. Suppose we declared a timer in our PLC program with the address `T4:2`, which would be timer number two contained in timer file four. If we wished to address that timer's current value, we would do so as `T4:2.ACC` (the "Accumulator" word of timer number two in file four). The "Done" bit of that same timer would be addressed as `T4:2/DN` (the "Done" bit of timer number two in file four)[5].

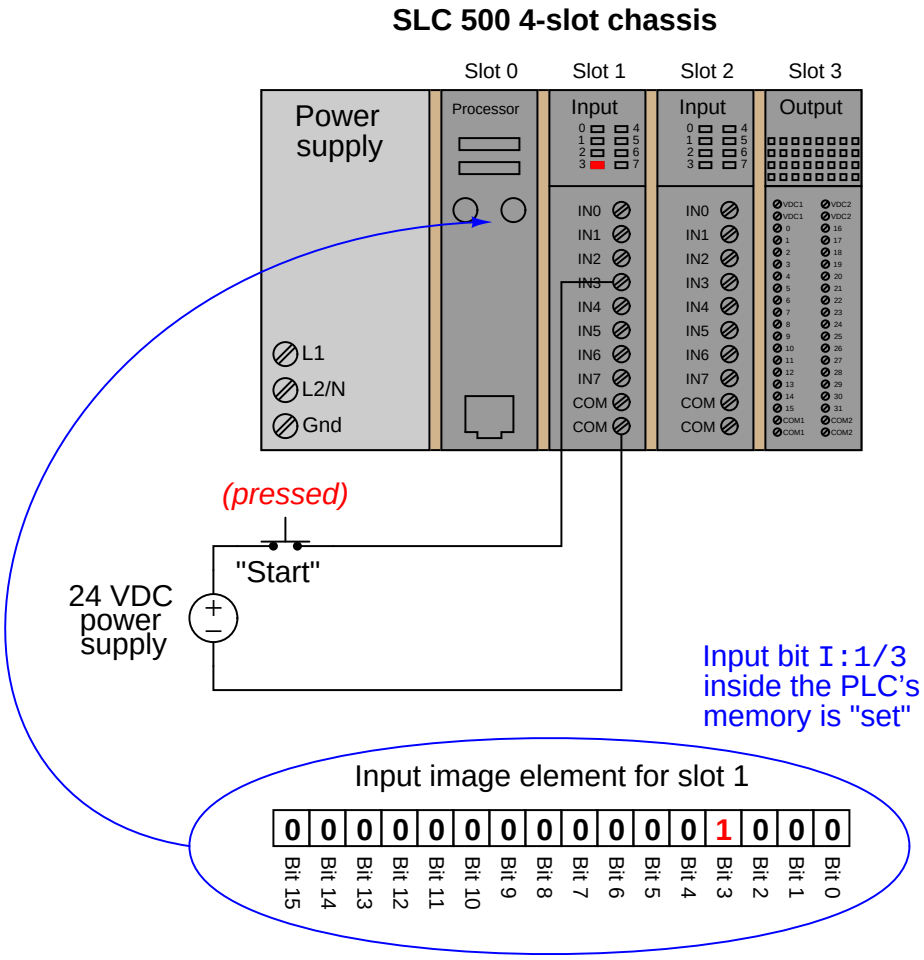
[3]This is not to say one *cannot* specify a particular bit in an otherwise whole word. In fact, this is one of the powerful advantages of Allen-Bradley's addressing scheme: it gives you the ability to precisely specify portions of data, even if that data is not generally intended to be portioned into smaller pieces!

[4]Programmers familiar with languages such as C and C++ might refer to an Allen-Bradley "element" as a *data structure*, each type with a set configuration of words and/or bits.

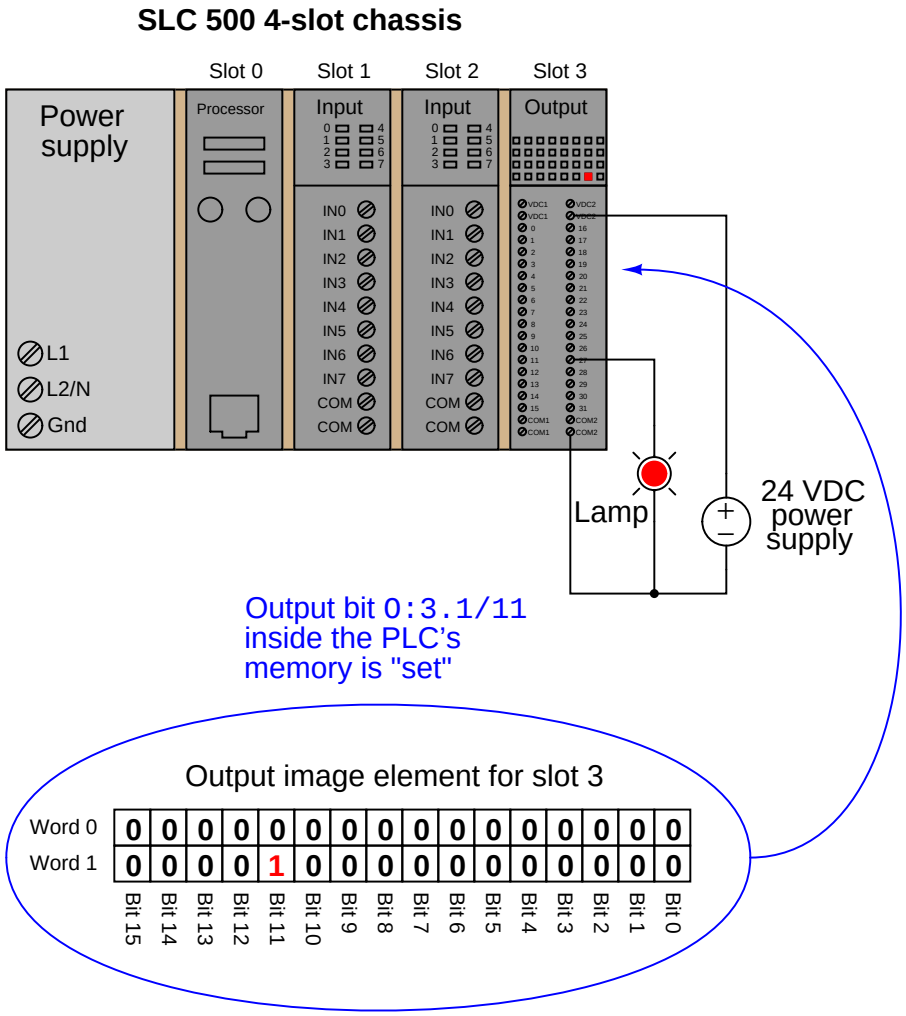
[5]Referencing the Allen-Bradley engineering literature, we see that the accumulator word may alternatively be addressed by number rather than by mnemonic, `T4:2.2` (word 2 being the accumulator word in the timer data structure), and that the "done" bit may be alternatively addressed as `T4:2.0/13` (bit number 13 in word 0 of the timer's data structure). The mnemonics provided by Allen-Bradley are certainly less confusing than referencing word and bit numbers for particular aspects of a timer's function!

A hallmark of the SLC 500's addressing scheme common to many legacy PLC systems is that the address labels for input and output bits explicitly reference the physical locations of the I/O channels. For instance, if an 8-channel discrete input card were plugged into slot 4 of an Allen-Bradley SLC 500 PLC, and you wished to specify the second bit (bit 1 out of a 0 to 7 range), you would address it with the following label: `I:4/1`. Addressing the seventh bit (bit number 6) on a discrete output card plugged into slot 3 would require the label `O:3/6`. In either case, the numerical structure of that label tells you exactly where the real-world input signal connects to the PLC.

To illustrate the relationship between physical I/O and bits in the PLC's memory, consider this example of an Allen-Bradley SLC 500 PLC, showing one of its discrete input channels energized (the switch being used as a "Start" switch for an electric motor):



If an input or output card possesses more than 16 bits – as in the case of the 32-bit discrete output card shown in slot 3 of the example SLC 500 rack – the addressing scheme further subdivides each element into *words* and bits (each "word" being 16 bits in length). Thus, the address for bit number 27 of a 32-bit input module plugged into slot 3 would be `I:3.1/11` (since bit 27 is equivalent to bit 11 of word 1 – word 0 addressing bits 0 through 15 and word 1 addressing bits 16 through 31):



A close-up photograph of a 32-bit DC input card for an Allen-Bradley SLC 500 PLC system shows this multi-word addressing:

The first sixteen input points on this card (the left-hand LED group numbered 0 through 15) are addressed `I:X.0/0` through `I:X.0/15`, with "`X`" referring to the slot number the card is plugged into. The next sixteen input points (the right-hand LED group numbered 16 through 31) are addressed `I:X.1/0` through `I:X.1/15`.

Legacy PLC systems typically reference each one of the I/O channels by labels such as "I:1/3" (or equivalent[6]) indicating the actual location of the input channel terminal on the PLC unit. The IEC 61131-3 programming standard refers to this channel-based addressing of I/O data points as *direct addressing*. A synonym for direct addressing is *absolute addressing*.

Addressing I/O bits directly by their card, slot, and/or terminal labels may seem simple and elegant, but it becomes very cumbersome for large PLC systems and complex programs. Every time a technician or programmer views the program, they must "translate" each of these I/O labels to some real-world device (e.g. "Input `I:1/3` is actually the *Start* pushbutton for the middle tank mixer motor") in order to understand the function of that bit. A later effort to enhance the clarity of PLC programming was the concept of addressing variables in a PLC's memory by arbitrary names rather than fixed codes. The IEC 61131-3 programming standard refers to this as *symbolic addressing* in contrast to "direct" (channel-based) addressing, allowing programmers arbitrarily

[6] Some systems such as the Texas Instruments 505 series used "X" labels to indicate discrete input channels and "Y" labels to indicate discrete output channels (e.g. input `X9` and output `Y14`). This same labeling convention is still used by Koyo in its DirectLogic and "CLICK" PLC models. Siemens continues a similar tradition of I/O addressing by using the letter "I" to indicate discrete inputs and the letter "Q" to indicate discrete outputs (e.g. input channel `I0.5` and output `Q4.1`).

name I/O channels in ways that are meaningful to the system as a whole. To use our simple motor "Start" switch example, it is now possible for the programmer to designate input `I:1/3` (an example of a *direct address*) as "`Motor_start_switch`" (an example of a *symbolic address*) within the program, thus greatly enhancing the readability of the PLC program. Initial implementations of this concept maintained direct addresses for I/O data points, with symbolic names appearing as supplements to the absolute addresses.

The modern trend in PLC addressing is to avoid the use of direct addresses such as `I:1/3` altogether, so they do not appear anywhere in the programming code. The Allen-Bradley "Logix" series of programmable logic controllers is the most prominent example of this new convention at the time of this writing. Each I/O point, regardless of type or physical location, is assigned a *tag name* which is meaningful in a real-world sense, and these tag names (or *symbols* as they are alternatively called) are referenced to absolute I/O channel locations by a database file. An important requirement of tag names is that they contain no space characters between words (e.g. instead of "`Motor start switch`", a tag name should use hyphens or underscore marks as spacing characters: "`Motor_start_switch`"), since spaces are generally assumed by computer programming languages to be delimiters (separators between different variables).

Chapter 5

Questions

This learning module, along with all others in the ModEL collection, is designed to be used in an inverted instructional environment where students independently read[1] the tutorials and attempt to answer questions on their own *prior* to the instructor's interaction with them. In place of lecture[2], the instructor engages with students in Socratic-style dialogue, probing and challenging their understanding of the subject matter through inquiry.

Answers are not provided for questions within this chapter, and this is by design. Solved problems may be found in the Tutorial and Derivation chapters, instead. The goal here is *independence*, and this requires students to be challenged in ways where others cannot think for them. Remember that you always have the tools of *experimentation* and *computer simulation* (e.g. SPICE) to explore concepts!

The following lists contain ideas for Socratic-style questions and challenges. Upon inspection, one will notice a strong theme of *metacognition* within these statements: they are designed to foster a regular habit of examining one's own thoughts as a means toward clearer thinking. As such these sample questions are useful both for instructor-led discussions as well as for self-study.

[1] Technical reading is an essential academic skill for any technical practitioner to possess for the simple reason that the most comprehensive, accurate, and useful information to be found for developing technical competence is in textual form. Technical careers in general are characterized by the need for continuous learning to remain current with standards and technology, and therefore any technical practitioner who cannot read well is handicapped in their professional development. An excellent resource for educators on improving students' reading prowess through intentional effort and strategy is the book textitReading For Understanding – How Reading Apprenticeship Improves Disciplinary Learning in Secondary and College Classrooms by Ruth Schoenbach, Cynthia Greenleaf, and Lynn Murphy.

[2] Lecture is popular as a teaching method because it is easy to implement: any reasonably articulate subject matter expert can talk to students, even with little preparation. However, it is also quite problematic. A good lecture always makes complicated concepts seem easier than they are, which is bad for students because it instills a false sense of confidence in their own understanding; reading and re-articulation requires more cognitive effort and serves to verify comprehension. A culture of teaching-by-lecture fosters a debilitating dependence upon direct personal instruction, whereas the challenges of modern life demand independent and critical thought made possible only by gathering information and perspectives from afar. Information presented in a lecture is ephemeral, easily lost to failures of memory and dictation; text is forever, and may be referenced at any time.

General challenges following tutorial reading

- Summarize as much of the text as you can in one paragraph of your own words. A helpful strategy is to explain ideas as you would for an intelligent child: as simple as you can without compromising too much accuracy.
- Simplify a particular section of the text, for example a paragraph or even a single sentence, so as to capture the same fundamental idea in fewer words.
- Where did the text make the most sense to you? What was it about the text's presentation that made it clear?
- Identify where it might be easy for someone to misunderstand the text, and explain why you think it could be confusing.
- Identify any new concept(s) presented in the text, and explain in your own words.
- Identify any familiar concept(s) such as physical laws or principles applied or referenced in the text.
- Devise a proof of concept experiment demonstrating an important principle, physical law, or technical innovation represented in the text.
- Devise an experiment to disprove a plausible misconception.
- Did the text reveal any misconceptions you might have harbored? If so, describe the misconception(s) and the reason(s) why you now know them to be incorrect.
- Describe any useful problem-solving strategies applied in the text.
- Devise a question of your own to challenge a reader's comprehension of the text.

General follow-up challenges for assigned problems

- Identify where any fundamental laws or principles apply to the solution of this problem, especially before applying any mathematical techniques.
- Devise a thought experiment to explore the characteristics of the problem scenario, applying known laws and principles to mentally model its behavior.
- Describe in detail your own strategy for solving this problem. How did you identify and organized the given information? Did you sketch any diagrams to help frame the problem?
- Is there more than one way to solve this problem? Which method seems best to you?
- Show the work you did in solving this problem, even if the solution is incomplete or incorrect.
- What would you say was the most challenging part of this problem, and why was it so?
- Was any important information missing from the problem which you had to research or recall?
- Was there any extraneous information presented within this problem? If so, what was it and why did it not matter?
- Examine someone else's solution to identify where they applied fundamental laws or principles.
- Simplify the problem from its given form and show how to solve this simpler version of it. Examples include eliminating certain variables or conditions, altering values to simpler (usually whole) numbers, applying a limiting case (i.e. altering a variable to some extreme or ultimate value).
- For quantitative problems, identify the real-world meaning of all intermediate calculations: their units of measurement, where they fit into the scenario at hand. Annotate any diagrams or illustrations with these calculated values.
- For quantitative problems, try approaching it qualitatively instead, thinking in terms of "increase" and "decrease" rather than definite values.
- For qualitative problems, try approaching it quantitatively instead, proposing simple numerical values for the variables.
- Were there any assumptions you made while solving this problem? Would your solution change if one of those assumptions were altered?
- Identify where it would be easy for someone to go astray in attempting to solve this problem.
- Formulate your own problem based on what you learned solving this one.

General follow-up challenges for experiments or projects

- In what way(s) was this experiment or project easy to complete?
- Identify some of the challenges you faced in completing this experiment or project.

- Show how thorough documentation assisted in the completion of this experiment or project.
- Which fundamental laws or principles are key to this system's function?
- Identify any way(s) in which one might obtain false or otherwise misleading measurements from test equipment in this system.
- What will happen if (component X) fails (open/shorted/etc.)?
- What would have to occur to make this system unsafe?

5.1 Conceptual reasoning

These questions are designed to stimulate your analytic and synthetic thinking[3]. In a Socratic discussion with your instructor, the goal is for these questions to prompt an extended dialogue where assumptions are revealed, conclusions are tested, and understanding is sharpened. Your instructor may also pose additional questions based on those assigned, in order to further probe and refine your conceptual understanding.

Questions that follow are presented to challenge and probe your understanding of various concepts presented in the tutorial. These questions are intended to serve as a guide for the Socratic dialogue between yourself and the instructor. Your instructor's task is to ensure you have a sound grasp of these concepts, and the questions contained in this document are merely a means to this end. Your instructor may, at his or her discretion, alter or substitute questions for the benefit of tailoring the discussion to each student's needs. The only absolute requirement is that each student is challenged and assessed at a level equal to or greater than that represented by the documented questions.

It is far more important that you convey your *reasoning* than it is to simply convey a correct answer. For this reason, you should refrain from researching other information sources to answer questions. What matters here is that *you* are doing the thinking. If the answer is incorrect, your instructor will work with you to correct it through proper reasoning. A correct answer without an adequate explanation of how you derived that answer is unacceptable, as it does not aid the learning or assessment process.

You will note a conspicuous lack of answers given for these conceptual questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your conceptual answers, where applicable, is to use circuit simulation software to explore the effects of changes made to circuits. For example, if one of these conceptual questions challenges you to predict the effects of altering some component parameter in a circuit, you may check the validity of your work by simulating that same parameter change within software and seeing if the results agree.

[3] *Analytical* thinking involves the "disassembly" of an idea into its constituent parts, analogous to dissection. *Synthetic* thinking involves the "assembly" of a new idea comprised of multiple concepts, analogous to construction. Both activities are high-level cognitive skills, extremely important for effective problem-solving, necessitating frequent challenge and regular practice to fully develop.

5.1.1 Reading outline and reflections

"Reading maketh a full man; conference a ready man; and writing an exact man" – Francis Bacon

Francis Bacon's advice is a blueprint for effective education: reading provides the learner with knowledge, writing focuses the learner's thoughts, and critical dialogue equips the learner to confidently communicate and apply their learning. Independent acquisition and application of knowledge is a powerful skill, well worth the effort to cultivate. To this end, students should read these educational resources closely, journal their own reflections on the reading, and discuss in detail their findings with classmates and instructor(s). You should be able to do all of the following after reading any instructional text:

☑ Briefly SUMMARIZE THE TEXT in the form of a journal entry documenting your learning as you progress through the course of study. Share this summary in dialogue with your classmates and instructor. Journaling is an excellent self-test of thorough reading because you cannot clearly express what you have not read or did not comprehend.

☑ Demonstrate ACTIVE READING STRATEGIES, including verbalizing your impressions as you read, simplifying long passages to convey the same ideas using fewer words, annotating text and illustrations with your own interpretations, working through mathematical examples shown in the text, cross-referencing passages with relevant illustrations and/or other passages, identifying problem-solving strategies applied by the author, etc. Technical reading is a special case of problem-solving, and so these strategies work precisely because they help solve any problem: paying attention to your own thoughts (metacognition), eliminating unnecessary complexities, identifying what makes sense, paying close attention to details, drawing connections between separated facts, and noting the successful strategies of others.

☑ Identify IMPORTANT THEMES, especially GENERAL LAWS and PRINCIPLES, expounded in the text and express them in the simplest of terms as though you were teaching an intelligent child. This emphasizes connections between related topics and develops your ability to communicate complex ideas to anyone.

☑ Form YOUR OWN QUESTIONS based on the reading, and then pose them to your instructor and classmates for their consideration. Anticipate both correct and incorrect answers, the incorrect answer(s) assuming one or more plausible misconceptions. This helps you view the subject from different perspectives to grasp it more fully.

☑ Devise EXPERIMENTS to test claims presented in the reading, or to disprove misconceptions. Predict possible outcomes of these experiments, and evaluate their meanings: what result(s) would confirm, and what would constitute disproof? Running mental simulations and evaluating results is essential to scientific and diagnostic reasoning.

☑ Specifically identify any points you found CONFUSING. The reason for doing this is to help diagnose misconceptions and overcome barriers to learning.

5.1.2 Foundational concepts

Correct analysis and diagnosis of electric circuits begins with a proper understanding of some basic concepts. The following is a list of some important concepts referenced in this module's full tutorial. Define each of them in your own words, and be prepared to illustrate each of these concepts with a description of a practical example and/or a live demonstration.

???

???

???

???

???

???

???

???

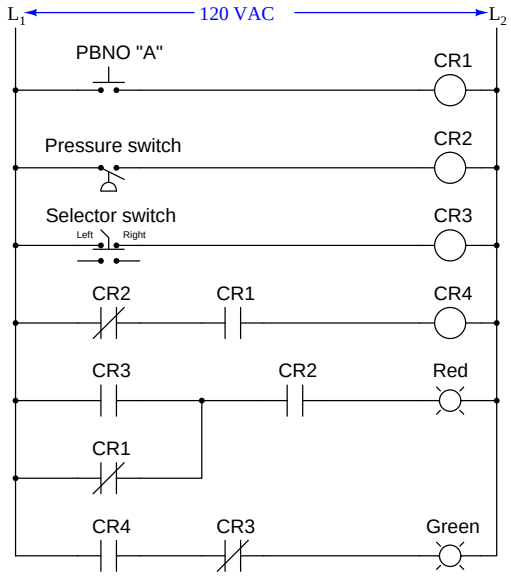
???

???

???

5.1.3 Relay ladder logic analogy for a PLC

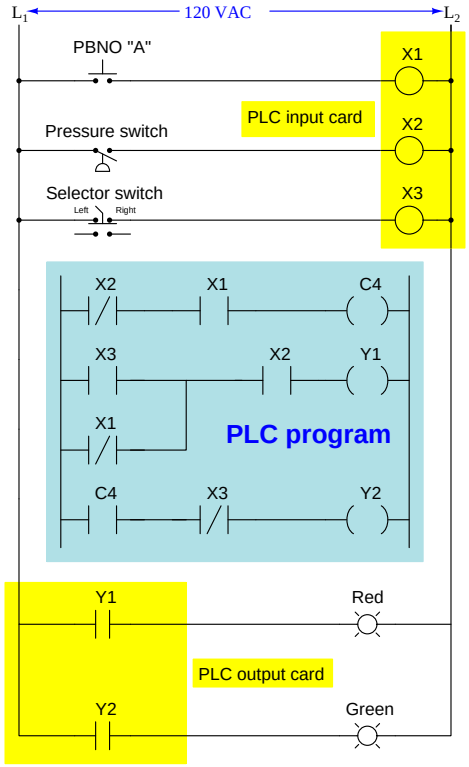
Analyze the status of all relay contacts and lamps in this hard-wired relay "ladder logic" control circuit, annotating the diagram to show electrical conductivity and non-conductivity of all contacts and energization statuses of all relay coils and loads:



Assume the following input conditions:

- Pushbutton switch *unpressed*
- Pressure *above* trip threshold
- Selector switch in its *right-hand* position

Now, analyze the status of this PLC-controlled system assuming the same input conditions. Note the distinction between the 120 VAC circuitry and the "virtual circuit" in the blue-shaded area representing the program executed by the PLC's microprocessor. As with the relay-based system, annotate all real-world contact conductivities and coil energization statuses, as well as show which of the PLC program's elements should be "colored" to reflect virtual conductivity and virtual energization:



Assume the same input conditions:

- Pushbutton switch *unpressed*
- Pressure *above* trip threshold
- Selector switch in its *right-hand* position

How is the PLC-controlled system similar to the hard-wired relay control system? How is it different?

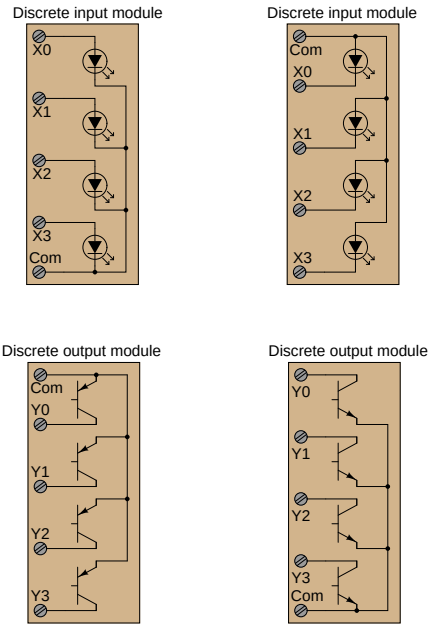
Challenges

- What advantages does the PLC enjoy over the hard-wired relay control system?
- What advantages does the electromechanical relay design enjoy over the PLC control system?

5.1.4 Sourcing versus sinking PLC I/O

Discrete (on/off) I/O for PLCs often works on AC (alternating current) power. AC input circuitry usually consists of an optocoupler (LED) with rectification and a large dropping resistor to allow 120 Volt AC operation. AC output circuitry usually consists of TRIACs. Explain how both of these technologies work.

DC I/O for a PLC generally consists of optocoupled LEDs for inputs and bipolar transistors for outputs. Some examples are shown in the following schematics. Note carefully the different variations:



Determine for each of these input and output module types, whether they would be properly designated *sourcing* or *sinking*.

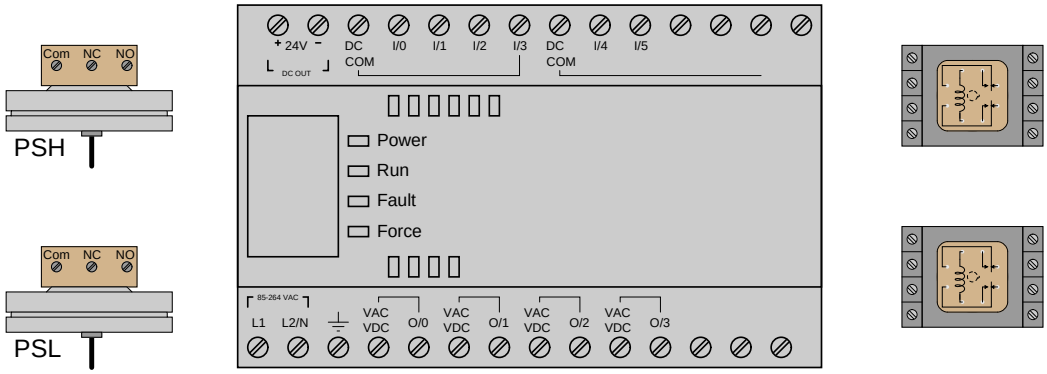
Lastly, sketch components and wires to make at least two channels on each of these I/O modules functional. For each input module this means an electrical source and at least two switches. For each output module this means an electrical source and at least two loads (e.g. lamps).

Challenges

- Determine how real input and output devices (e.g. switches, solenoid coils) would need to be connected to the I/O terminals of these modules.
- Explain why we might need to connect a semiconductor diode across the terminals of an inductive load controlled by a PLC output module.

5.1.5 Sketching wires to PLC discrete I/O

Sketch the wires necessary to connect two pressure switches and two relay coils to the following Allen-Bradley MicroLogix 1000 PLC (model 1761-L10BWA, with 6 discrete DC inputs either sourcing or sinking, and 4 discrete relay contact outputs). Be sure to wire the two switches so they *source* current to the PLC's inputs (the low-pressure switch to `I/2` and the high-pressure switch to `I/5`, normally-open contacts on both) and wire the relay coils so the PLC *sources* current to them (`O/0` and `O/1`):



Challenges

- Define "sinking" and "sourcing" as these terms apply to PLC I/O terminals.

5.2 Diagnostic reasoning

These questions are designed to stimulate your deductive and inductive thinking, where you must apply general principles to specific scenarios (deductive) and also derive conclusions about the failed circuit from specific details (inductive). In a Socratic discussion with your instructor, the goal is for these questions to reinforce your recall and use of general circuit principles and also challenge your ability to integrate multiple symptoms into a sensible explanation of what's wrong in a circuit. Your instructor may also pose additional questions based on those assigned, in order to further challenge and sharpen your diagnostic abilities.

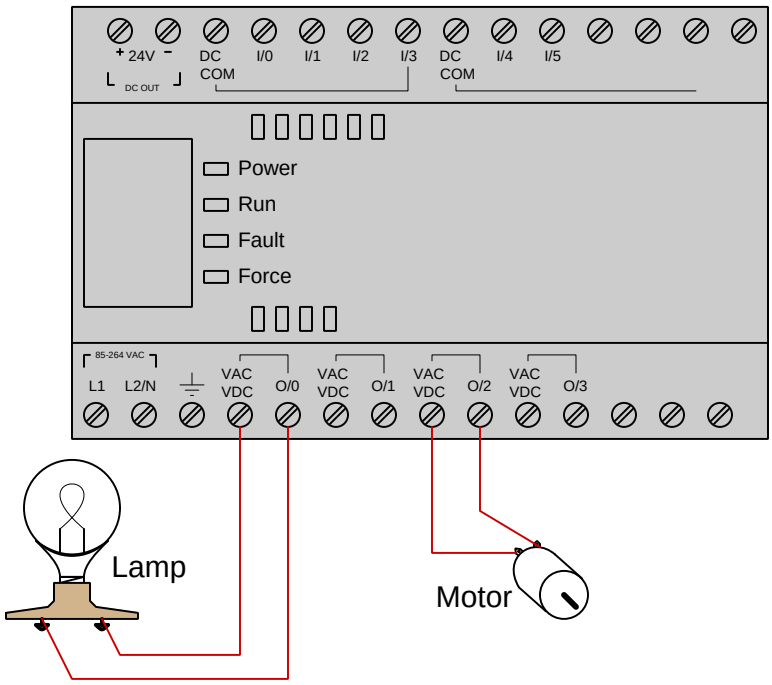
As always, your goal is to fully *explain* your analysis of each problem. Simply obtaining a correct answer is not good enough – you must also demonstrate sound reasoning in order to successfully complete the assignment. Your instructor's responsibility is to probe and challenge your understanding of the relevant principles and analytical processes in order to ensure you have a strong foundation upon which to build further understanding.

You will note a conspicuous lack of answers given for these diagnostic questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your diagnostic answers, where applicable, is to use circuit simulation software to explore the effects of faults placed in circuits. For example, if one of these diagnostic questions requires that you predict the effect of an open or a short in a circuit, you may check the validity of your work by simulating that same fault (substituting a very high resistance in place of that component for an open, and substituting a very low resistance for a short) within software and seeing if the results agree.

5.2.1 Incorrect PLC output wiring

The PLC shown below is supposed to control the energization of a lamp and a small motor. Someone wired both loads to different PLC output channels as shown in the illustration below, but was surprised to find neither load energized when its respective PLC output channel activated:



Explain what is wrong in this circuit, and then re-sketch the wiring so that these loads will function properly. Incidentally, this same error happens to be a *very* common misconception among students new to PLCs. Identify what the fundamental misconception is, and how it may be remedied.

Challenges

- What information should we ideally have regarding the PLC and the two loads shown in order to design a complete circuit that will function safely and reliably?
- From the information available in the illustration, is it possible to formulate an educated guess about the type of discrete output channels offered by this PLC?

Appendix A

Problem-Solving Strategies

The ability to solve complex problems is arguably one of the most valuable skills one can possess, and this skill is particularly important in any science-based discipline.

- Study principles, not procedures. Don't be satisfied with merely knowing how to compute solutions – learn *why* those solutions work.
- Identify what it is you need to solve, identify all relevant data, identify all units of measurement, identify any general principles or formulae linking the given information to the solution, and then identify any "missing pieces" to a solution. Annotate all diagrams with this data.
- Sketch a diagram to help visualize the problem. When building a real system, always devise a plan for that system and analyze its function *before* constructing it.
- Follow the units of measurement and meaning of every calculation. If you are ever performing mathematical calculations as part of a problem-solving procedure, and you find yourself unable to apply each and every intermediate result to some aspect of the problem, it means you don't understand what you are doing. Properly done, every mathematical result should have practical meaning for the problem, and not just be an abstract number. You should be able to identify the proper units of measurement for each and every calculated result, and show where that result fits into the problem.
- Perform "thought experiments" to explore the effects of different conditions for theoretical problems. When troubleshooting real systems, perform *diagnostic tests* rather than visually inspecting for faults, the best diagnostic test being the one giving you the most information about the nature and/or location of the fault with the fewest steps.
- Simplify the problem until the solution becomes obvious, and then use that obvious case as a model to follow in solving the more complex version of the problem.
- Check for exceptions to see if your solution is incorrect or incomplete. A good solution will work for *all* known conditions and criteria. A good example of this is the process of testing scientific hypotheses: the task of a scientist is not to find support for a new idea, but rather to *challenge* that new idea to see if it holds up under a battery of tests. The philosophical

principle of *reductio ad absurdum* (i.e. disproving a general idea by finding a specific case where it fails) is useful here.

- Work "backward" from a hypothetical solution to a new set of given conditions.
- Add quantities to problems that are qualitative in nature, because sometimes a little math helps illuminate the scenario.
- Sketch graphs illustrating how variables relate to each other. These may be quantitative (i.e. with realistic number values) or qualitative (i.e. simply showing increases and decreases).
- Treat quantitative problems as qualitative in order to discern the relative magnitudes and/or directions of change of the relevant variables. For example, try determining what happens if a certain variable were to increase or decrease before attempting to precisely calculate quantities: how will each of the dependent variables respond, by increasing, decreasing, or remaining the same as before?
- Consider limiting cases. This works especially well for qualitative problems where you need to determine which direction a variable will change. Take the given condition and magnify that condition to an extreme degree as a way of simplifying the direction of the system's response.
- Check your work. This means regularly testing your conclusions to see if they make sense. This does *not* mean repeating the same steps originally used to obtain the conclusion(s), but rather to use some other means to check validity. Simply repeating procedures often leads to *repeating the same errors* if any were made, which is why alternative paths are better.

Appendix B

Instructional philosophy

B.1 First principles of learning

- **Anyone can learn anything** given appropriate time, effort, resources, challenges, encouragement, and expectations. Dedicating time and investing effort are the student's responsibility; providing resources, challenges, and encouragement are the teacher's responsibility; high expectations are a responsibility shared by both student and teacher.

- **Transfer is not automatic.** The human mind has a natural tendency to compartmentalize information, which means the process of taking knowledge learned in one context and applying it to another usually does not come easy and therefore should never be taken for granted.

- **Learning is iterative.** The human mind rarely learns anything perfectly on the first attempt. Anticipate mistakes and plan for multiple tries to achieve full understanding, using the lessons of those mistakes as feedback to guide future attempts.

- **Information is absorbed, but understanding is created.** Facts and procedures may be memorized easily enough by repeated exposure, but the ability to reliably apply principles to novel scenarios only comes through intense personal effort. This effort is fundamentally creative in nature: casting new concepts in one's own words, running mental experiments to perceive patterns and trends, etc. These acts of making knowledge "one's own" need not be perfect in order to be effective, as the value lies in the activity and not necessarily the finished product. Depth of understanding and permanence of memory are both directly related to mental effort during the learning process – i.e. *easy learning is shallow and fleeting.*

- **Education trumps training.** There is no such thing as an entirely isolated subject, as all fields of knowledge are connected. Training is narrowly-focused and task-oriented. Education is broad-based and principle-oriented. When preparing for a life-long technical career, education beats training every time.

- **Character matters.** Poor habits are more destructive than deficits of knowledge or skill. This is especially true in collective endeavors, where a team's ability to function depends on trust between its members. Simply put, no one wants an untrustworthy person on their team. An essential component of education then, is character development.

- **People learn to be responsible by bearing responsibility.** An irresponsible person is someone who has never *had* to be responsible for anything that mattered enough to them. Just as anyone can learn anything, anyone can become responsible if the personal cost of irresponsibility becomes high enough.

- **What gets measured, gets done.** Accurate and relevant assessment of learning is key to ensuring all students learn. Therefore, it is imperative to measure what matters.

- **Failure is nothing to fear.** Every human being fails, and fails in multiple ways at multiple times. Eventual success only happens when we don't stop trying.

B.2 Proven strategies for instructors

- Assume every student is capable of learning anything they desire given the proper conditions. Treat them as capable adults by granting real responsibility and avoiding artificial incentives such as merit or demerit points.
- Create a consistent culture of high expectations across the entire program of study. Demonstrate and encourage patience, persistence, and a healthy sense of self-skepticism. Anticipate and de-stigmatize error. Teach respect for the capabilities of others as well as respect for one's own fallibility.
- Replace lecture with "inverted" instruction, where students first encounter new concepts through reading and then spend class time in Socratic dialogue with the instructor exploring those concepts and solving problems individually. There is a world of difference between observing someone solve a problem versus actually solving a problem yourself, and so the point of this form of instruction is to place students in a position where they *cannot* passively observe.
- Require students to read extensively, write about what they learn, and dialogue with you and their peers to sharpen their understanding. Apply Francis Bacon's advice that "reading maketh a full man; conference a ready man; and writing an exact man". These are complementary activities helping students expand their confidence and abilities.
- Use artificial intelligence (AI) to challenge student understanding rather than merely provide information. Find productive ways for AI to critique students' clarity of thought and of expression, for example by employing AI as a Socratic-style interlocutor or as a reviewer of students' journals. Properly applied, AI has the ability to expand student access to critical review well outside the bounds of their instructor's reach.
- Build frequent and rapid feedback into the learning process so that students know at all times how well they are learning, to identify problems early and fix them before they grow. Model the intellectual habit of self-assessing and self-correcting your own understanding (i.e. a cognitive *feedback loop*), encouraging students to do the same.
- Use "mastery" as the standard for every assessment, which means the exam or experiment or project must be done with 100% competence in order to pass. Provide students with multiple opportunity for re-tries (different versions of the assessment every time).
- Require students to devise their own hypotheses and procedures on all experiments, so that the process is truly a scientific one. Have students assess their proposed experimental procedures for risk and devise mitigations for those risks. Let nothing be pre-designed about students' experiments other than a stated task (i.e. what principle the experiment shall test) at the start and a set of demonstrable knowledge and skill objectives at the end.
- Have students build as much of their lab equipment as possible: building power sources, building test assemblies[1], and building complete working systems (no kits!). In order to provide

[1] In the program I teach, every student builds their own "Development Board" consisting of a metal chassis with DIN rail, terminal blocks, and an AC-DC power supply of their own making which functions as a portable lab environment they can use at school as well as take home.

this same "ground-up" experience for every new student, this means either previous students take their creations with them, or the systems get disassembled in preparation for the new students, or the systems grow and evolve with each new student group.

- Incorporate external accountability for you and for your students, continuously improving the curriculum and your instructional methods based on proven results. Have students regularly network with active professionals through participation in advisory committee meetings, service projects, tours, jobshadows, internships, etc. Practical suggestions include requiring students to design and build projects for external clients (e.g. community groups, businesses, different departments within the institution), and also requiring students attend all technical advisory committee meetings and to dialogue with the industry representatives at those meetings.
- Repeatedly explore difficult-to-learn concepts across multiple courses, so that students have multiple opportunities to build their understanding.
- Relate all new concepts, whenever possible, to previous concepts and to relevant physical laws. Challenge each and every student, every day, to *reason* from concept to concept and to explain the logical connections between. Challenge students to verify their conclusions by multiple approaches (e.g. double-checking their work using different methods). Ask *"Why?"* often.
- Maintain detailed records on each student's performance and share these records privately with them. These records should include academic performance as well as professionally relevant behavioral tendencies.
- Hold mandatory "check-in" meetings between all program faculty and each new student during their first term. Offer these to all other students as an option, except for any students continuing to manifest unprofessional behaviors, poor academic performance, or who have some other need for a face-to-face meeting with faculty.
- Address problems while they are small, before they grow larger. This is equally true for tutoring technical concepts as it is for helping students build professional habits.
- Build rigorous quality control into the curriculum to ensure every student masters every important concept, and that the mastery is retained over time. This includes (1) review questions added to every exam to re-assess knowledge taught in previous terms, (2) cumulative exams at the end of every term to re-assess all important concepts back to the very beginning of the program, and (3) review assessments in practical (hands-on) coursework to ensure critically-important skills were indeed taught and are still retained. What you will find by doing this is that it actually boosts retention of students by ensuring that important knowledge gets taught and is retained over long spans of time. In the absence of such quality control, student learning and retention tends to be spotty and this contributes to drop-out and failure rates later in their education.
- Finally, *never rush learning.* Education is not a race. Give your students ample time to digest complex ideas, as you continually remind yourself of just how long it took you to achieve mastery! Long-term retention and the consistently correct application of concepts are always the result of *focused effort over long periods of time* which means there are no shortcuts to learning.

B.3 Proven strategies for students

The single most important piece of advice I have for any student of any subject is to take responsibility for your own development in all areas of life including mental development. Expecting others in your life to entirely guide your own development is a recipe for disappointment. This is just as true for students enrolled in formal learning institutions as it is for auto-didacts pursuing learning entirely on their own. Learning to think in new ways is key to being able to gainfully use information, to make informed decisions about your life, and to best serve those you care about. With this in mind, I offer the following advice to students:

- **Approach all learning as valuable**. No matter what course you take, no matter who you learn from, no matter the subject, there is something useful in every learning experience. If you don't see the value of every new experience, you are not looking closely enough!
- **Continually challenge yourself**. Let other people take shortcuts and find easy answers to easy problems. The purpose of education is to stretch your mind, in order to shape it into a more powerful tool. This doesn't come by taking the path of least resistance. An excellent analogy for an empowering education is productive physical exercise: becoming stronger, more flexible, and more persistent only comes through intense personal effort.
- **Master the use of language**. This includes reading extensively, writing every day, listening closely, and speaking articulately. To a great extent language channels and empowers thought, so the better you are at wielding language the better you will be at grasping abstract concepts and articulating them not only for your benefit but for others as well.
- **Do not limit yourself to the resources given to you**. Read books that are not on the reading list. Run experiments that aren't assigned to you. Form study groups outside of class. Take an entrepreneurial approach to your own education, as though it were a business you were building for your future benefit.
- **Express and share what you learn**. Take every opportunity to teach what you have learned to others, as this will not only help them but will also strengthen your own understanding[2].
- Realize that **no one can give you understanding**, just as no one can give you physical fitness. These both must be *built*.
- **Above all, recognize that learning is hard work, and that a certain level of frustration is unavoidable**. There are times when you will struggle to grasp some of these concepts, and that struggle is a natural thing. Take heart that it will yield with persistent and varied[3] effort, and never give up! That concepts don't immediately come to you is not a sign of something wrong, but rather of something right: that you have found a worthy challenge!

[2] On a personal note, I was surprised to learn just how much my own understanding of electronics and related subjects was strengthened by becoming a teacher. When you are tasked every day with helping other people grasp complex topics, it catalyzes your own learning by giving you powerful incentives to study, to articulate your thoughts, and to reflect deeply on the process of learning.

[3] As the old saying goes, "Insanity is trying the same thing over and over again, expecting different results." If you find yourself stumped by something in the text, you should attempt a different approach. Alter the thought experiment, change the mathematical parameters, do whatever you can to see the problem in a slightly different light, and then the solution will often present itself more readily.

B.4 Design of these learning modules

"The unexamined circuit is not worth energizing" – Socrates (if he had taught electricity)

These learning modules, although useful for self-study, were designed to be used in a formal learning environment where a subject-matter expert challenges students to digest the content and exercise their critical thinking abilities in the answering of questions and in the construction and testing of working circuits. Every effort has been made to embed the following instructional and assessment philosophies within:

- The first goal of education is to enhance clear and independent thought, in order that every student reach their fullest potential in a highly complex and inter-dependent world. Robust reasoning is *always* more important than particulars of any subject matter, because its application is universal.
- Literacy is fundamental to independent learning and thought because text continues to be the most efficient way to communicate complex ideas over space and time. Those who cannot read with ease are limited in their ability to acquire knowledge and perspective.
- Articulate communication is fundamental to work that is complex and interdisciplinary.
- Faulty assumptions and poor reasoning are best corrected through challenge, not presentation. The rhetorical technique of *reductio ad absurdum* (disproving an assertion by exposing an absurdity) works well to discipline student's minds, not only to correct the problem at hand but also to learn how to detect and correct future errors.
- Important principles should be repeatedly explored and widely applied throughout a course of study, not only to reinforce their importance and help ensure their mastery, but also to showcase the interconnectedness and utility of knowledge.

These learning modules were expressly designed to be used in an "inverted" teaching environment[4] where students first read the introductory and tutorial chapters on their own, then individually attempt to answer the questions and construct working circuits according to the experiment and project guidelines. The instructor never lectures, but instead meets regularly with each individual student to review their progress, answer questions, identify misconceptions, and challenge the student to new depths of understanding through further questioning. Regular meetings between instructor and student should resemble a Socratic[5] dialogue, where questions serve as scalpels to dissect topics and expose assumptions. The student passes each module only after consistently demonstrating their ability to logically analyze and correctly apply all major concepts in each question or project/experiment. The instructor must be vigilant in probing each student's understanding to ensure they are truly *reasoning* and not just *memorizing*. This is why "Challenge" points appear throughout, as prompts for students to think deeper about topics and as starting points for instructor queries. Sometimes these challenge points require additional knowledge that hasn't been covered in the series to answer in full. This is okay, as the major purpose of the Challenges is to stimulate analysis and synthesis on the part of each student.

The instructor must possess enough mastery of the subject matter and awareness of students' reasoning to generate their own follow-up questions to practically any student response. Even completely correct answers given by the student should be challenged by the instructor for the purpose of having students practice articulating their thoughts and defending their reasoning. Conceptual errors committed by the student should be exposed and corrected not by direct instruction, but rather by reducing the errors to an absurdity[6] through well-chosen questions and thought experiments posed by the instructor. Becoming proficient at this style of instruction requires time and dedication, but the positive effects on critical thinking for both student and instructor are spectacular.

An inspection of these learning modules reveals certain unique characteristics. One of these is a bias toward thorough explanations in the tutorial chapters. Without a live instructor to explain concepts and applications to students, the text itself must fulfill this role. This philosophy results in lengthier explanations than what you might typically find in a textbook, each step of the reasoning process fully explained, including footnotes addressing common questions and concerns students raise while learning these concepts. Each tutorial seeks to not only explain each major concept in sufficient detail, but also to explain the logic of each concept and how each may be developed

[4] In a traditional teaching environment, students first encounter new information via *lecture* from an expert, and then independently apply that information via *homework*. In an "inverted" course of study, students first encounter new information via *homework*, and then independently apply that information under the scrutiny of an expert. The expert's role in lecture is to simply *explain*, but the expert's role in an inverted session is to *challenge*, *critique*, and if necessary *explain* where gaps in understanding still exist.

[5] Socrates is a figure in ancient Greek philosophy famous for his unflinching style of questioning. Although he authored no texts, he appears as a character in Plato's many writings. The essence of Socratic philosophy is to leave no question unexamined and no point of view unchallenged. While purists may argue a topic such as electric circuits is too narrow for a true Socratic-style dialogue, I would argue that the essential thought processes involved with scientific reasoning on *any* topic are not far removed from the Socratic ideal, and that students of electricity and electronics would do very well to challenge assumptions, pose thought experiments, identify fallacies, and otherwise employ the arsenal of critical thinking skills modeled by Socrates.

[6] This rhetorical technique is known by the Latin phrase *reductio ad absurdum*. The concept is to expose errors by counter-example, since only one solid counter-example is necessary to disprove a universal claim. As an example of this, consider the common misconception among beginning students of electricity that voltage cannot exist without current. One way to apply *reductio ad absurdum* to this statement is to ask how much current passes through a fully-charged battery connected to nothing (i.e. a clear example of voltage existing without current).

from "first principles". Again, this reflects the goal of developing clear and independent thought in students' minds, by showing how clear and logical thought was used to forge each concept. Students benefit from witnessing a model of clear thinking in action, and these tutorials strive to be just that.

Another feature of these learning modules is that they do not treat topics in isolation. Rather, important concepts are introduced early in the series, and appear repeatedly as stepping-stones toward other concepts in subsequent modules. This helps to avoid the "compartmentalization" of knowledge, demonstrating the inter-connectedness of concepts and simultaneously reinforcing them. Each module is fairly complete in itself, reserving the beginning of its tutorial to a review of foundational concepts.

To high standards of education,

Tony R. Kuphaldt

Appendix C

Tools used

I am indebted to the developers of many open-source software applications in the creation of these learning modules. The following is a list of these applications with some commentary on each.

You will notice a theme common to many of these applications: a bias toward *code*. Although I am by no means an expert programmer in any computer language, I understand and appreciate the flexibility offered by code-based applications where the user (you) enters commands into a plain ASCII text file, which the software then reads and processes to create the final output. Code-based computer applications are by their very nature *extensible*, while WYSIWYG (What You See Is What You Get) applications are generally limited to whatever user interface the developer makes for you.

The `GNU/Linux` computer operating system

> There is so much to be said about Linus Torvalds' `Linux` and Richard Stallman's `GNU` project. First, to credit just these two individuals is to fail to do justice to the *mob* of passionate volunteers who contributed to make this amazing software a reality. I first learned of `Linux` back in 1996, and have been using this operating system on my personal computers almost exclusively since then. It is *free*, it is completely *configurable*, and it permits the continued use of highly efficient `Unix` applications and scripting languages (e.g. shell scripts, Makefiles, `sed`, `awk`) developed over many decades. `Linux` not only provided me with a powerful computing platform, but its open design served to inspire my life's work of creating open-source educational resources.

Bram Moolenaar's `Vim` text editor

> Writing code for any code-based computer application requires a *text editor*, which may be thought of as a word processor strictly limited to outputting plain-ASCII text files. Many good text editors exist, and one's choice of text editor seems to be a deeply personal matter within the programming world. I prefer `Vim` because it operates very similarly to `vi` which is ubiquitous on `Unix`/`Linux` operating systems, and because it may be entirely operated via keyboard (i.e. no mouse required) which makes it fast to use.

Donald Knuth's TEX typesetting system

Developed in the late 1970's and early 1980's by computer scientist extraordinaire Donald Knuth to typeset his multi-volume magnum opus *The Art of Computer Programming*, this software allows the production of formatted text for screen-viewing or paper printing, all by writing plain-text code to describe how the formatted text is supposed to appear. TEX is not just a markup language for documents, but it is also a Turing-complete programming language in and of itself, allowing useful algorithms to be created to control the production of documents. Simply put, TEX *is a programmer's approach to word processing*. Since TEX is controlled by code written in a plain-text file, this means anyone may read that plain-text file to see exactly how the document was created. This openness afforded by the code-based nature of TEX makes it relatively easy to learn how other people have created their own TEX documents. By contrast, examining a beautiful document created in a conventional WYSIWYG word processor such as Microsoft `Word` suggests nothing to the reader about *how* that document was created, or what the user might do to create something similar. As Mr. Knuth himself once quipped, conventional word processing applications should be called WYSIAYG (What You See Is *All* You Get).

Leslie Lamport's LATEX extensions to TEX

Like all true programming languages, TEX is inherently extensible. So, years after the release of TEX to the public, Leslie Lamport decided to create a massive extension allowing easier compilation of book-length documents. The result was LATEX, which is the markup language used to create all ModEL module documents. You could say that TEX is to LATEX as `C` is to `C++`. This means it is permissible to use any and all TEX commands within LATEX source code, and it all still works. Some of the features offered by LATEX that would be challenging to implement in TEX include automatic index and table-of-content creation.

Tim Edwards' `Xcircuit` drafting program

This wonderful program is what I use to create all the schematic diagrams and illustrations (but not photographic images or mathematical plots) throughout the ModEL project. It natively outputs PostScript format which is a true vector graphic format (this is why the images do not pixellate when you zoom in for a closer view), and it is so simple to use that I have never had to read the manual! Object libraries are easy to create for `Xcircuit`, being plain-text files using PostScript programming conventions. Over the years I have collected a large set of object libraries useful for drawing electrical and electronic schematics, pictorial diagrams, and other technical illustrations.

`Gimp` graphic image manipulation program

Essentially an open-source clone of Adobe's `PhotoShop`, I use `Gimp` to resize, crop, and convert file formats for all of the photographic images appearing in the ModEL modules. Although `Gimp` does offer its own scripting language (called `Script-Fu`), I have never had occasion to use it. Thus, my utilization of `Gimp` to merely crop, resize, and convert graphic images is akin to using a sword to slice bread.

`SPICE` circuit simulation program

`SPICE` is to circuit analysis as TEX is to document creation: it is a form of markup language designed to describe a certain object to be processed in plain-ASCII text. When the plain-text "source file" is compiled by the software, it outputs the final result. More modern circuit analysis tools certainly exist, but I prefer `SPICE` for the following reasons: it is *free*, it is *fast*, it is *reliable*, and it is a fantastic tool for *teaching* students of electricity and electronics how to write simple code. I happen to use rather old versions of `SPICE`, version 2g6 being my "go to" application when I only require text-based output. `NGSPICE` (version 26), which is based on Berkeley `SPICE` version 3f5, is used when I require graphical output for such things as time-domain waveforms and Bode plots. In all `SPICE` example netlists I strive to use coding conventions compatible with all `SPICE` versions.

Andrew D. Hwang's `ePiX` mathematical visualization programming library

This amazing project is a `C++` library you may link to any `C/C++` code for the purpose of generating PostScript graphic images of mathematical functions. As a completely free and open-source project, it does all the plotting I would otherwise use a Computer Algebra System (CAS) such as `Mathematica` or `Maple` to do. It should be said that `ePiX` is *not* a Computer Algebra System like `Mathematica` or `Maple`, but merely a mathematical *visualization* tool. In other words, it won't determine integrals for you (you'll have to implement that in your own `C/C++` code!), but it can graph the results, and it does so beautifully. What I really admire about `ePiX` is that it is a `C++` programming library, which means it builds on the existing power and toolset available with that programming language. Mr. Hwang could have probably developed his own stand-alone application for mathematical plotting, but by creating a `C++` library to do the same thing he accomplished something much greater.

`gnuplot` mathematical visualization software

Another open-source tool for mathematical visualization is `gnuplot`. Interestingly, this tool is *not* part of Richard Stallman's GNU project, its name being a coincidence. For this reason the authors prefer "gnu" *not* be capitalized at all to avoid confusion. This is a much "lighter-weight" alternative to a spreadsheet for plotting tabular data, and the fact that it easily outputs directly to an X11 console or a file in a number of different graphical formats (including PostScript) is very helpful. I typically set my `gnuplot` output format to default (X11 on my Linux PC) for quick viewing while I'm developing a visualization, then switch to PostScript file export once the visual is ready to include in the document(s) I'm writing. As with my use of `Gimp` to do rudimentary image editing, my use of `gnuplot` only scratches the surface of its capabilities, but the important points are that it's *free* and that it *works well*.

`Python` programming language

Both Python and C++ find extensive use in these modules as instructional aids and exercises, but I'm listing Python here as a *tool* for myself because I use it almost daily as a *calculator*. If you open a Python interpreter console and type `from math import *` you can type mathematical expressions and have it return results just as you would on a hand calculator. Complex-number (i.e. *phasor*) arithmetic is similarly supported if you include the complex-math library (`from cmath import *`). Examples of this are shown in the Programming References chapter (if included) in each module. Of course, being a fully-featured programming language, Python also supports conditionals, loops, and other structures useful for calculation of quantities. Also, running in a console environment where all entries and returned values show as text in a chronologically-ordered list makes it easy to copy-and-paste those calculations to document exactly how they were performed.

Appendix D

Creative Commons License

Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or

limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part; and

B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures.

For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

A. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors,

whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully

be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at `creativecommons.org/policies`, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at `creativecommons.org`.

Appendix E

Version history

This is a list showing all significant additions, corrections, and other edits made to this learning module. Each entry is referenced by calendar date in reverse chronological order (newest version first), which appears on the front cover of every learning module for easy reference. Any contributors to this open-source document are listed here as well.

28-29 January 2026 – updated some of the text to reflect the fact that not all of the PLCs photographed represent brands and models currently found in industry. Also added a page to the "Discrete PLC I/O" section of the Tutorial discussing conventional flow notation and why this matters for DC I/O modules.

25 January 2026 – added more content to the "Discrete PLC I/O" section of the Tutorial regarding inductive DC loads and commutating diodes.

15 December 2025 – converted Tutorial subsections into regular sections.

13 May 2025 – document first created, taking content from the legacy "Introduction to PLCs" (mod_plc) module.

Index

Absolute addressing, 62
ADC, 43
Adding quantities to a qualitative problem, 80
Allen-Bradley ControlLogix 5000 PLC, 26
Allen-Bradley PLC-5, 23
Allen-Bradley SLC 500 PLC, 24
Analog-to-digital converter, 43
Annotating diagrams, 79

Bipolar junction transistor, 33
Bit, 32
BJT, 33

Card, I/O, 13
Checking for exceptions, 80
Checking your work, 80
Code, computer, 89
Coil, immediate, 18
Consumed tag, 53, 54
Contact, dry, 35
Contact, wetted, 35
Contactor, 22
Conventional flow notation, 33
Current notation, conventional, 33
Current sinking, 36, 38
Current sourcing, 36, 38

DAC, 43
de facto standard, 44
de jure standard, 44
Digital-to-analog converter, 43
Dimensional analysis, 79
Direct addressing, 62
Dry contact, 35

Edwards, Tim, 90

GE Series One PLC, 29
Graph values to solve a problem, 80
Greenleaf, Cynthia, 65

Hot-swappable PLC I/O, 30
How to teach with these modules, 87
Hwang, Andrew D., 91

I/O, 13, 21, 30
I/O, analog (PLC), 43
I/O, discrete (PLC), 32
I/O, hot-swappable, 30
I/O, network (PLC), 44
I/O, remote, 31
Identify given data, 79
Identify relevant principles, 79
Immediate coil, 18
Inductance, 39
Intermediate results, 79
Inverted instruction, 87

Kickback, inductive, 39
Knuth, Donald, 90
Koyo CLICK PLC, 27
Koyo DL06 PLC, 28

Ladder Diagram, 3, 13
Lamport, Leslie, 90
Limiting cases, 80

Memory map, 58
Metacognition, 70
Modbus, 44
Moolenaar, Bram, 89
Murphy, Lynn, 65

One-shot, 21
Open-source, 89

PLC, 3, 13
PLC scan cycle, 18
Problem-solving: annotate diagrams, 79
Problem-solving: check for exceptions, 80
Problem-solving: checking work, 80
Problem-solving: dimensional analysis, 79
Problem-solving: graph values, 80
Problem-solving: identify given data, 79
Problem-solving: identify relevant principles, 79
Problem-solving: interpret intermediate results, 79
Problem-solving: limiting cases, 80
Problem-solving: qualitative to quantitative, 80
Problem-solving: quantitative to qualitative, 80
Problem-solving: reductio ad absurdum, 80
Problem-solving: simplify the system, 79
Problem-solving: thought experiment, 79
Problem-solving: track units of measurement, 79
Problem-solving: visually represent the system, 79
Problem-solving: work in reverse, 80
Produced tag, 53, 54
Profibus, 44
Programmable Logic Controller, 3, 13

Qualitatively approaching a quantitative problem, 80

Reading Apprenticeship, 65
Reductio ad absurdum, 80, 86, 87
Relay ladder logic, 13
Remote PLC I/O, 31
Requested Packet Interval, 53, 54
Rockwell ControlLogix 5000 PLC, 26
Rockwell PLC-5, 23
Rockwell SLC 500 PLC, 24
RPI, 53, 54

Scan cycle, PLC, 18
Schoenbach, Ruth, 65
Scientific method, 70
Siemens 505 PLC, 22
Siemens S7-300 PLC, 26
Simplifying a system, 79
Sinking current, 36, 38
Socrates, 86
Socratic dialogue, 87
Sourcing current, 36, 38
SPICE, 65
Stallman, Richard, 89
Standard, de facto, 44
Standard, de jure, 44
Symbol, 63
Symbolic addressing, 63

Tag name, 63
Tag, consumed, 53, 54
Tag, produced, 53, 54
Texas Instruments 505 PLC, 22
Thought experiment, 79
Torvalds, Linus, 89
Transistor, 33
TRIAC, 34

Units of measurement, 79

Visualizing a system, 79

Wetted contact, 35
Work in reverse to solve a problem, 80
WYSIWYG, 89, 90