**CONTENT**

|  |  |
| --- | --- |
| **TITLE** | **PAGE NO.** |
| INTRODUCTION | 1-4 |
| PLCs | 5-6 |
| History of PLC | 7 |
| PLC Hardware | 8-9 |
| PLC Input Output Modules | 10-11 |
| [PLC Operation Sequence](https://instrumentationforum.com/t/plc-operation-sequence/4553) | 12 |
| PLC Programming Languages | 13-16 |
| Ladder logic | 17-22 |
| Ladder logic for basic gates | 23-28 |
| Timer | 29-32 |
| Counters | 33-37 |
| PLC compared with other control systems | 38-39 |
| 1. Advantages and disadvantages of PLC | 40 |
| 1. PLC: Industrial Applications of Programmable Logic Controller | 41 |
| 1. PLC Program Used Examples | 42-49 |

# **INTRODUCTION**

# **1-The PLC system.**

A programmable logic controller consists of the following components:

Central Processing Unit (CPU)

Memory

Input modules

Output modules and

Power supply.

A PLC hardware block diagram is shown in Figure 1.1. The programming terminal in the diagram is not a part of the PLC, but it is essential to have a terminal for programming or monitoring a PLC. In the diagram, the arrows between blocks indicate the information and power flowing directions.

Programming

Terminal

CPU

Output

Module

Memory

Power Supply

Input

Module

Figure 1.1 PLC Hardware Block Diagram

# **1.1 CPU**

Like other computerized devices, there is a Central Processing Unit (CPU) in a PLC. The CPU, which is the “brain” of a PLC, does the following operations:

* Updating inputs and outputs. This function allows a PLC to read the status of its input terminals and energize or de energize its output terminals.
* Performing logic and arithmetic operations. A CPU conducts all the mathematic and logic operations involved in a PLC.
* Communicating with memory. The PLC’s programs and data are stored in memory. When a PLC is operating, its CPU may read or change the contents of memory locations.
* Scanning application programs. An application program, which is called a *ladder logic program*, is a set of instructions written by a PLC programmer. The scanning function allows the PLC to execute the application program as specified by the programmer.
* Communicating with a programming terminal. The CPU transfers program and data between itself and the programming terminal.

A PLC’s CPU is controlled by operating system software. The operating system software is a group of supervisory programs that are loaded and stored permanently in the PLC’s memory by the PLC manufacturer.

# **1.2 Memory**

Memory is the component that stores information, programs, and data in a PLC. The process of putting new information into a memory location is called *writing*. The process of retrieving information from a memory location is called *reading*.

The common types of memory used in PLCs are Read Only Memory (ROM) and Random Access Memory (RAM). A ROM location can be read, but not written. ROM is used to store programs and data that should not be altered. For example, the PLC’s operating programs are stored in ROM.

A RAM location can be read or written. This means the information stored in a RAM location can be retrieved and/or altered. Ladder logic programs are stored in RAM. When a new ladder logic program is loaded into a PLC’s memory, the old program that was stored in the same locations is over-written and essentially erased.

The memory capacities of PLCs vary. Memory capacities are often expressed in terms of *kilo-bytes* (K). One byte is a group of 8 bits. One bit is a memory location that may store one binary number that has the value of either 1 or 0. (Binary numbers are addressed in Module 2). 1K memory means that there are 1024 bytes of RAM. 16K memory means there are 16 x 1024 =16384 bytes of RAM.

# **1.3 Input modules and output modules**

A PLC is a control device. It takes information from inputs and makes decisions to energize or de-energize outputs. The decisions are made based on the statuses of inputs and outputs and the ladder logic program that is being executed.

The input devices used with a PLC include pushbuttons, limit switches, relay contacts, photo sensors, proximity switches, temperature sensors, and the like. These input devices can be AC (alternating current) or DC (direct current). The input voltages can be high or low. The input signals can be digital or analog.

Differing inputs require different input modules. An input module provides an interface between input devices and a PLC’s CPU, which uses only a low DC voltage. The input module’s function is to convert the input signals to DC voltages that are acceptable to the CPU. Standard discrete input modules include 24 V AC, 48 V AC, 120 V AC, 220 V AC, 24 V DC, 48 V DC, 120 V DC, 220 V DC, and transistor-transistor logic (TTL) level.

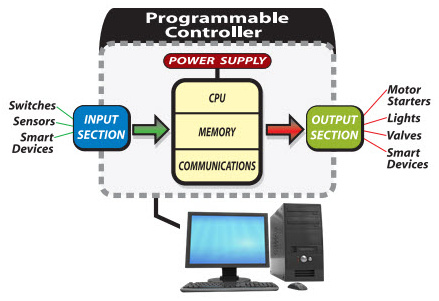
The devices controlled by a PLC include relays, alarms, solenoids, fans, lights, and motor starters. These devices may require different levels of AC or DC voltages. Since the signals processed in a PLC are low DC voltages, it is the function of the output module to convert PLC control signals to the voltages required by the controlled circuits or devices. Standard discrete output modules include 24 V AC, 48 V AC, 120 V AC, 220 V AC, 24 V DC, 48 V DC, 120 V DC, 220 V DC, and TTL level.

# **1.4 Power Supply**

PLCs are powered by standard commercial AC power lines. However, many PLC components, such as the CPU and memory, utilize 5 volts or another level of DC power. The PLC power supply converts AC power into DC power to support those components of the PLC.

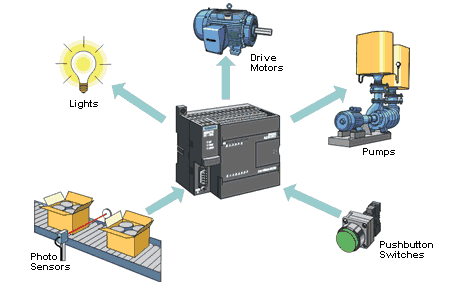
# **1.5 Programming Terminal**

A PLC requires a programming terminal and programming software for operation. The programming terminal can be a dedicated terminal or a generic computer purchased anywhere. The programming terminal is used for programming the PLC and monitoring the PLC’s operation. It may also download a ladder logic program (the sending of a program from the programming terminal to the PLC) or upload a ladder logic program (the sending of a program from the PLC to the programming terminal). The terminal uses programming software for programming and “talking” to a PLC.



# **PLCs**

A programmable logic controller (PLC), also referred to as a programmable controller, is the name given to a type of computer commonly used in commercial and industrial control applications. PLCs differ from office computers in the types of tasks that they perform and the hardware and software they require to perform these tasks. While the specific applications vary widely, all PLCs monitor inputs and other variable values, make decisions based on a stored program, and control outputs to automate a process or machine. This course is meant to supply you with basic information on the functions and configurations of PLCs with emphasis on the S7-200 PLC family.



**1.1 Basic PLC Operation**

The basic elements of a PLC include input modules or points, a central processing unit (CPU), output modules or points, and a programming device. The type of input modules or points used by a PLC depends upon the types of input devices used. Some input modules or points respond to digital inputs, also called discrete inputs, which are either on or off. Other modules or inputs respond to analog signals. These analog signals represent machine or process conditions as a range of voltage or current values.

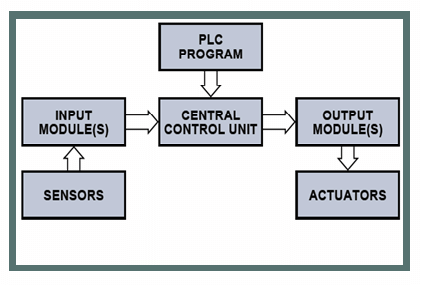
The primary function of a PLC’s input circuitry is to convert the signals provided by these various switches and sensors into logic signals that can be used by the CPU. The CPU evaluates the status of inputs, outputs, and other variables as it executes a stored program. The CPU then sends signals to update the status of outputs. Output modules convert control signals from the CPU into digital analog values that can be used to control various output devices. Output modules convert control signals from the CPU into digital or analog values that can be used to control various output devices.

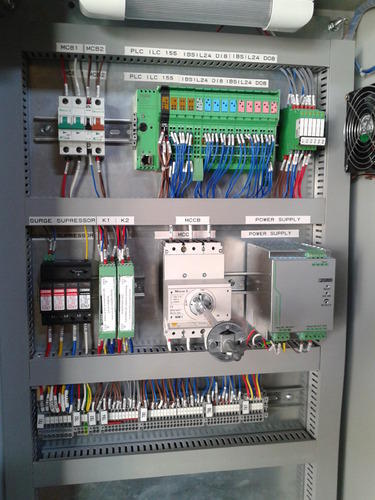
1.2 **Disadvantages of PLC control**

– Too much work required in connecting wires.  
– Difficulty with changes or replacements.  
– Difficulty in finding errors; requiring skillful work force.  
– When a problem occurs, hold-up time is indefinite, usually long.

**1.3 Advantages of PLC control**

\* Rugged and designed to withstand vibrations, temperature, humidity, and noise.  
\* Have interfacing for inputs and outputs already inside the controller.  
\* Easily programmed and have an easily understood programming language.





# **History of PLC**

 PLC development began in 1968 in response to a request from an US car manufacturer (GE). The first PLCs were installed in industry in 1969. Communications abilities began to appear in approximately 1973. They could also be used in the 70’s to send and receive varying voltages to allow them to enter the analog world. standardize communications with manufacturing automation protocol (MAP), reduce the size of the PLC, and making them software programmable through symbolic programming on personal computers instead of dedicated programming terminals or handheld programmers.

The 90’s have seen a gradual reduction in the introduction of new protocols, and the modernization of the physical layers of some of the more popular protocols that survived the 1980’s.The latest standard “IEC 1131-3” has tried to merge plc programming languages under one international standard. We now have PLCs that are programmable in function block diagrams, instruction lists, C and structured text all at the same time.Before the days of the PLC the only way to control machinery was through the use of relays.

Relays work by utilizing a coil that, when energized, creates a magnetic force to effectively pull a switch to the ON or OFF position. When the relay is de-energized, the switch releases and returns the device to its standard ON or OFF position. So, for example, if I wanted to control whether a motor was ON or OFF, I could attach a relay between the power source and the motor. Then I could control when the motor is getting power by either energizing or de-energizing the relay.

These relays are known as control relays because they control the relays that control the switch that turns the motor ON and OFF. I could keep going, but I think you get the picture of how machines were controlled pre-PLC, and, more importantly, I think you start to see some of the problems with this system of electromechanical control via relays.



# **PLC Hardware**

**1- Hardware Components of a PLC System**

Processor unit (CPU), Memory, Input/Output, Power supply unit, Programming device, and other devices.



**1.1 Central Processing Unit (CPU)**

CPU – Microprocessor based, may allow arithmetic operations, logic operators, block memory moves, computer interface, local area network, functions, etc.  
CPU makes a great number of check-ups of the PLC controller itself so eventual errors would be discovered early.

**1.2 System Busses**

The internal paths along which the digital signals flow within the PLC are called  
busses.  
The system has four busses:  
– The CPU uses the data bus for sending data between the different elements,  
– The address bus to send the addresses of locations for accessing stored data,  
– The control bus for signals relating to internal control actions,  
– The system bus is used for communications between the I/O ports and the I/O unit.

**1.3 Memory**

System (ROM) to give permanent storage for the operating system and the fixed data used by the CPU.  
RAM for data. This is where information is stored on the status of input and output devices and the values of timers and counters and other internal devices. EPROM for ROM’s that can be programmed and then the program made permanent.

**1.4 I/O Sections**

Inputs monitor field devices, such as switches and sensors.  
Outputs control other devices, such as motors, pumps, solenoid valves, and lights.

**1.5 Power Supply**

Most PLC controllers work either at 24 VDC or 220 VAC. Some PLC controllers have electrical supply as a separate module, while small and medium series already contain the supply module.

**1.6 Programming Device**

The programming device is used to enter the required program into the memory of the processor.  
The program is developed in the programming device and then transferred to the  memory unit of the PLC.

# **1.7 Software**

PLC consists of two parts: Operating systems and user program. The PLC operating system  
provides effective support ranging from the creation of project structure to the creation of user  
programs. The OS system is accessed through a graphical user interface window ( also known as  
Main window). The main window contains all the functions needed to set up a project , configure  
the hardware , write and test programs. User program can be written in any standard PLC  
programming language like ladder diagram or statement list.

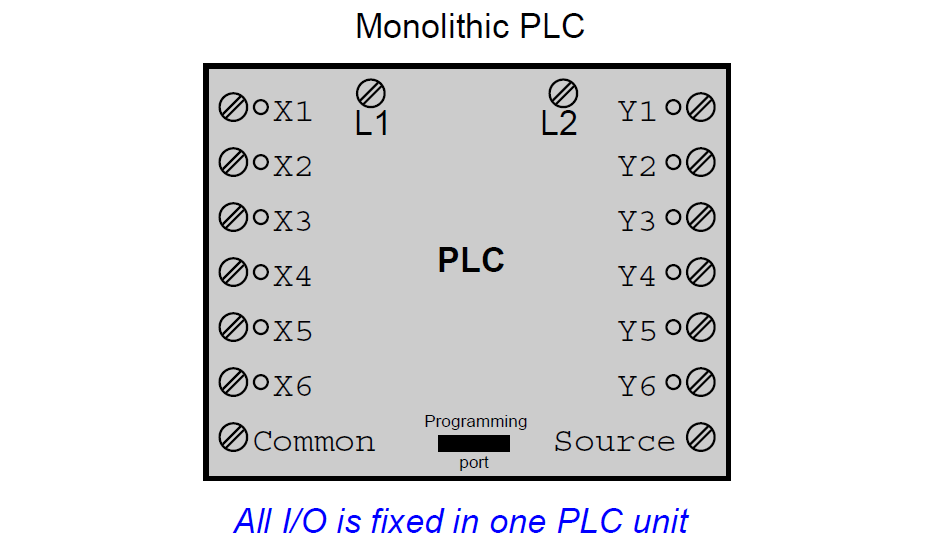
While processing a PLC program, the CPU scans and executes the main program cyclically;A  
program scan cycle consists of sequential operations that include input scan, program scan, and  
output scan. In the input scan, the CPU updates the process image input table, in the output scan;  
the CPU updates the process image output table.

After the completion of each scan cycle, the CPU returns to the beginning of the next cycle and again repeats the cycle. The time taken to scan one program is called scan –cycle time.

# **PLC Input Output Modules**

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](https://instrumentationtools.com/solenoid-valves-energized-or-de-energized-state/), [valves](https://instrumentationtools.com/shutdown-valve/), and [motors](https://instrumentationtools.com/synchronous-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](https://instrumentationtools.com/4-20ma-transmitter-works/) and similar signals.

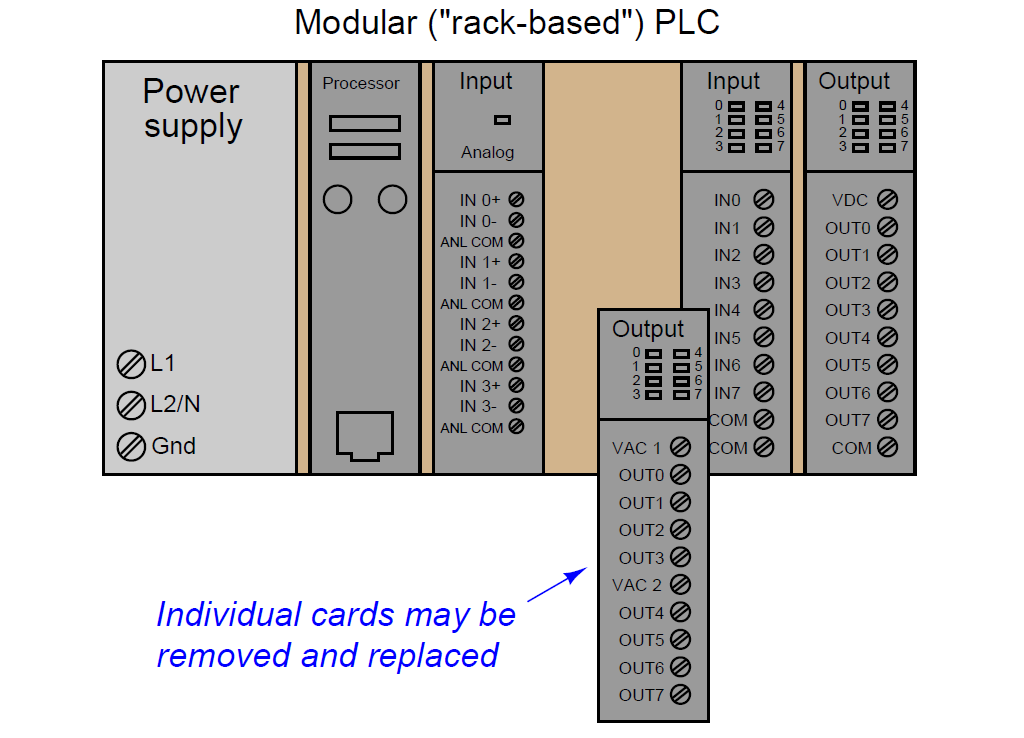
Some PLCs even offer the feature of hot-swappable cards, meaning each card may be removed and a new one inserted without de-energizing power to the PLC processor and rack.

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!

An alternative scheme for system expansion is to network multiple PLCs together, where each PLC has its own dedicated rack and processor.

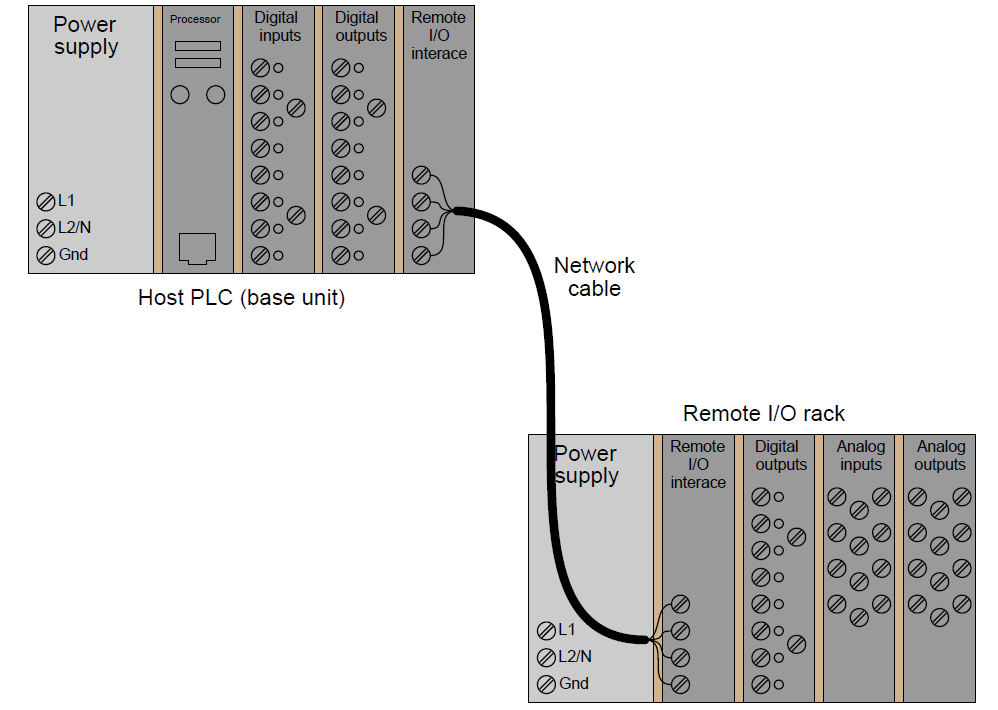
Through the use of communication instructions, one [PLC](https://instrumentationforum.com/t/valve-position-switch-wiring-in-plc/5744) may be programmed to read data from and/or write data to another PLC, effectively using the other PLC as an extension of its own I/O.

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.



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:



# [**PLC Operation Sequence**](https://instrumentationforum.com/t/plc-operation-sequence/4553)

All [PLCs 2](https://instrumentationtools.com/category/plc/) have four basic stages of operations that are repeated many times per second. Initially when turned on the first time it will check its own hardware and software for faults. If there are no problems it will copy all the input and copy their values into memory, this is called the input scan. Using only the memory copy of the inputs the ladder logic program will be solved once, this is called the logic scan.

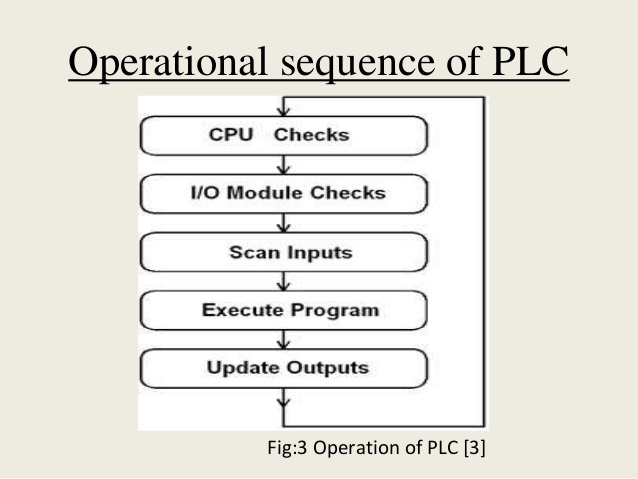
**1. Self test –** Checks to see if all cards error free, reset watch-dog timer, etc. (A watchdog timer will cause an error, and shut down the PLC if not reset within a short period of time – this would indicate that the ladder logic is not being scanned normally).

**2. Input scan –** Reads input values from the chips in the input cards, and copies their values to memory. This makes the PLC operation faster, and avoids cases where an input changes from the start to the end of the program (e.g., an emergency stop). There are special PLC functions that read the inputs directly, and avoid the input tables.

**3. Logic solve/scan –** Based on the input table in memory, the program is executed 1 step at a time, and outputs are updated. This is the focus of the later sections.

**4. Output scan –** The output table is copied from memory to the output chips. These chips then drive the output devices.

The input and output scans often confuse the beginner, but they are important. The input scan takes a snapshot of the inputs, and solves the logic. This prevents potential problems that might occur if an input that is used in multiple places in the ladder logic program changed while half ways through a ladder scan and thus changing the behaviors of half of the ladder logic program. This problem could have severe effects on complex programs. One side effect of the input scan is that if a change in input is too short in duration, it might fall between input scans and be missed.



# **PLC Programming Languages**

The 5 most popular PLC Programming Languages are:

1. Ladder Diagram (LD)

2. Sequential Function Charts (SFC)

3. Function Block Diagram (FBD)

4. Structured Text (ST)

5. Instruction List (IL)

**1. Ladder Diagram (LD)**

Ladder Diagram was originally modeled from relay-logic which used physical devices, such as switches and mechanical relays to control processes. Ladder Diagram utilizes internal logic to replace all, except the physical devices that need an electrical signal to activate them.

Ladder Diagram is built in the form of horizontal rungs with two vertical rails that represent the electrical connection on relay-logic schematics. You can program all the necessary input conditions to affect the output conditions, whether logical or physical.

**1.1 Ladder Diagram Advantages**

The main advantages of the Ladder Diagram language are:

1. The rungs allow it to be organized and easy to follow.

2. It also lets you document comments that are readily visible.

3. It supports online editing very successfully.

**1.2 Ladder Diagram Disadvantages**

The main disadvantage is that there are some instructions that are not available, which might make it more difficult for programming such as motion or batching.

The next PLC Programming Language that I will talk to you about is the Sequential Function Charts which uses a graphical type of programming.

**2. Sequential Function Charts (SFC)**

If you have any experience with flowcharts, then this PLC Programming language will feel familiar to you. In Sequential Function Charts, you use steps and transitions to achieve your end results.

Steps act as a major function in your program. These steps house the actions that occur when you program them to happen. This decision can be based on timing, a certain phase of the process, or a physical state of an equipment.

Transitions are the instructions that you use to move from one step to another step by setting conditions of true or false.

Unlike traditional flowcharts, the Sequential Function Charts can have multiple paths. You can use branches to initiate multiple steps at one time.

**2.1 Sequential Function Charts Advantages**

A couple of the advantages of Sequential Function Charts are:

1. Processes can be broken into major steps that can make troubleshooting faster and easier.

2.You have direct access in the logic to see where a piece of equipment faulted.

3. It can be faster to design and write the logic due to the ability to use repeated executions of individual pieces of logic.

**2.2 Sequential Function Charts Disadvantages**

Even when you consider the advantages of the Sequential Function Charts, this PLC Programming Language does not always fit every application.

Now we are on to our third PLC Programming Language.

**3. Function Block Diagram (FBD)**

The Function Block Diagram which is also a graphical type of language. The Function Block Diagram describes a function between inputs and outputs that are connected in blocks by connection lines.

Function Blocks were originally developed to create a system that you could set up many of the common, repeatable tasks, such as counters, timers, PID Loops, etc.

You program the blocks onto sheets and then the PLC constantly scans the sheets in numerical order or is determined by connections which you program between the blocks.

**3.1 Function Block Diagram Disadvantages**

The code can get disorganized using this PLC Programming Language because you can place the function blocks anywhere on the sheet. This can also make it more difficult to troubleshoot.

**3.2 Function Block Diagram Advantages**

1. The Function Block Diagram does work well with motion controls.

2. The visual method is easier for some users.

3. The biggest advantage of Function Block Diagram is that you can take many lines of programming and put it into one or several function blocks.

**4. Structured Text (ST)**

The 4th PLC Programming Language is the Structured Text. This language is a textual based language.

Structured Text is a high-level language that is like Basic, Pascal and “C”.

It is a very powerful tool that can execute complex tasks utilizing algorithms and mathematical functions along with repetitive tasks.

The code uses statements that are separated by semicolons and then either inputs, outputs, or variables are changed by these statements.

You must write out each line of code and it uses functions such as FOR, WHILE, IF, ELSE, ELSEIF AND CASE.

#### 4.1 Structured Text Advantages

Some of the advantages of Structured Text are:

1. It is very organized and good at computing large mathematical calculations.

2. It will enable you to cover some instructions that are not available in some other languages like the Ladder Diagram.

#### 4.2 Structured Text Disadvantages

The disadvantages of the Structured Text PLC programming language are:

1. The syntax can be difficult.

2. It is hard to debug.

3. It is difficult to edit online.

### 5. Instruction List (IL)

I will now show you the 5th and final PLC Programming Language which is Instruction List. The Instruction List is also a textual based language.

The Instruction List language resembles Assembly Language. When you use this PLC Programming Language, you will use mnemonic codes such as LD (Load), AND, OR, etc.

The Instruction List contains instructions with each instruction on a new line with any comments you might want to annotate at the end of each line.

#### 5.1 Instruction List Advantages

The Instruction List language is valuable for applications that need code that is compact and time critical.

**5.2 Instruction List Disadvantages**

The main disadvantages of this PLC Programming Language are:

1. There are few structuring possibilities with the “Goto” command being one of them.

2. There can also be many errors that are more difficult to deal with in comparison to many of the other languages that I have previously reviewed.

So, have you decided which PLC Programming Language you consider to be the most popular?

After reading many reviews and opinions and with my own experiences, the Ladder Diagram is by far the most popular PLC programing language .

The main reason for this is that the Ladder Diagram language naturally followed the technology advancement from a physical relay logic to a digital and logical one. This allowed the engineers and skilled workers to follow and troubleshoot and make that transition.

In summary, there is certainly a place for all the PLC Programming Languages that we have reviewed. Your background, experience and the application you are working with are really going to be the key to which PLC Programming Language you choose.

# **Ladder logic**

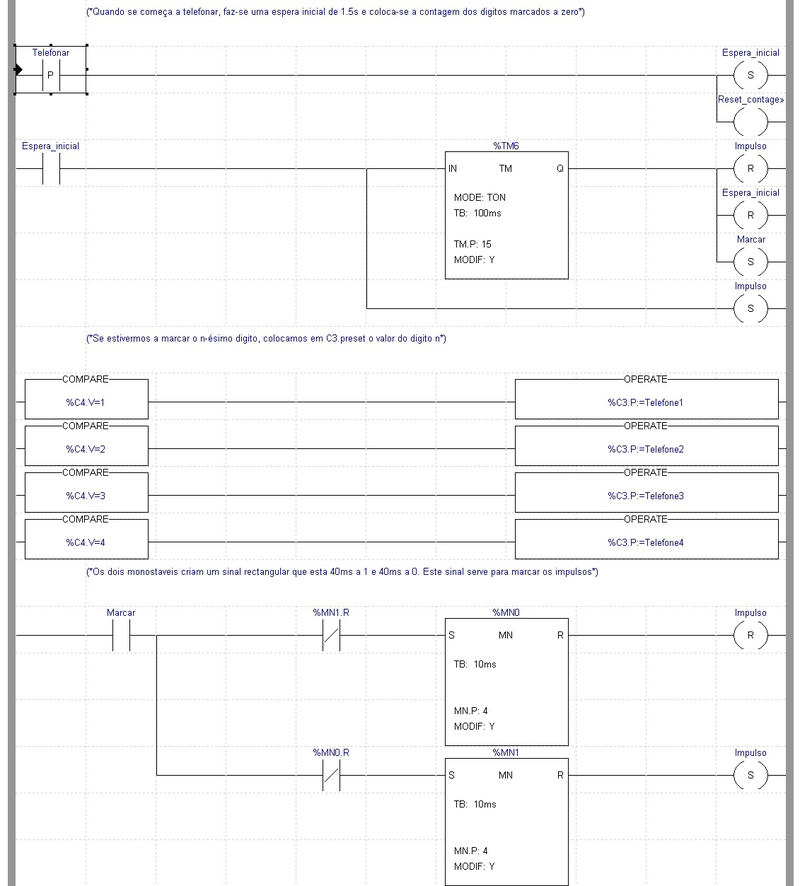
**Ladder logic** was originally a written method to document the design and construction of [relay racks](https://en.wikipedia.org/wiki/Relay_logic) as used in manufacturing and [process control](https://en.wikipedia.org/wiki/Process_control).[[1]](https://en.wikipedia.org/wiki/Ladder_logic#cite_note-1) Each device in the relay rack would be represented by a symbol on the ladder diagram with connections between those devices shown. In addition, other items external to the relay rack such as pumps, heaters, and so forth would also be shown on the ladder diagram.

Ladder logic has evolved into a [programming language](https://en.wikipedia.org/wiki/Programming_language) that represents a program by a graphical diagram based on the [circuit diagrams](https://en.wikipedia.org/wiki/Circuit_diagram) of [relay logic](https://en.wikipedia.org/wiki/Relay_logic) hardware. Ladder logic is used to develop software for [programmable logic controllers](https://en.wikipedia.org/wiki/Programmable_logic_controller) (PLCs) used in industrial control applications. The name is based on the observation that programs in this language resemble [ladders](https://en.wikipedia.org/wiki/Ladder), with two vertical rails and a series of horizontal rungs between them. While ladder diagrams were once the only available notation for recording programmable controller programs, today other forms are standardized in [IEC 61131-3](https://en.wikipedia.org/wiki/IEC_61131-3) (For example, as an alternative to the graphical ladder logic form, there is also a more assembly language like format called [Instruction list](https://en.wikipedia.org/wiki/Instruction_list) within the [IEC 61131-3](https://en.wikipedia.org/wiki/IEC_61131-3) standard.).

Ladder logic is widely used to program [PLCs](https://en.wikipedia.org/wiki/Programmable_logic_controller), where sequential control of a process or manufacturing operation is required. Ladder logic is useful for simple but critical control systems or for reworking old [hardwired](https://en.wikipedia.org/wiki/Electrical_wiring) relay circuits. As programmable logic controllers became more sophisticated it has also been used in very complex automation systems. Often the ladder logic program is used in conjunction with an [HMI](https://en.wikipedia.org/wiki/User_Interface) program operating on a computer workstation.

The motivation for representing [sequential](https://en.wikipedia.org/wiki/Sequential_logic) [control logic](https://en.wikipedia.org/wiki/Control_logic) in a ladder diagram was to allow factory engineers and technicians to develop software without additional training to learn a language such as [FORTRAN](https://en.wikipedia.org/wiki/FORTRAN) or other general purpose computer language. Development and maintenance were simplified because of the resemblance to familiar relay hardware systems. Implementations of ladder logic may have characteristics, such as sequential execution and support for control flow features, that make the analogy to hardware somewhat inaccurate.

Ladder logic can be thought of as a [rule-based language](https://en.wikipedia.org/wiki/Rule-based_language) rather than a [procedural language](https://en.wikipedia.org/wiki/Procedural_language). A "rung" in the ladder represents a rule. When implemented with relays and other electromechanical devices, the various rules execute simultaneously and immediately. When implemented in a programmable logic controller, the rules are typically executed sequentially by software in a continuous loop, or "scan". By executing the loop fast enough, typically many times per second, the effect of simultaneous and immediate execution is achieved. Proper use of programmable controllers requires an understanding of the limitations of the execution order of rungs.



**1-Syntax & Examples**

The language itself can be seen as a set of connections between logical checkers (contacts) and actuators (coils). If a path can be traced between the left side of the rung and the output, through asserted (true or "closed") contacts, the rung is true and the output coil storage bit is asserted (1) or true. If no path can be traced, then the output is false (0) and the "coil" by analogy to electromechanical [relays](https://en.wikipedia.org/wiki/Relay) is considered "de-energized". The analogy between logical propositions and relay contact status is due to [Claude Shannon](https://en.wikipedia.org/wiki/Claude_Shannon).

Ladder logic has contacts that make or break circuits to control coils. Each coil or contact corresponds to the status of a single bit in the programmable controller's memory. Unlike electromechanical relays, a ladder program can refer any number of times to the status of a single bit, equivalent to a relay with an indefinitely large number of contacts.

So-called "contacts" may refer to physical ("hard") inputs to the programmable controller from physical devices such as pushbuttons and [limit switches](https://en.wikipedia.org/wiki/Limit_switch) via an integrated or external input module, or may represent the status of internal storage bits which may be generated elsewhere in the program.

Each rung of ladder language typically has one coil at the far right. Some manufacturers may allow more than one output coil on a rung.

* Rung input : checkers (contacts)
  + —[ ]— Normally open contact, closed whenever its corresponding coil or an input which controls it is energized. (Open contact at rest)
  + —[\]— Normally closed ("not") contact, closed whenever its corresponding coil or an input which controls it is not energized. (Closed contact at rest)
* Rung output: actuators (coils)
  + —( )— Normally inactive coil, energized whenever its rung is closed. (Inactive at rest)
  + —(\)— Normally active ("not") coil, energized whenever its rung is open. (Active at rest)

The "coil" (output of a rung) may represent a physical output which operates some device connected to the programmable controller, or may represent an internal storage bit for use elsewhere in the program.

### 1.2 Logical AND

|  |
| --- |
| ------[ ]--------------[ ]----------------( )  Key switch 1 Key switch 2 Door motor |

The above realizes the function: Door motor = Key switch 1 [AND](https://en.wikipedia.org/wiki/Logical_AND) Key switch 2

This circuit shows two key switches that security guards might use to activate an electric motor on a bank vault door. When the normally open contacts of both switches close, electricity is able to flow to the motor which opens the door.

### 1.3 Logical AND with NOT

|  |
| --- |
| ------[ ]--------------[\]----------------( )  Close door Obstruction Door motor |

The above realizes the function: Door motor = Close door [AND](https://en.wikipedia.org/wiki/Logical_AND) [NOT](https://en.wikipedia.org/wiki/Logical_NOT)(Obstruction).

This circuit shows a push button that closes a door, and an obstruction detector that senses if something is in the way of the closing door. When the normally open push button contact closes and the normally closed obstruction detector is closed (no obstruction detected), electricity is able to flow to the motor which closes the door.

### 1.4 Logical OR

|  |
| --- |
| --+-------[ ]-------+-----------------( )  | Exterior unlock | Unlock  | |  +-------[ ]-------+  Interior unlock |

The above realizes the function: Unlock = Interior unlock [OR](https://en.wikipedia.org/wiki/Logical_OR) Exterior unlock

This circuit shows the two things that can trigger a car's [power door locks](https://en.wikipedia.org/wiki/Power_door_locks). The remote receiver is always powered. The unlock [solenoid](https://en.wikipedia.org/wiki/Solenoid) gets power when either set of contacts is closed.

### 2-Industrial STOP/START

In common industrial latching start/stop logic we have a "Start" button to turn on a motor contactor, and a "Stop" button to turn off the contactor.

When the "Start" button is pushed the input goes true, via the "Stop" button NC contact. When the "Run" input becomes true the seal-in "Run" NO contact in parallel with the "Start" NO contact will close maintaining the input logic true (latched or sealed-in). After the circuit is latched the "Stop" button may be pushed causing its NC contact to open and consequently the input to go false. The "Run" NO contact then opens and the circuit logic returns to its inactive state.

|  |
| --- |
| --+----[ ]--+----[\]----( )  | Start | Stop Run  | |  +----[ ]--+  Run  -------[ ]--------------( )  Run Motor |

The above realizes the function: Run = (Start [OR](https://en.wikipedia.org/wiki/Logical_OR) Run) [AND](https://en.wikipedia.org/wiki/Logical_AND) ([NOT](https://en.wikipedia.org/wiki/Logical_NOT) Stop)

This [latch](https://en.wikipedia.org/wiki/Latch_(electronic)) configuration is a common [idiom](https://en.wikipedia.org/wiki/Idiom) in ladder logic. It may also be referred to as "seal-in logic". The key to understanding the latch is in recognizing that the "Start" switch is a momentary switch (once the user releases the button, the switch is open again). As soon as the "Run" solenoid engages, it closes the "Run" NO contact, which latches the solenoid on. The "Start" switch opening up then has no effect.

For safety reasons, an Emergency-Stop may be hardwired in series with the Start switch, and the relay logic should reflect this.

|  |
| --- |
| --[\]----[\]----+--[ ]--+---------( )  ES Stop | Start | Run  | |  +--[ ]--+  Run  -------[ ]--------------( )  Run Motor  The above realizes the function:  Run = ([NOT](https://en.wikipedia.org/wiki/Logical_NOT) Emergency Stop) [AND](https://en.wikipedia.org/wiki/Logical_AND) ([NOT](https://en.wikipedia.org/wiki/Logical_NOT) Stop) [AND](https://en.wikipedia.org/wiki/Logical_AND) (Start [OR](https://en.wikipedia.org/wiki/Logical_OR) Run) |

### 3-Complex logic

Here is an example of what two rungs in a ladder logic program might look like. In real world applications, there may be hundreds or thousands of rungs.

Typically, complex ladder logic is 'read' left to right and top to bottom. As each of the lines (or rungs) are evaluated the output coil of a rung may feed into the next stage of the ladder as an input. In a complex system there will be many "rungs" on a ladder, which are numbered in order of evaluation.

|  |
| --- |
| 1. ----[ ]---------+----[ ]-----+----( )  Switch | HiTemp | A/C  | |  +----[ ]-----+  Humid  2. ----[ ]----[\]--------------------( )  A/C Heat Cooling |

Line 1 realizes the function: A/C = Switch [AND](https://en.wikipedia.org/wiki/Logical_AND) (HiTemp [OR](https://en.wikipedia.org/wiki/Logical_OR) Humid)

Line 2 realizes the function: Cooling = A/C [AND](https://en.wikipedia.org/wiki/Logical_AND) ([NOT](https://en.wikipedia.org/wiki/Logical_NOT) Heat)

This represents a slightly more complex system for rung 2. After the first line has been evaluated, the output coil "A/C" is fed into rung 2, which is then evaluated and the output coil "Cooling" could be fed into an output device "Compressor" or into rung 3 on the ladder. This system allows very complex logic designs to be broken down and evaluated.

**4-Additional functionality**

Additional functionality can be added to a ladder logic implementation by the PLC manufacturer as a special block. When the special block is powered, it executes code on predetermined arguments. These arguments may be displayed within the special block.

|  |
| --- |
| +-------+  -----[ ]--------------------+ A +----  Remote unlock +-------+  Remote counter  +-------+  -----[ ]--------------------+ B +----  Interior unlock +-------+  Interior counter  +--------+  --------------------+ A + B +-----------  | into C |  +--------+  Adder |

In this example, the system will count the number of times that the interior and remote unlock buttons are pressed. This information will be stored in memory locations A and B. Memory location C will hold the total number of times that the door has been unlocked electronically.

PLCs have many types of special blocks. They include timers, arithmetic operators and comparisons, table lookups, text processing, [PID](https://en.wikipedia.org/wiki/PID_controller) control, and filtering functions. More powerful PLCs can operate on a group of internal memory locations and execute an operation on a range of addresses, for example, to simulate a physical sequential drum controller or a [finite state machine](https://en.wikipedia.org/wiki/Finite_state_machine). In some cases, users can define their own special blocks, which effectively are subroutines or macros. The large library of special blocks along with high speed execution has allowed use of PLCs to implement very complex automation systems.

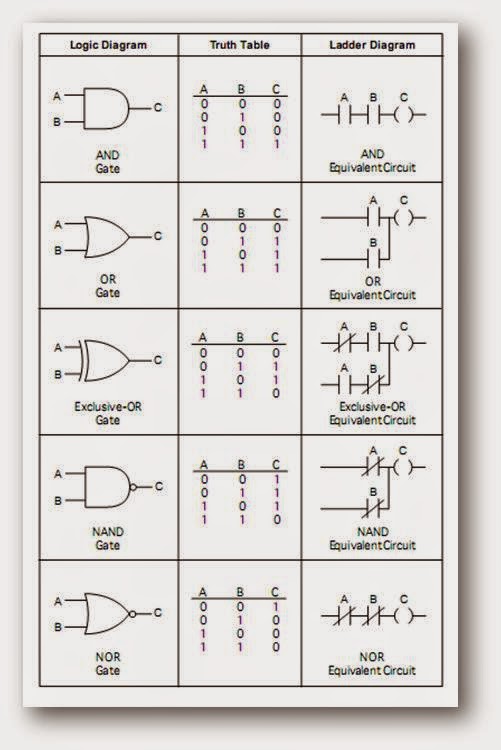
5-Limitations and successor languages

Ladder notation is best suited to control problems where only binary variables are required and where interlocking and sequencing of binary is the primary control problem. Like all [parallel programming languages](https://en.wikipedia.org/wiki/Parallel_programming_language), the sequential order of operations may be undefined or obscure; logic [race conditions](https://en.wikipedia.org/wiki/Race_condition) are possible which may produce unexpected results. Complex rungs are best broken into several simpler steps to avoid this problem. Some manufacturers avoid this problem by explicitly and completely defining the execution order of a rung, however programmers may still have problems fully grasping the resulting complex semantics.

Analog quantities and arithmetical operations are clumsy to express in ladder logic and each manufacturer has different ways of extending the notation for these problems. There is usually limited support for arrays and loops, often resulting in duplication of code to express cases which in other languages would call for use of indexed variables.

As [microprocessors](https://en.wikipedia.org/wiki/Microprocessors) have become more powerful, notations such as [sequential function charts](https://en.wikipedia.org/wiki/Sequential_function_chart) and [function block diagrams](https://en.wikipedia.org/wiki/Function_block_diagram) can replace ladder logic for some limited applications. Some newer PLCs may have all or part of the programming carried out in a dialect that resembles [BASIC](https://en.wikipedia.org/wiki/BASIC), [C](https://en.wikipedia.org/wiki/C_(programming_language)), or other [programming language](https://en.wikipedia.org/wiki/Programming_language) with bindings appropriate for a real-time application environment.

**Ladder logic for basic gates**



In the world of automation these types of TRUE or FALSE conditions come down to a device being ON or OFF, CLOSED or OPEN, PRESENT or ABSENT, 24 VOLTS or 0 VOLTS.  In the PLC it all boils down to our now familiar [binary system](http://www.plcdev.com/plc_basics/number_systems_and_codes/binary_people_learning_your_1s_and_0s) of a 1 or a 0.  Typically having a bit ON represents a TRUE condition while OFF is FALSE.  This is abitrary though as it may make more sense to use what is called failsafe logic and have an ON bit as a FALSE condition.

Let's turn again to some simple statements but this time using automation examples.

* When the button is pressed **AND**the door is closed then turn on the motor.
* If the process is done **OR**the emergency stop button signal is **NOT**on then turn off the motor.  (This is an example of a failsafe operation as the emergency stop button could be pressed or the wire has been disconnected.  In either case we want to check this for safety reasons.  Relying on a signal to turn on when a wire has fallen off long ago may cause an awkward moment when we truly have to stop the machine in an emergency.)
* If the tank is full **OR**the button is pressed **AND**there are no alarms then start the process.

It would be nice to program like this but computers like to be a little bit more structured.  A series of graphical objects have been used for years to represent these logic elements and they can be easily converted to a common ladder logic equivalent.  These functions are also called gates as they act like gate keepers for different logic.

### 1-The NOT function

The simplest of all logic functions is the NOT gate.

http://www.plcdev.com/files/plcdev/images/not.png

It's sole function in life is to invert of flip the logic state.  So an input of 1 will come out as a 0 and visa versa.  Shown below is a truth table (it doesn't lie) showing all possible inputs and the resulting logical output.

|  |  |
| --- | --- |
| **Input A** | **Output** |
| 0 | 1 |
| 1 | 0 |

The ladder logic equivalent for a NOT function looks like a normal contact but with a slash through it.

http://www.plcdev.com/files/plcdev/images/not-ladder-logic.png

### 2-The AND function

The AND gate is associated with the following symbol that can have any number of inputs but only one output.

http://www.plcdev.com/files/plcdev/images/and.png

The truth table below shows that the output is only turned on when all the inputs are true (1).  An easy way to remember this is AND works like multiplication.

|  |  |  |
| --- | --- | --- |
| **Input A** | **Input B** | **Output** |
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

The ladder logic equivalent for an AND function looks like two normal contacts side by side.

http://www.plcdev.com/files/plcdev/images/and-ladder-logic.png

### 3-The OR function

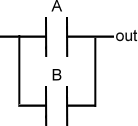
Last but not least the OR gate is associated with the following symbol that also can have any number of inputs but only one output.

http://www.plcdev.com/files/plcdev/images/or.png

The truth table below shows that the output is turned on (1) when any of the inputs are true (1).  An easy way to remember this is OR works like addition.

|  |  |  |
| --- | --- | --- |
| **Input A** | **Input B** | **Output** |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

The ladder logic equivalent for an OR function looks like two normal contacts on top of each other.



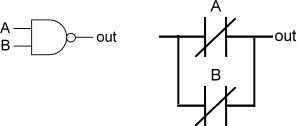
### 4-Combining AND or OR with NOT

The NOT gate might not look like much help if you haven't programmed much but you'll find yourself actually using it frequently.  It's very common to use it in combination with AND and OR.  So the engineering gods decided to make some symbols for these combinations.

Putting the NOT and AND gates together forms the NAND gate.  The truth table below shows that it is simply an inverted output of the AND gate.

|  |  |  |
| --- | --- | --- |
| **Input A** | **Input B** | **Output** |
| 0 | 0 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

A little circle (or if you like, a bubble) at the end of a AND gate is used to signify the NAND function.  It's symbol and corresponding ladder logic are shown below.  Now pay close attention to the ladder logic because the contacts are in parallel and not in series like the AND function.



Putting the NOT and OR gates together forms... you got it... the NOR gate.  The truth table below shows that it is simply an inverted output of the OR gate.

|  |  |  |
| --- | --- | --- |
| **Input A** | **Input B** | **Output** |
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 0 |

Again a little circle is placed at the end of an OR gate to signify the NOR function.  It's symbol and corresponding ladder logic are shown below.  The ladder logic is very different from the regular OR gate.

http://www.plcdev.com/files/plcdev/images/nor-plus-ladder-logic.png

### But wait!  Don't order yet... the XOR gate.

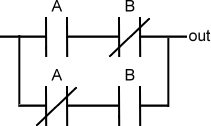
So far with our logic gates we've covered almost all possible combinations except for one shown by the truth table below.

|  |  |  |
| --- | --- | --- |
| **Input A** | **Input B** | **Output** |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

 The logic to produce this output is called an Exclusive OR gate otherwise known as the XOR gate.  It's a specialized form of the OR gate.  So if either one of the inputs are on then the output is true, otherwise you're out of luck.  The symbol for the XOR gate is shown by added a curved line to the OR gate symbol.

http://www.plcdev.com/files/plcdev/images/xor.png

The ladder logic to implement an XOR gate is a little more complex then the others.

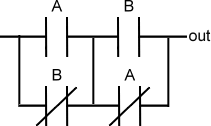


How useful is the XOR logic?  You probably use the XOR gate everyday without thinking about it if you have a room with a light that works off two switches.  If both switches are in the same position then the light will be off.  Therefore just flipping one switch will turn the light on.  In the PLC program this can be extremely useful for programming alternating actions or [gray codes](http://www.plcdev.com/using_ladder_logic_for_gray_code_conversion).

Ok, there is one more logic gate but I promise it is the last one.  It makes sense that there is a XNOR gate which is the combination of the NOT and XOR logic.  It simply inverts the output of the XOR function.

|  |  |  |
| --- | --- | --- |
| **Input A** | **Input B** | **Output** |
| 0 | 0 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

The symbol for the XNOR gate is shown below along with it's ladder logic equivalent.

http://www.plcdev.com/files/plcdev/images/xnor.png 

# **Timer**

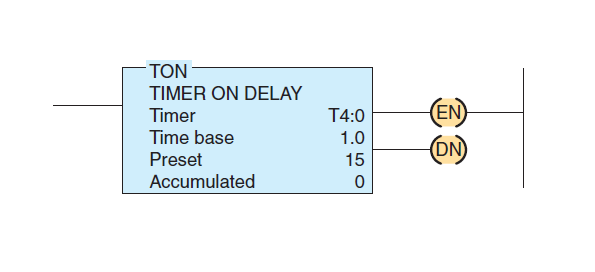
PLC timers are instructions that provide the same functions as on-delay and off-delay mechanical and electronic timing relays. A PLC timer provides a preset delay to the control actions.In general, there are three types of PLC timer delays, ON-delay timer, OFF-delay timer and retentive timer on.

The terms represented in the timer block in the PLC are a Preset value which means the delay period of the timer, an Accumulated value which is the current delay of the timer.

A timer begins the counting on time-based intervals and continues until the accumulated value equals the preset value. When the accumulated value equals the preset time the output will be energized. Then the timer sets the output.

**1.TON timer or ON delay timer**

An ON delay timer is used where we need a time delay before the time delay before an instruction becomes true.



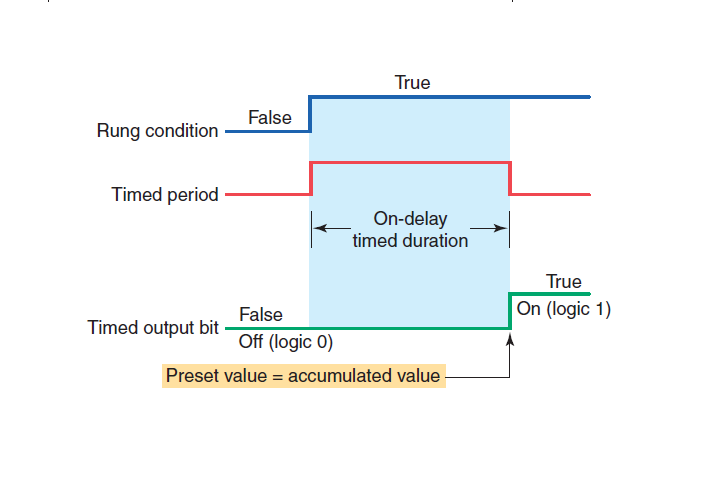
A representation TON timer is shown above, which contains,

*Timer number:* The timer file name

*Time base:* which is shown in seconds

*Preset value:* Numeric valve set as the delay required to the timer.

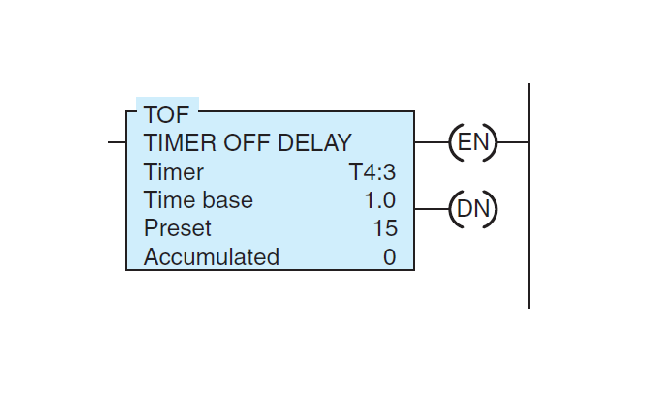
*Accumalated value:* The values are counting is displayed from zero. Value becomes zero whenever the timer is reset



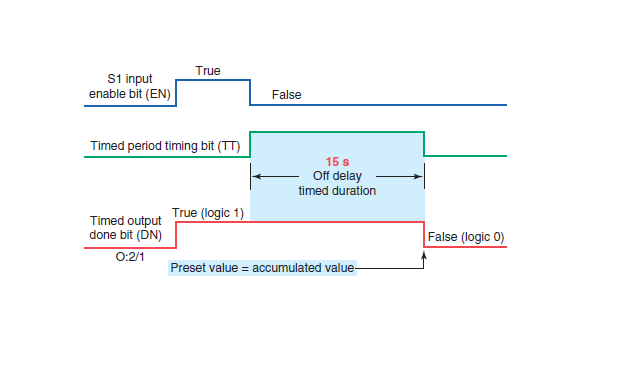
* The timer starts operating when the rung condition becomes true. The timer delay starts counting when the rung condition starts to accumulate.
* When the Preset value becomes equal to the accumulated value, the output is made true.
* The timed output becomes true sometime after the timer rung becomes true; hence, the timer is said to have an on-delay.
* The length of the delay can be adjusted by setting the preset value.

**2.TOFF timer or OFF delay timer:**

A TOFF timer will keep the output energized for a preset time after the rung signal has gone false.



The TOFF timer will have all the contents as in the TON timer, with the similar function.



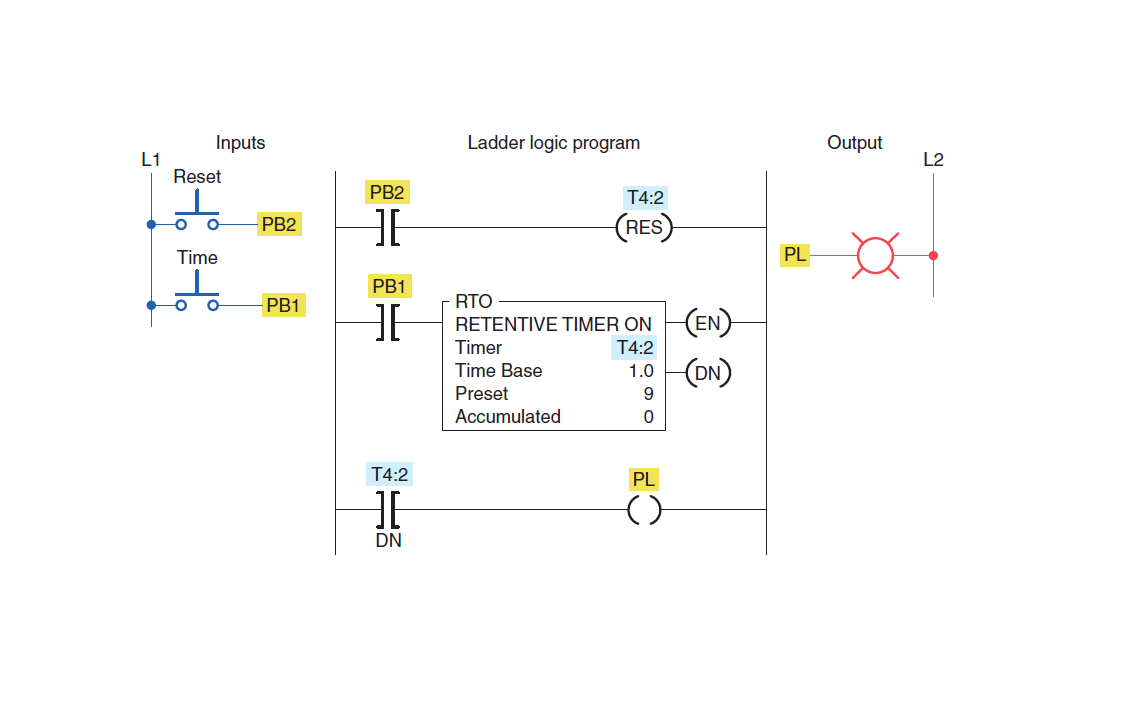
* When the rung timer is true, the output will be true without any delay. When the rung signal becomes false the timer starts operating.
* The timer starts accumulating times when the rung condition becomes true, until the accumulated value becomes equal to the Preset value.
* The output turns off when the output will turn false when the accumulated value equals the preset value.

**3.Retentive timer:**

A retentive timer is used when you want to retain accumulated time value through the power loss or the change in the rung state.

A retentive timer accumulates time whenever the device receives power, and it maintains the current time should power be removed from the device

Loss of power to the timer after reaching its preset value does not affect the state of the contacts. The retentive timer must be intentionally reset with a separate signal for the accumulated time to be reset



An example logic is shown above.

* When the push bottom PB1 is pressed, the timer starts working and the reading starts accumulating.
* When we push the PB1 button then rung become false and the timer stops working. Consider we are switching ON the push after a time, the timer starts counting from the previous value before the timer is stopped.
* We have to add another switch PB2 to reset the values in the timer

# **Counters**

A PLC counter is a function block that counts up or down until it reaches a limit. When the limit is reached the output is set.

The thing is that counting is in fact widely used in PLC programming. Often you will have the need to counts different things. An example of this could be to keep track of how many times a process has been completed. Or how many products has been produced.

PLC counters are also used to assist logging to [SCADA systems](https://www.plcacademy.com/scada-system/) by counting the amount of times these events has happened or setting alarms when an event has happened a certain amount of times.

With all that said timers are very useful and it is crucial for every PLC programmer to know the basics of counting in a PLC program. In this article I will explain how counters work, and how you can use them in your PLC programs.

## ****1.Basics of Counting in a PLC****

Before you start counting in any PLC program there are some basics you should know first. These are basic information about the counter instructions and the PLC itself.

First of all, it is important to know about the limits of counting in a PLC.

#### ****Counter Limits****

Counters use variables of certain data types to store numbers in the PLC. All counters need to store at least two numbers:

1. Counter Limit
2. Current Counter Value

Since these two numbers are saved in a certain data type they also have their limits. Many PLC’s save these two numbers as **WORD** or **Integer** data types and if you remember the basics of PLC data types, you will know that a WORD takes up 16 bits.

A signed integer also takes up 16 bits, but the first one is used for signing, so you will only have 15 bits for the actual number

With a little bit of calculation we’ll quickly find out that the maximum value of a WORD is **16.535**. The maximum value of the signed integer is **32.767**.

Although it is rare that you will need to count to such high numbers, it is still important know to avoid overflow errors.

#### ****1.1 High-Speed Counters****

Sometimes the frequency of the pulse you’re trying to count is just too high. This often happens with encoders and other high speed components. In this case you will need a special input module for high frequency (speed) signals.

These input modules are better known as high-speed counters and they are built to capture inputs of various high frequencies.

#### ****1.2 Counter Function Blocks****

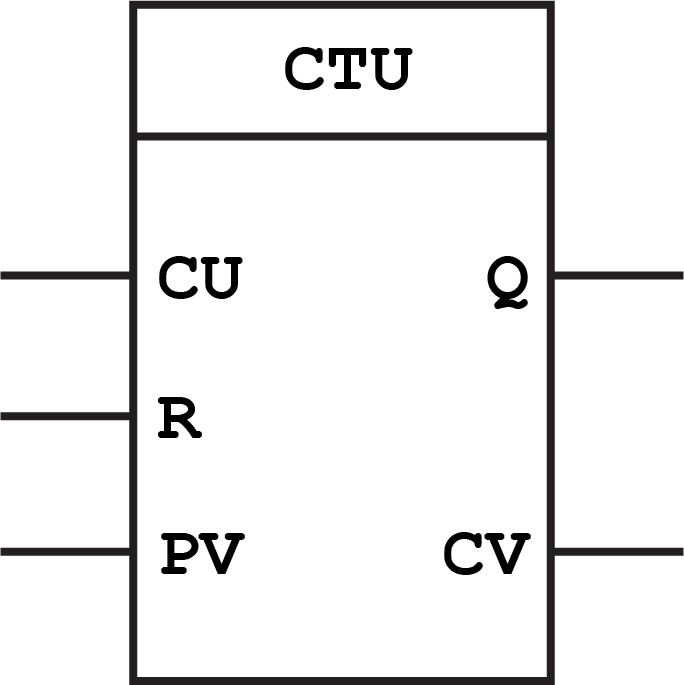
All counter operations or counter function blocks has some inputs and some outputs. In fact, I talked about two of them before (counter limit and current counter value).

What is important to know here is which data type the inputs and outputs take. Normally for counters this is boolean and WORD data types, but it can depend on the PLC platform you’re using. So, don’t forget to check the documentation for the counter on the PLC platform you’re using.

## ****1.3 Up Counter (CTU)****

The first counter instruction I will introduce you to is the up counter, also known as just CTU. As the name implies, this PLC counter is used for counting up.

You can see the up counter function block illustrated below:



Up Counter (CTU) Function Block

The way is works is that it will set an output, when is has counted a certain amount of times. To be a little more specific this is how it works.

#### ****How it works****

Each pulse on the count input (CU) will increase the current counter value (CV) by 1. When CV is greater than or equal to the counter limit (PV) the output (Q) is set. A pulse on the reset input (R) will reset the value of CV to 0.

#### ****Example:****

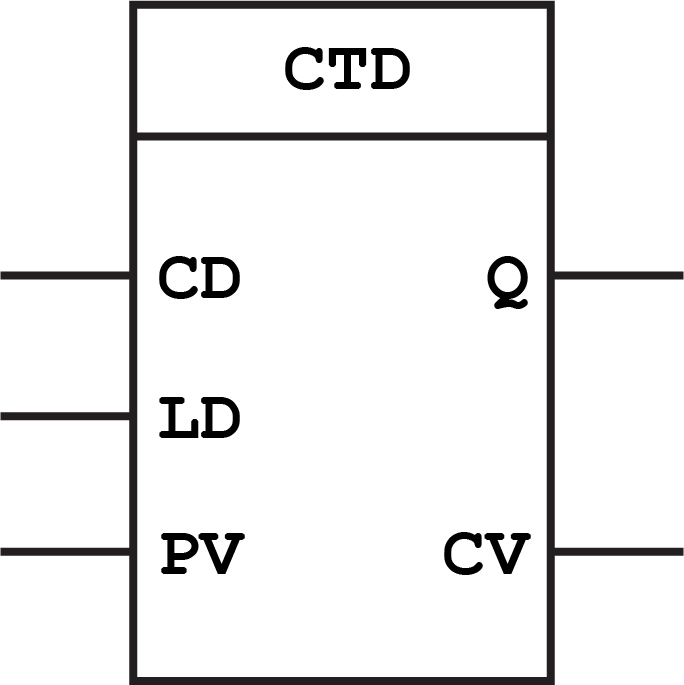
Up counters are usually used to keep track of how many times an event has happened. Let’s say you want a process to complete 10 times before cleaning needs to happen.

For this you have to set the counter limit (PV) to 10. Each time the process has completed you will give a pulse on the count input (CU). When the process has completed 10 times, the output (Q) will be set. Now you can use that output to for example set an alarm that the system needs cleaning.

## ****1.4 Down Counter (CTD)****

Counting down is another operation that is widely used in PLC programming. In some cases you want to know how many counts are remaining before the limit is reached. With the up counter you can use some math to do it. But you can do it easily with a down counter.

Because with the down counter you will count down from a certain number until it reaches 0.



Down Counter (CTD) Function Blok

As you can see the down counter has a pin called LD instead of the reset. It is called load and is used for loading a value into the current counter value. Because when you count down to 0 you will need some initial value of the counter.

#### ****How it works****

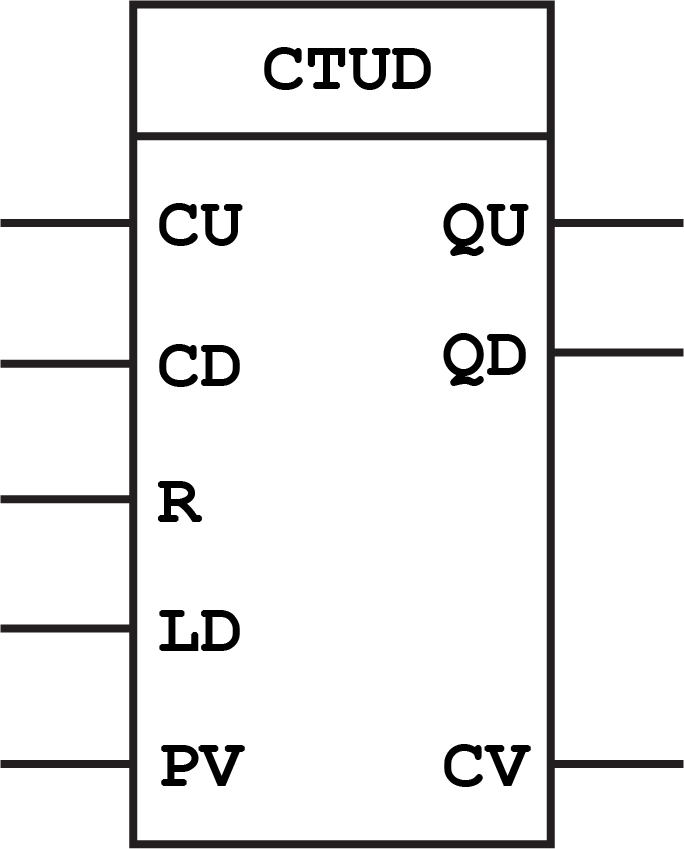
Each pulse on the count input (CD) will decrement current counter value (CV) by 1. When CV is less than or equal to 0 the output (Q) is set. A pulse on the load input (LD) will assign the value of counter limit (PV) to CV.

#### ****Example:****

Imagine you have a semi-automatic process where the operator needs to do a manual task to start a process. The process need to be completed 10 times but when there is 2 times left the operator needs to inspect. It is important that the operator can see how many processes are left, since the total amount of processes can vary. This is where the down counter comes in handy. What you need to do here is to assign the total amount of process times to the counter limit (PV) and then give at pulse at the load input (LD). Now you have to total amount as the current counter value (CV). Every time the process has completed you give a pulse on the count input (CD). You can now use CV to show the operator how many processes are left. When he can see that there are 2 left, he needs to inspect. Additionally you can even add an alarm by comparing CV to 2 with an [equality operator](https://www.plcacademy.com/function-block-diagram-programming/#comparison-function-blocks).

## ****1.5 Up Down Counter (CTUD)****

At last you have the up down counter which can count both ways. Sometimes the combination of the up and the down counter can be useful. You can count the same number both up and down and set both an upper and a lower limit. As you can see below it has a bit more pins than the two other counters:



Up Down Counter CTUD Function Block

The whole idea about this counter is that it can count both up and down the same current counter value (CV). At the top of the function block you can see that there are 2 count inputs. One for counting up and one for counting down.

#### ****How it works****

Each pulse on the count input (CU) will increase the current counter value (CV) by 1. And each pulse on the count input (CD) will decrement current counter value (CV) by 1.

When CV is greater than or equal to the counter limit (PV) the output (QU) is set. When CV is less than or equal to 0 the output (QD) is set.

A pulse on the reset input (R) will reset the value of CV to 0. A pulse on the load input (LD) will assign the value of counter limit (PV) to CV.

#### ****Example:****

One way of using the up down counter is to count finished products. Say you want to produce a certain amount of products in a [batch control](https://www.plcacademy.com/isa-88-s88-batch-control-explained/) system. But after every product is finished a manual inspection is needed. If a product is discarded you will have to subtract it from the number of finished products.

For this you can use the up down counter. Every time a product is finished you can give a pulse at the count up input (CU). But if at inspection the product fails the quality test you can give a pulse at the count down input (CD) to decrement the amount of products produced.

### 1.6 Programmable logic relay (PLR)

In more recent years, small products called PLRs (programmable logic relays), and also by similar names, have become more common and accepted. These are much like PLCs, and are used in light industry where only a few points of [I/O](https://en.wikipedia.org/wiki/I/O) (i.e. a few signals coming in from the real world and a few going out) are needed, and low cost is desired.

These small devices are typically made in a common physical size and shape by several manufacturers, and branded by the makers of larger PLCs to fill out their low end product range. Popular names include PICO Controller, NANO PLC, and other names implying very small controllers. Most of these have 8 to 12 discrete inputs, 4 to 8 discrete outputs, and up to 2 analog inputs. Size is usually about 4" wide, 3" high, and 3" deep.

Most such devices include a tiny postage-stamp-sized LCD screen for viewing simplified ladder logic (only a very small portion of the program being visible at a given time) and status of I/O points, and typically these screens are accompanied by a 4-way rocker push-button plus four more separate push-buttons, similar to the key buttons on a VCR remote control, and used to navigate and edit the logic.

# **PLC compared with other control systems**

[](https://en.wikipedia.org/wiki/File:BMA_Automation_Allen_Bradley_PLC_3.JPG)

PLC installed in a control panel

[https://upload.wikimedia.org/wikipedia/commons/thumb/f/f3/Control-panel.jpg/220px-Control-panel.jpg](https://en.wikipedia.org/wiki/File:Control-panel.jpg)

Control center with a PLC for a [RTO](https://en.wikipedia.org/wiki/Regenerative_thermal_oxidiser).

PLCs are well adapted to a range of [automation](https://en.wikipedia.org/wiki/Automation) tasks. These are typically industrial processes in manufacturing where the cost of developing and maintaining the automation system is high relative to the total cost of the automation, and where changes to the system would be expected during its operational life. PLCs contain input and output devices compatible with industrial pilot devices and controls; little electrical design is required, and the design problem centers on expressing the desired sequence of operations. PLC applications are typically highly customized systems, so the cost of a packaged PLC is low compared to the cost of a specific custom-built controller design.

On the other hand, in the case of mass-produced goods, customized control systems are economical. This is due to the lower cost of the components, which can be optimally chosen instead of a "generic" solution, and where the non-recurring engineering charges are spread over thousands or millions of units.

For high volume or very simple fixed automation tasks, different techniques are used. For example, a cheap consumer [dishwasher](https://en.wikipedia.org/wiki/Dishwasher) would be controlled by an electromechanical [cam timer](https://en.wikipedia.org/wiki/Cam_timer) costing only a few dollars in production quantities.

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

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

Programmable controllers are widely used in motion, positioning, or torque control. Some manufacturers produce motion control units to be integrated with PLC so that [G-code](https://en.wikipedia.org/wiki/G-code) (involving a [CNC](https://en.wikipedia.org/wiki/CNC) machine) can be used to instruct machine movements.

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

PLCs have similar functionality as [remote terminal units](https://en.wikipedia.org/wiki/Remote_terminal_unit). An RTU, however, usually does not support control algorithms or control loops. As hardware rapidly becomes more powerful and cheaper, RTUs, PLCs, and DCSs are increasingly beginning to overlap in responsibilities, and many vendors sell RTUs with PLC-like features, and vice versa. The industry has standardized on the [IEC 61131-3](https://en.wikipedia.org/wiki/IEC_61131-3) functional block language for creating programs to run on RTUs and PLCs, although nearly all vendors also offer proprietary alternatives and associated development environments.

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

# **1.Discrete (digital) and analog signals**

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

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

[Current inputs](https://en.wikipedia.org/wiki/Current_loop) are less sensitive to electrical noise (e.g. from welders or electric motor starts) than voltage inputs.

PLCs are at the forefront of manufacturing automation. An engineer working in a manufacturing environment will at least encounter some PLCs, if not use them on a regular basis. Electrical engineering students should have basic knowledge of PLCs because of their widespread use in industrial applications.

**2.ADVANTAGES OF PROGRAMMABLE CONTROLLER**

1. Very fast
2. Easy to change logic i.e. flexibility
3. Reliable due to absence of moving parts
4. Low power consumption
5. Easy maintenance due to modular assembly
6. Facilities in fault finding and diagnostic
7. Capable of handling of very complicated logic operations
8. Good documentation facilities
9. Easy to couple with the process computers
10. Analog signal handling and close loop control programming
11. Counter, timer and comparator can be programmed
12. Ease operator interface due to colourographic and advisory system introduction

**3. Disadvantages**

• It requires training because it is still a new technology  
• The use of PLC is not really efficient when dealing with some applications that perform a single function  
• In some cases, it can be quite difficult when dealing with replacements or changes  
• It involves too much work when we need to connect wires  
• Sometimes, it can be quite difficult to find errors

## PLC: Industrial Applications of Programmable Logic Controller

In the most basic terms, a programmable logic controller ([PLC](https://en.wikipedia.org/wiki/Programmable_logic_controller)) is a computer with a microprocessor but has no keyboard, mouse or monitor. It is essentially built to withstand very harsh industrial environments.

It is a distinctive form of computer device designed for use in industrial control systems. It has a robust construction and unique functional features such as sequential control, ease of programming, timers and counters, easy-to-use hardware and reliable controlling capabilities.

It is designed to be enormously robust, so it could withstand harsh industrial environments such as extreme temperatures, vigorous vibrations, humidity and electrical noise.

The logic controllers are often tasked to control and monitor a very large number of sensors and actuators. They are therefore different from other regular computer systems in their extensive I/O (input/output) arrangements.

In addition to being used as a special-purpose digital computer, the PLC can be used in other control-system areas and industries. This explains why PLCs are often referred to as industrial PCs.

The PLC is also commonly used in civil applications such as in washing machines and for controlling traffic signals and elevators. They are used in many industries to monitor and control production processes and building systems.Once programmed, the PLC will perform a sequence of events triggered by stimuli referred to as inputs. It receives these stimuli through delayed actions such as counted occurrences or time delays

These special computer devices are different from regular computers such as PCs or smartphones in that:

1. A PLC performs only a single set or sequence of tasks, with greater reliability and performance, except when it is under real-time constraints. This is in contrast to regular PCs and smartphones that are designed to execute any number of roles simultaneously within the Windows framework.

2. The PLC has a number of features that you don’t find in normal computers, such as protection from the open area conditions like heat, dust and cold.

3. It is low cost compared with other microcontroller systems. When you’re using a PLC in various applications, you only need to change the software component for each application. With other microcontroller systems however, you would have to change the hardware components too with different applications.

# **PLC Program Used Examples**

# **1.PLC Program to Implement 4:1 Multiplexer**

This is a PLC Program to Implement 4:1 Multiplexer.

**Problem Description**

Implementing 4:1 multiplexer in PLC using Ladder Diagram programming language.

**Problem Solution**

* There are m-data inputs, one output and n select lines, with 2m = n.
* To select n inputs, we need m select lines such that 2m = n. Depending on the output. The selection of one of the n inputs is done by the select pins.
* It does not need K-map and simplification so one step is eliminated to create Ladder Logic Diagram.
* Realize the multiplexer using Logic Gates.
* Truth Table can be written as given below.

Data Select Inputs Output

Inputs S1 S0 Q

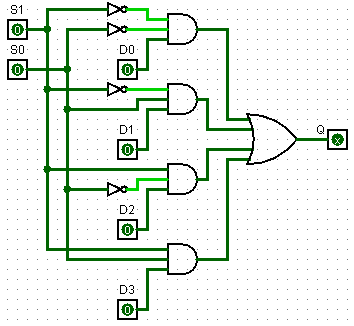
D0 0 0 D0

D1 0 1 D1

D2 1 0 D2

D3 1 1 D3

**Realizing 4:1 Mux using Logic Gates**

[](https://www.sanfoundry.com/wp-content/uploads/2016/09/plc-program-implement-4-1-multiplexer-01.png)

**PLC Program**

Here is PLC program to Implement 4:1 Multiplexer, along with program explanation and run time test cases.

List of Inputs and Outputs

S1= I:1/0 (Select Line Input)

S0= I:1/1 (Select Line Input)

D0= I:1/2 (Data Line Input)

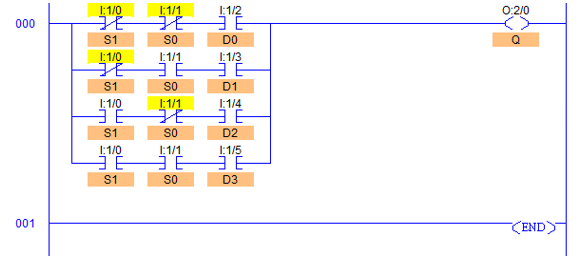
D1= I:1/3 (Data Line Input)

D2= I:1/4 (Data Line Input)

D3= I:1/5 (Data Line Input)

Q= O:2/0 (Output)

**Ladder Diagram to obtain Binary output**

[](https://www.sanfoundry.com/wp-content/uploads/2016/09/plc-program-implement-4-1-multiplexer-02.png)

**Program Description**

* In all the rungs, S1 (I:1/0) and S0 (I:1/1) are used as a selector line input as shown in Logic Circuit.
* D0, D1, D2 and D3, I:1/2, I:1/3, I:1/4 and I:1/5 are Data Inputs respectively.
* When S1 (I:1/0) and S0 (I:1/1) are low, output will have whatever state D0 I:1/2 holds, either 1 or 0.
* When S1 (I:1/0) is low and S0 (I:1/1) is high, output will have whatever state D1 I:1/3 holds.
* Similarly remaining two different inputs are performed.
* In other words, according to bit pattern of S1 and S0, Data bits D0 to D3 are passed to output.
* Here, instead of bits D0 to D3, any functions such as moving, jumping or moving can be performed as well depending upon the application.

**Runtime Test Cases**

Data Select Inputs Output

Inputs S1 S0 Q

D0-0/1 0 0 D0-0/1

D1-0/1 0 1 D1-0/1

D2-0/1 1 0 D2-0/1

D3-0/1 1 1 D3-0/1

# **2.PLC Program to Implement 8:1 Multiplexer**

This is a PLC Program to Implement 8:1 Multiplexer.

**Problem Description**

Implementing 8:1 Multiplexer in PLC using Ladder Diagram programming language.

**Problem Solution**

* There are m-data inputs, one output and n select lines, with 2m = n.
* To select n inputs, we need m select lines such that 2m = n.
* Depending on the output. The selection of one of the n inputs is done by the select pins.
* It does not need K-map and simplification so one step is eliminated to create Ladder Logic Diagram.
* Realize the multiplexer using Logic Gates.
* Truth Table can be written as given below.

Data Select Inputs Output

Inputs S2 S1 S0 Q

D0 0 0 0 D0

D1 0 0 1 D1

D2 0 1 0 D2

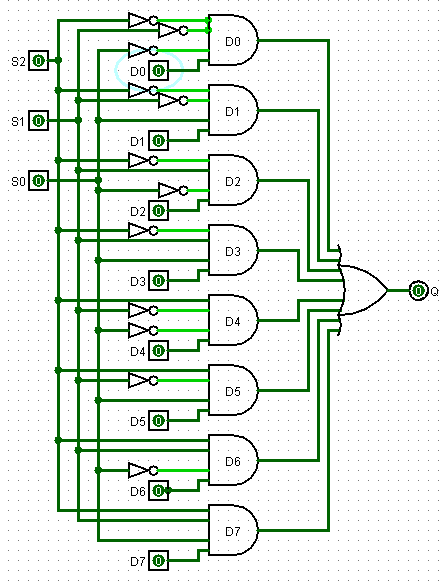
D3 0 1 1 D3

D4 1 0 0 D4

D5 1 0 1 D5

D6 1 1 0 D6

D7 1 1 1 D7

**Realizing 8:1 Mux using Logic Gates**  
[](https://www.sanfoundry.com/wp-content/uploads/2016/09/plc-program-implement-8-1-multiplexer-01.png)

**PLC Program**

Here is PLC program to Implement 8:1 Multiplexer, along with program explanation and run time test cases.

List of Inputs and Outputs

S2= I:1/0 (Select Line Input)

S1= I:1/1 (Select Line Input)

S0= I:1/2 (Select Line Input)

D0= I:1/3 (Data Line Input)

D1= I:1/4 (Data Line Input)

D2= I:1/5 (Data Line Input)

D3= I:1/6 (Data Line Input)

D4= I:1/7 (Data Line Input)

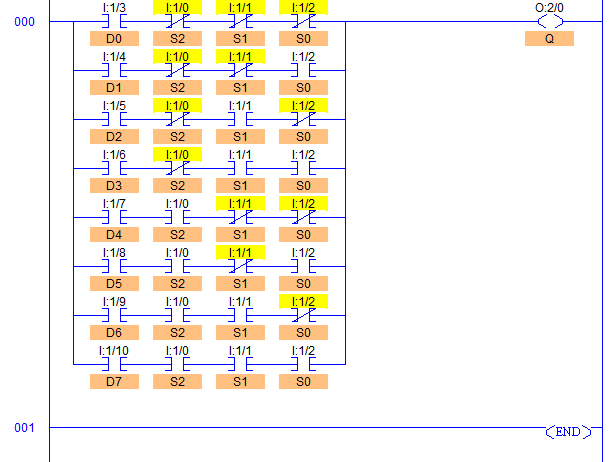
D5= I:1/8 (Data Line Input)

D6= I:1/9 (Data Line Input)

D7= I:1/10 (Data Line Input)

Q= O:2/0 (Output)

**Ladder Diagram to obtain output**

[](https://www.sanfoundry.com/wp-content/uploads/2016/09/plc-program-implement-8-1-multiplexer-02.png)

**Program Description**

* In all the rungs, S2 (I:1/0), S1 (I:1/1) and S0 (I:1/2) are used as a selector line input as shown in Logic Circuit.
* D0 to D7, I:1/3 to I:1/10 are Data Inputs respectively.
* When S2 (I:1/0), S1 (I:1/1) and S0 (I:1/2) are low, output will have whatever state D0 I:1/3 holds, either 1 or 0.
* When S2 (I:1/0) and S1 (I:1/1) are low and S0 (I:1/1) is high, output will have whatever state D1 I:1/4 holds.
* Similarly remaining six different inputs are performed.
* In other words, according to bit pattern of S2, S1 and S0, Data bits from D0 to D7 are passed to output.
* Here, instead of bits D0 to D7, any functions such as moving, jumping or moving can be performed as well depending upon the application.

**Runtime Test Cases**

Data Select Inputs Output

Inputs S2(I:1/0) S1(I:1/1) S0(I:1/2) Q(O:2/0)

D0 (I:1/3) 0 0 0 D0 0 or 1

D1 (I:1/4) 0 0 1 D1 0 or 1

D2 (I:1/5) 0 1 0 D2 0 or 1

D3 (I:1/6) 0 1 1 D3 0 or 1

D4 (I:1/7) 1 0 0 D4 0 or 1

D5 (I:1/8) 1 0 1 D5 0 or 1

D6 (I:1/9) 1 1 0 D6 0 or 1

D7 (I:1/10) 1 1 1 D7 0 or 1

# **3.PLC Program to Implement SR Flip-Flop**

This is a PLC Program to Implement SR Flip-Flop.

**Problem Description**

To Implement SR Set-Reset Flip Flop in PLC using Ladder Logic programming language.

**Problem Solution**

* SR Flip-Flop is also known as Latch since it is capable of locking the information.

The SR flip flop first executes the set instruction and then reset instruction, so the address remains reset for the remainder of program scanning.

* In many PLC vendors like Siemens, Omron and many others, SR Flip Flop is included as an instruction in the instruction set. So no logic is required to implement SR Flip Flop in such PLCs.
* In Omron PLCs, you can develop the same logic using KEEP instruction.
* K-map solving is again not required to solve this problem.
* By observing Truth table of SR Flip Flop, required output is obtained in accordance with the input provided and bits storing.
* Truth table relating SR Flip Flop is given below.

**Truth Table relating to SR Flip Flop**

Inputs Outputs

S R Q Q^ Qn+1 Q^n+1

0 0 0 1 0 1

0 0 1 0 1 0

0 1 0 1 0 1

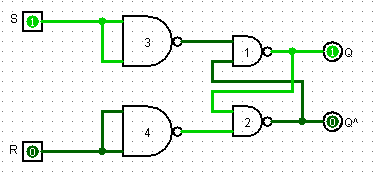
0 1 1 0 0 1

1 0 0 1 1 0

1 0 1 0 1 0

1 1 0 0 x x

**Realization of S-R Latch using Logic Gates**

[](https://www.sanfoundry.com/wp-content/uploads/2016/09/plc-program-implement-sr-flip-flop-01.png)

**PLC Program**

Here is PLC program to Implement SR Flip-Flop, along with program explanation and run time test cases.

List of Inputs and Outputs

Set = I:1/0 (Set Input)

Reset = I:1/1 (Reset Input)

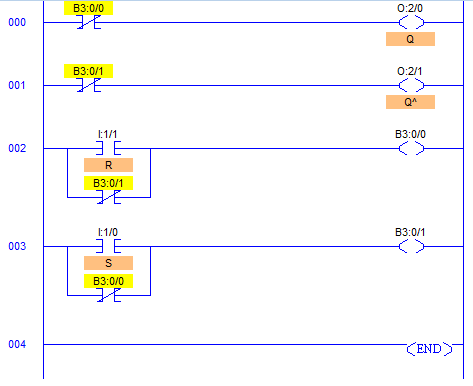
Q (Set) = O:2/0 (Q Output)

Q^(Reset) = O:2/1 (Q^ Output)

Qn+1 = B3:0/0 (Relay Bit)

Q^n+1 = B3:0/1 (Relay Bit)

**Ladder Diagram to obtain output**

[](https://www.sanfoundry.com/wp-content/uploads/2016/09/plc-program-implement-sr-flip-flop-02.png)

**Program Description**

* By definition, a condition of Q (O:2/0) = 1 and Q^ (O:2/1) = 0 is Set and a condition of Q (O:2/0) = 0 and Q^ (O:2/1) = 1 is Reset.
* If the signal state is high at input I:1/0 and low at I:1/1, bit B3:0/1 is set and output O2:0/0 is set to 1 which is a SET condition of this logic.
* Otherwise, if the signal state at input I:1/0 is low and at input I:1/1 is high, bit B3:0/1 is reset and output O:2/0 is reset which is a RESET condition of this logic.
* During power up, when both the inputs are low, Q (O:2/0) will go high because of its order.
* And after either of the states is achieved, if both signal states go low, nothing is changed which is latched state.
* If both signal states are set to high, the Q^ output O:2/1 instruction dominates because of the order in the ladder diagram, B3:0/1 is low and B3:0/0 is high causing Q output O:2/0 to reset.
* When S I:1/0 and R I:1/1 both are equal to 0, the outputs “latch” in their prior states.
* Slight delay may occur in inputs and resulting changes in outputs due to PLC’s program scan time.

**Runtime Test Cases**

Input Output

S R Q Q^

0 0 Latch Latch

0 1 0 1

1 0 1 0

1 1 0 0