PLC Basics: The Ultimate Guide in 2026 - PLCGurus.NET

By PLCGuru

Published: 2025-03-12 · Archived: 2026-04-05 17:10:09 UTC

We are often discussing more advanced topics here at PLCGurus.NET for the more seasoned programmers. This article aims to change that! Operator training is crucial for the effective use and maintenance of PLC systems.

Targeting foundational Programmable Logic Controller skills (PLC Basics), this article will serve as a starting point for those individuals looking to enter into the fascinating world of PLCs, Automation and Control.

This article will cover PLC basics in the following topic areas:

- What is a PLC?
- Why You Should Use a PLC
- How Do PLCs Work?
- Fundamental Difference Between a PLC and PC
- Working With Number Systems
- Difference Between Discrete and Analog I/O?
- Logic Gates & Boolean Expressions
- Ladder Logic Programming
- Function Block Programming
- Sequential Text Programming
- Considerations When Choosing a PLC

The complexity of control processes plays a crucial role in determining the type of PLC needed, with simpler processes requiring basic controls and more intricate operations necessitating advanced capabilities.

If you're new to programmable logic control and the world of industrial automation, then join us as we explore each one of these topics at length.

Programming

PLC Basics – What Is A PLC?

The term PLC stands for **P**rogrammable **L**ogic **C**ontroller. The PLC was invented by a young engineer, Dick Morley, back in 1964. Since this time, the PLC has revolutionized the industrial and manufacturing landscape to become arguably the most integral component of any industrial process in existence today. PLCs have revolutionized industrial automation by replacing complex relay-based control systems with easier software-based logic.

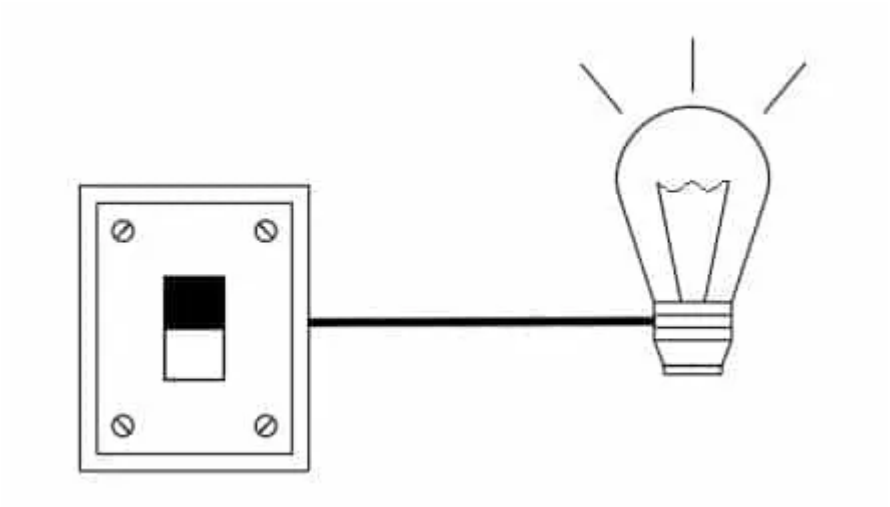
Initially the PLC was used to replace relay logic, but today its range of functions includes timing, counting, calculating, comparing, and the processing of analog signals.

A PLC is known as a “hard real-time system” due to the fact that outputs must respond to changing inputs within a very stringent time constraint. That said, you can think of a PLC as nothing more than a dedicated computer designed for fast switching and decision making.

The main advantage of a PLC over a “hard-wired” type relay control systems, is that PLCs lend themselves to functional changes very easily. What do I mean by this you ask? PLCs simplify system updates and maintenance by eliminating the need for rewiring when modifying control logic.

A Simple Example

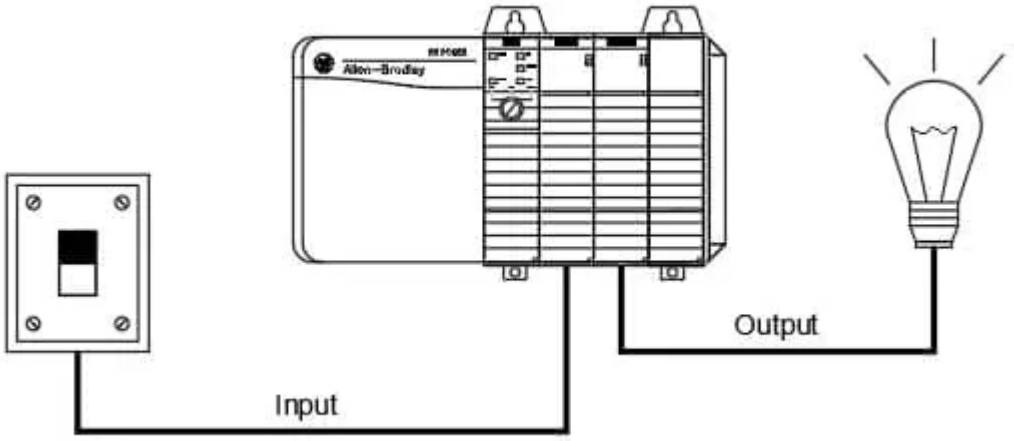
Imagine you have a switch that turns on a light. The light has two states, ON or OFF, and will respond almost instantaneously to someone either flipping the switch ON, or OFF.



Now imagine you wire this switch up directly to the light and your boss comes to tell you that he would like the light to come on precisely 30 seconds after the switch turns on…you have a problem! In order to achieve this it will require additional hardware, i.e., a timing relay, and some rewiring. Now enter the PLC!

With a PLC there will be no need to buy additional hardware every time a change is required. All that will be needed is a simple programming change that will delay the light from turning on until 30 seconds after the switch is thrown.

Computer Hardware



As you can see the light switch becomes an “input” to the PLC, and the light itself is an “output”.

Granted, this is a very simple example, however, even larger more complex systems are nothing more than a combination of switch inputs and an assortment of outputs. Then by using software we can build logic to control the outputs based in the input conditions. The PLC sends control signals to manage the light, ensuring it turns on 30 seconds after the switch is activated.

PLC Basics – How Do PLCs Work?

Programmable logic controllers are specialized computers designed to run manufacturing processes. The structure of a PLC is based on the same principles as those employed in computer architectures. You can think of a PLC as a highly “ruggedized” industrial computer that is designed to withstand harsh environments (i.e., high temperatures, dirty or dusty environments) and is highly modular.

This means that you can add various Input/Output modules and module types in an almost unlimited ordering – the only constraint being the number of Input/Output slots you have available in your physical rack (chassis). The main components of a PLC are:

- Rack or Chassis
- Power Supply
- Central Processing Unit (CPU)
- Communication Modules
- Inputs and Outputs

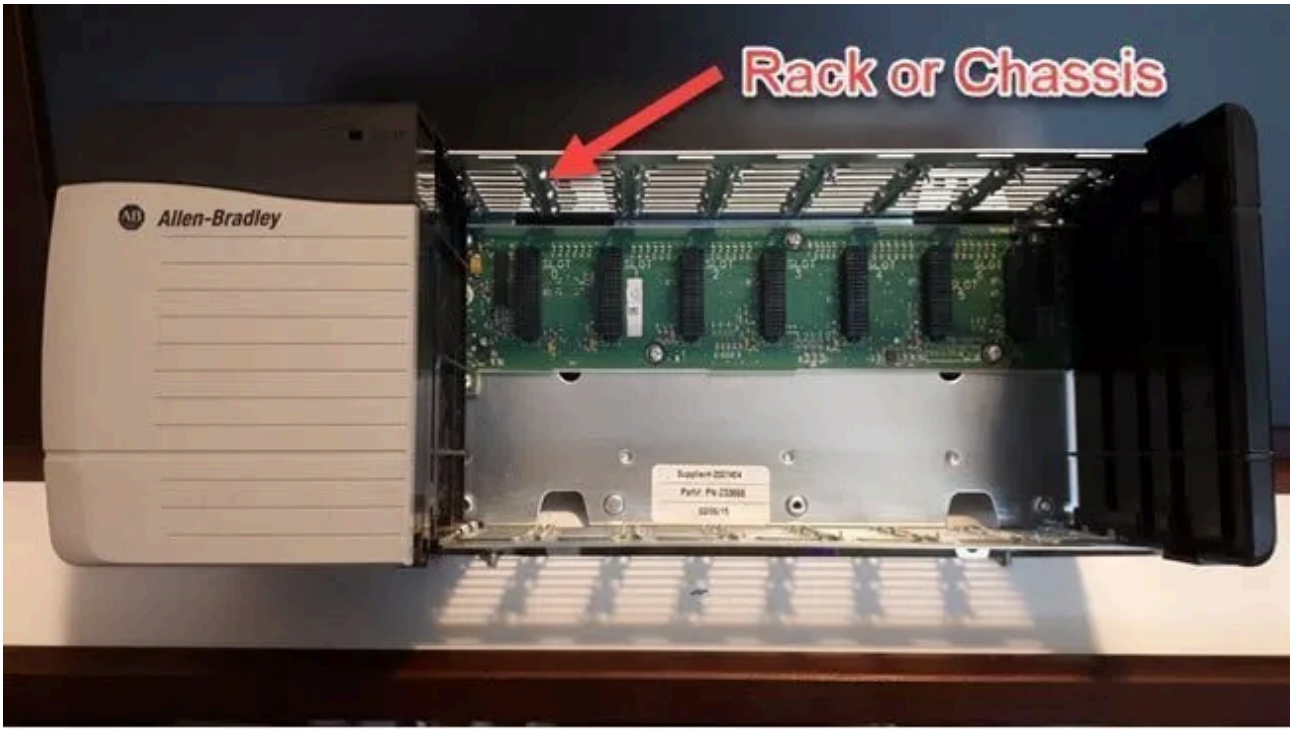
Digital inputs play a crucial role in receiving and processing signals from various sensors and switches, converting real-world signals into binary data that the CPU can interpret and act upon. Input devices transmit signals to the PLC from sensors and switches that monitor environmental changes.

PLCs come in all shapes and sizes depending on the process you are intending to control, the memory and Input/Output (I/O) requirements.

PLC Basics – Rack (Chassis)

The PLC rack (also known as “chassis”) is the backbone of all PLC systems. The chassis provides all the necessary mounting for the power supply, backplane and all I/O modules that will reside in it. Illustrated in the images below is an example of an Allen-Bradley ControlLogix 1756-A7 chassis.

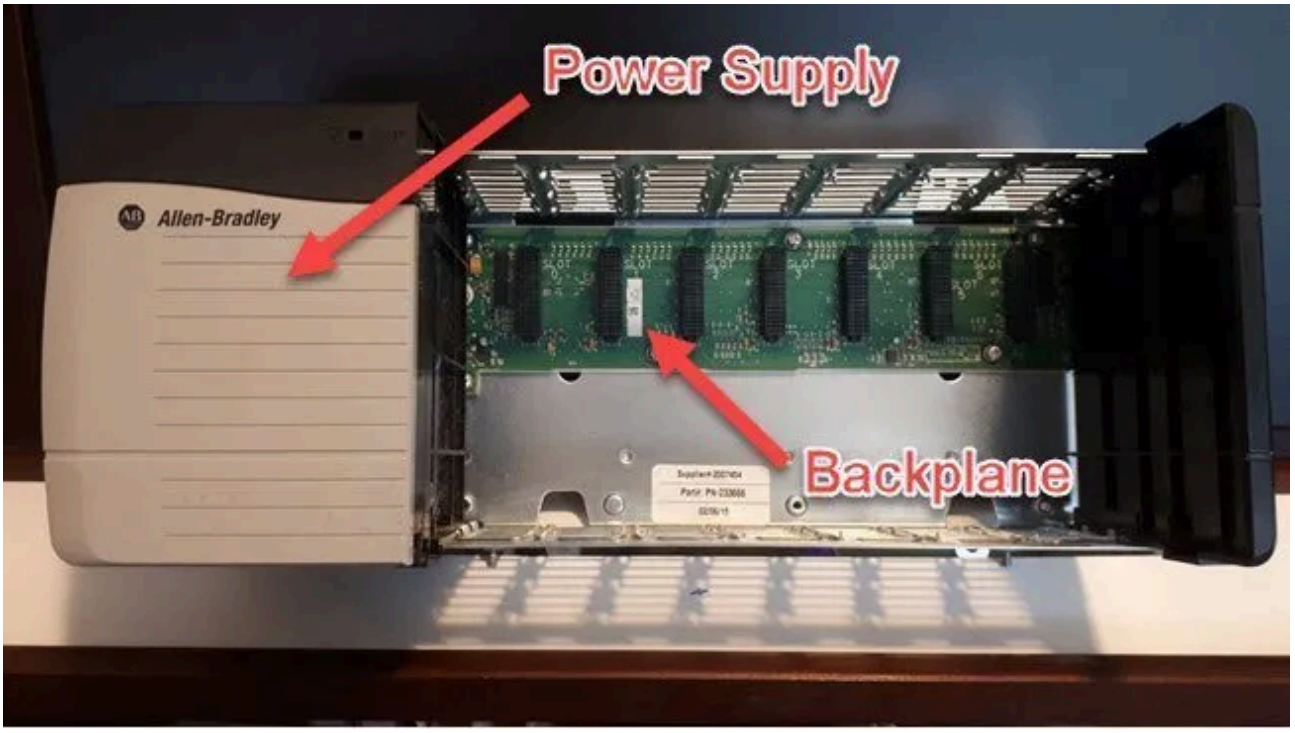
Electronics & Electrical



As you can see in this image that aside from the power supply unit located to the far left of the chassis, the rack is essentially empty.

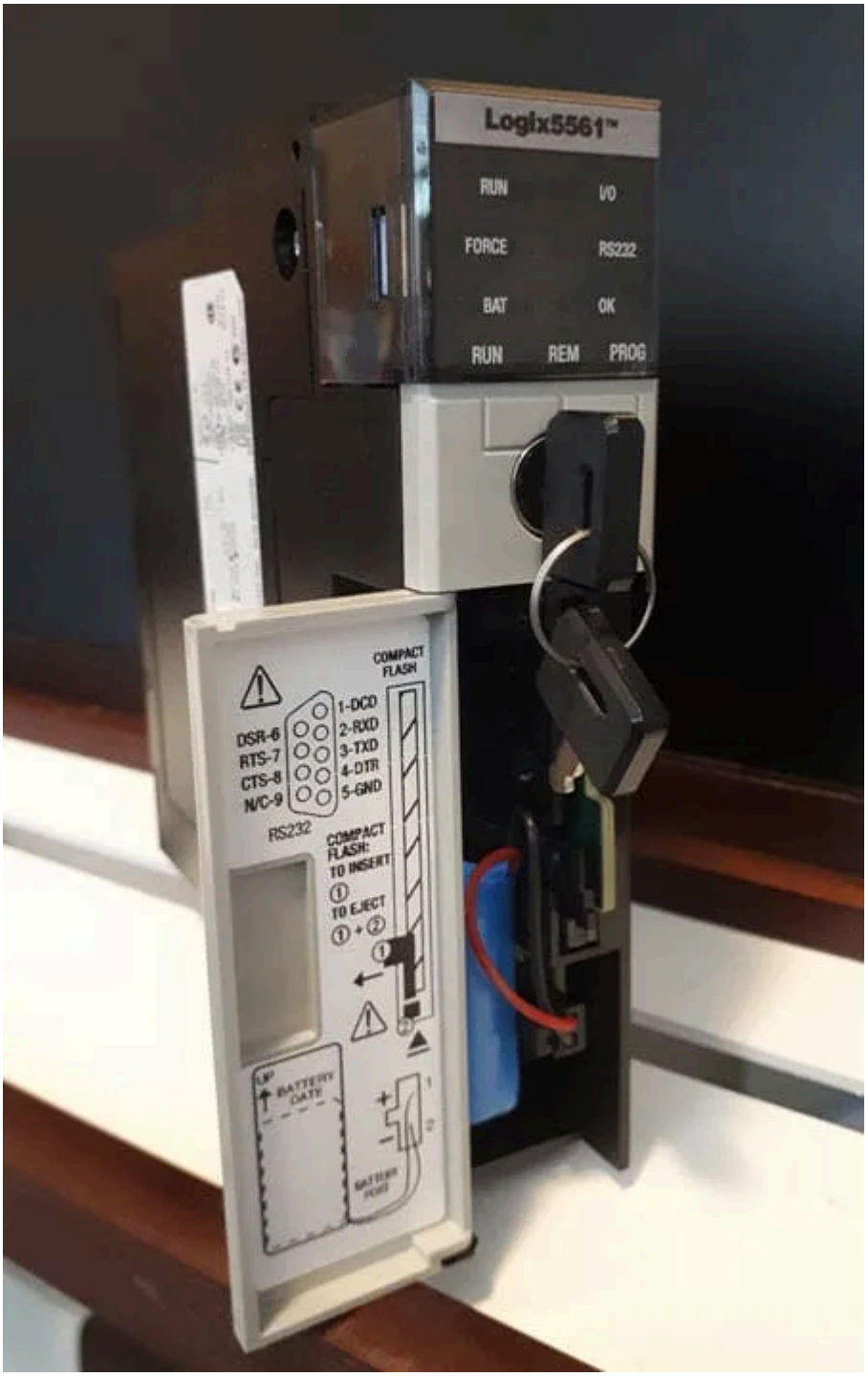
PLC Basics – Power Supply and Backplane

The power supply provides all the necessary voltages to the “backplane” of the PLC chassis. Typical voltages will be 24 VDC and 5 VDC to power the CPU, Communication, and Input/Output Modules.

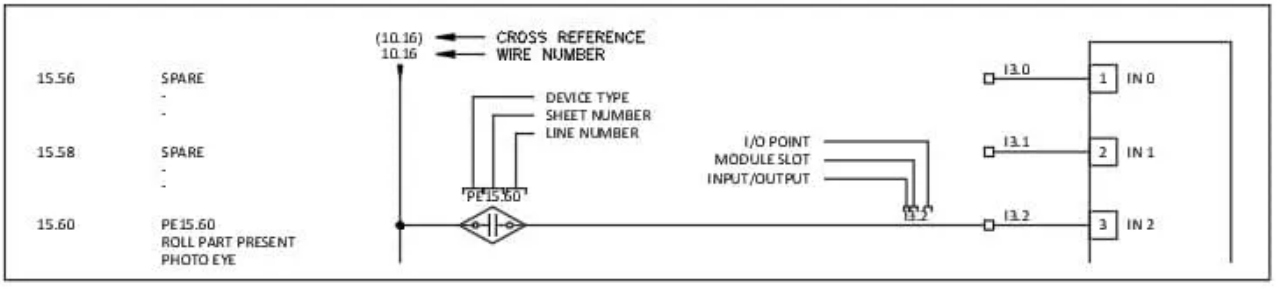


PLC Basics – Central Processing Unit (CPU)

The central processing unit or CPU is the main “brain” of the PLC. The memory structure of a PLC processor consists of several areas, some of these having specific roles.



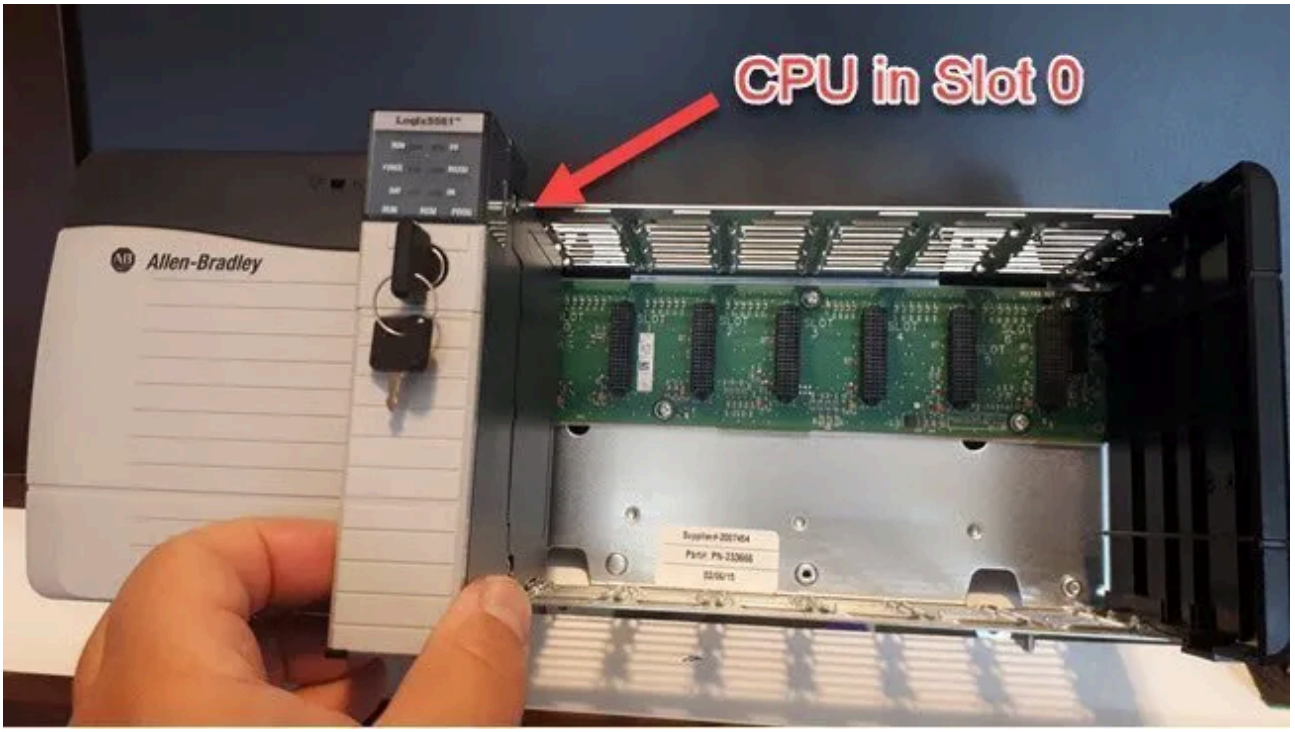
With “rack-based” memory structure addresses are derived using the rack number, the I/O module slot number and the screw terminal number where the I/O device is wired into.



A typical “rack-based” PLC is the SLC 500 platform of programmable logic controllers. Their memory space is divided into two broad categories, namely, *Program Files* and *Data Files.*

With “tag-based” memory structures all data are assigned a variable name called a “tag”. A program can be developed using only tag names but you must assign input and output tags before the program can be executed. The ControlLogix platform of programmable logic controllers employ “tag-based” memory structures.

A PLC program enhances the efficiency and functionality of the system by allowing for complex applications and simplifying the programming process, reducing the need for extensive wiring and making system modifications easier.



Although the image above by convention is residing in Slot 0, with the Logix platform of controllers this is not mandatory as it was in previous platforms of PLC such as the SLC 500 family of controllers.

PLC Basics – Communication Modules

Communication modules allow the PLC to “talk” over various network protocols. Common modern network protocols used are ControlNet, DeviceNet, and Ethernet/IP. These modules will allow the CPU to communicate to other PLC controllers and/or Remote I/O racks that are distributed in the field.

The advantage of Remote I/O (I/O that is field distributed) versus Local I/O (these are input and output modules that reside in the same rack as the CPU), is that it saves on installation time and money. Rather than having to run the wires for all your field input and output devices back to the panel where the local PLC resides, you can “drop” a Remote I/O rack in close proximity to the field I/O devices and wire them up locally.

After the remote rack is wired up in the field, you merely need to run a communication cable, i.e., Ethernet cable, and possibly a couple of low voltage power conductors to power the remote I/O communication adapter its I/O.

Communication modules send control signals to connected output devices like motors and valves, ensuring efficient operation of the industrial process. They provide installation flexibility and data acquisition capabilities.

PLC Basics – Modes Of Operation

A PLC has basically two modes of operation: the *Program Mode* and some variation of the *Run Mode.* A three-position keyswitch may be used to select different processor modes of operation.

- ***Program Mode*** is used to enter a new program, edit or update an existing program, upload files and download files. It is important to note that in this mode of operation all outputs are de-energized.
- ***Run Mode*** is used to execute the user program. Remote PLC programming or mode selection is disabled when the key is in the *Run Mode* position.
- ***Remote Run Mode (REM)*** allows the PLC to be remotely changed between program and run mode by a personal computer connected either directly or via a communication protocol to the PLC processor. Typically most processors are placed into *REM* mode to allow the engineering or maintenance staff the greatest flexibility when performing PLC programming tasks.

PLC Basics – Understanding CPU Scan

Very simply, during each program scan cycle the processor reads all the inputs, takes these value, and energizes or de-energizes the outputs according to the user program.

The Program Scan Cylce looks like this:

1. **Scan Inputs** – is the input ON (1) or OFF (0).
2. **Execute Program Logic** – executing each instruction and solve the rung logic.
3. **Update Outputs** – write a logic 1 (ON) or 0 (OFF) to the output.
4. **House-Keeping** – perform internal checks and system tasks.

Granted this is a bit of a simplification and with more modern PLC's or PAC's (Programmable Automation Controllers) as they're commonly referred, more elaborate scan patterns can be configured. But I say let's not muddy the waters too much here.

If you are interested in a sneak peak at one of the videos in our Studio 5000 Essentials video series where we discuss more advanced scan patterns, go ahead and view it now!

Not to oversimplify the importance of processor scan and its impacts on overall response time, however, since this is an introductory welcome to PLC Basics article.

I will reserve those more advanced discussions for other articles. In fact, we've done an article on this very topic at System Overhead Time Slice.

PLC Basics – Difference Between a PLC and PC

As I mentioned the architecture of a PLC is basically the same as that of a personal computer. However, unlike PC's the PLC is designed to operate in the harshest of industrial environments that have a wide range of ambient temperatures and humidity. Additionally, properly designed PLC installations can mitigate EMI (electro-magnetic interferance – NOISE) present in almost all industrial establishments.

PC's are highly complex computing machines capable of executing several programs and tasks concurrently whereas a PLC is a dedicated real-time system that executes a single (can have multiple programs in ControlLogix system, however, they still executed one at a time in a prioritized ordering) program in an orderly and sequential fashion from first to last instruction.

Unlike PC's, the PLC is programmed in relay ladder logic or other "easily" learned languages. It comes with its programming language built into its memory and has no permanently attached keyboard, monitor, CD drive, printer etc.

PLC control systems have been designed with maintainability and ease of installation as a key factor. Troubleshooting is simplified by the use of fault indicators on the processor and I/O modules. Furthermore, in traditional PLC chassis I/O is modularized to allow easy replacement and configuration.

PLC Basics – Knowing Your Bits and Bytes!

It is important that you understand how PLC memory is arranged. In its most basic form, a piece of data is stored as either a 0 or a 1 in what's referred to as a "**Bit**" of memory.

When 4-bits are stored in contiguous memory it is referred to as a "**Nibble**". When 8-bits are stored in contiguous memory it is referred to as a "**Byte**".

Therefore, **1 Byte = 2 Nibbles = 8 Bits.** Expanding on this concept, when 2-bytes are stored in contiguous memory it is referred to as a "**Word**".

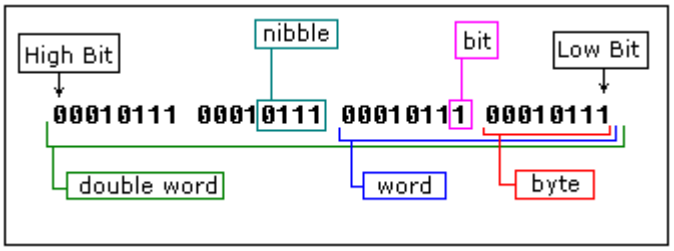
Therefore, **1 Word = 2 Bytes = 16 Bits**. When you hear that a given PLC or computer for that matter is an 8, 16, 32, 64-bit architecture, it is referring to the number of bits allocated in each contiguous memory location.

This means that each memory location in a 16-bit architecture, such as the Allen-Bradley SLC 500 platform of controllers, has 16-bit words, or can represent a signed integer range of -32,768 to 32,767.

With advancements in programmable logic controllers as of late, memory now supports 32-bit architectures. This means that each memory location has 32-bits, which is referred to a double word or "**DWord**".

Therefore, **1 DWord = 2 Words = 4 Bytes = 32 Bits.** A 32-bit architecture can represent a signed integer range of ?2,147,483,648 to 2,147,483,647.

The image below captures everything we've discussed here in visual form:



Now, if we expand this concept to the modern day PC's and laptops that boast a 64-bit architecture, also referred to as a quad-word or "**QWord**", what is the largest signed integer that we can represent with that??? I'll leave it to you to research that, but let's just say it's a really, really big number!

PLC Basics – Working With Number Systems

In order to be proficient as a PLC programmer it is imperative that you are comfortable moving in and out of different numbering systems

By far the most important number systems that you will need to have command over is the binary number system (base 2), or the number system that contains only 0's and 1's. However, there are other number systems we must be comfortable moving in and out of as well, namely, the hexadecimal system (base 16) and the octal system (base 8), and of course the decimal system (base 10), but I'm going to assume you're okay with that one!

If you're unclear why I've included the "base x" in each number system, it's because it gives us an indication of the permissible numbers in that system. What do I mean?

For example, the decimal (base 10) system that we use every day contains valid numbers, 0..9. In general, we can say that for a give number system with a *base n*, there is *0..n-1* numbers we can use in that system – with one exception the hexadecimal system.

- The **Binary System (base 2)** has valid numbers, 0..1
- The **Octal System (base 8)** had valid numbers, 0..7
- The **Hexadecimal System (base 16)** has valid numbers, 0..9 and A..F (more on this later)

PLC's like PC's can only interpret 0's (low signals) and 1's (high signals) because it is made up of millions of tiny transistors that act as switches being driven by high and low electrical signals. This reminds me of a funny joke I once heard,

> There are 3 types of people in the world…those who understand binary, and those who don't!

Alright, maybe not the best joke…but it's a little funny right? Anyhow, knowing the binary number system and these other systems is essential to our understanding of programmable logic control.

PLC Basics – The Binary System

Let's start by looking at the system we are most comfortable with, the decimal system. If we take the following number: 975**10**

What is this number in reality? We said that the decimal system is base 10, so to compute the number 975**10** it is equal to the following:

$$975_{10} = 9 \times 10^2 + 7 \times 10^1 + 5 \times 10^0 = 900 + 70 + 5$$

Notice the "**10**" subscript. We usually indicate the base of the number by including the subscript as shown. Let's try some more:

$$8984_{10} = 8 \times 10^3 + 9 \times 10^2 + 8 \times 10^1 + 4 \times 10^0 = 8000 + 900 + 80 + 4$$

$$75645_{10} = 7 \times 10^4 + 5 \times 10^3 + 6 \times 10^2 + 4 \times 10^1 + 5 \times 10^0 = 70000 + 5000 + 600 + 40 + 5$$

Are you getting the gist of it? The base of the number system represents the multiplier raised to the exponent "x" depending on the position of the digit. Let's try some binary!

Remember we said that the binary number system is base 2. This means that the multiplier is 2 raised to the exponent "x" depending on the position of the digit. Let's try some:

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 0 + 1 = 13_{10}$$

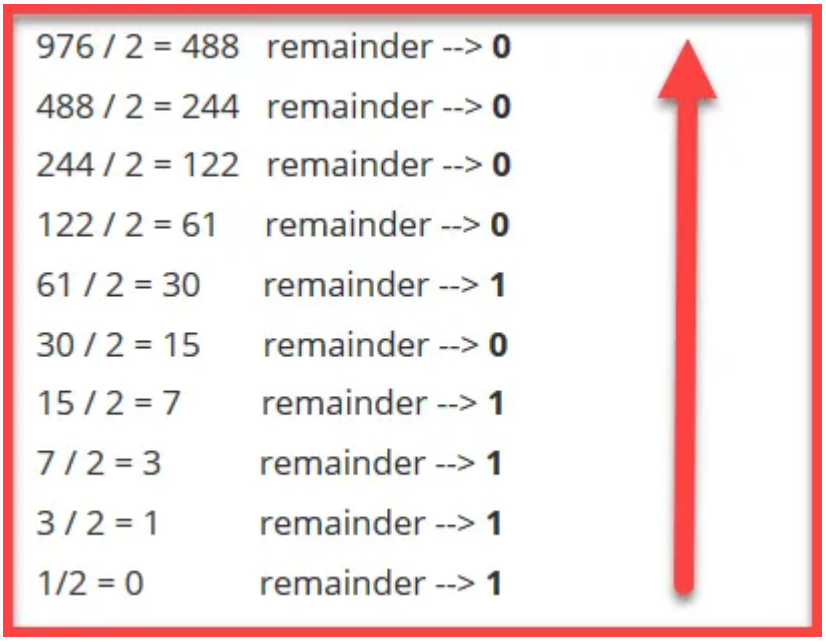
Let's try one that's a little harder:

$$\begin{aligned} &110101101_2 \\ &= 1 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 256 + 128 + 0 + 32 + 0 + 8 + 4 + 0 + 1 \\ &= 429_{10} \end{aligned}$$

So this shows us a nice convenient way to convert a binary number to a decimal number, but how about the other way around? What if we need to convert a decimal number to a binary number??? We must divide and conquer!

To convert a decimal number to a binary number we must divide by the base of the number we are converting to. In the case of binary this of course is 2. The remainder, either a 0 or 1 becomes the number in the binary sequence. Let's try one!

Convert 976**10** to binary. Start by dividing the initial number (976) by 2, then continue dividing the resultant number by 2 until the result is 0.



Since the result is 0, we are done!

The key is to **NOT** to read the binary string top to bottom, but to **read the resultant binary string bottom to top**. Therefore the correct answer: 976**10** = 1111010000**2**

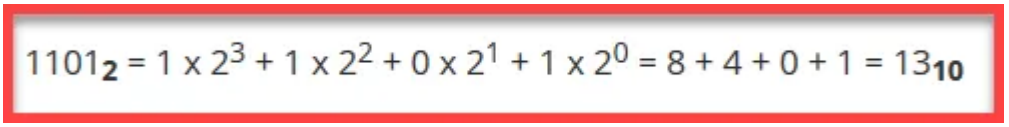
PLC Basics – The Octal System

The octal system is quickly disappearing, however, if you encounter an 8-bit architecture PLC such as the Allen-Bradley PLC 5, then knowing octal will be an asset.

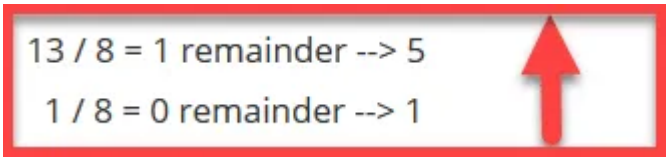
Luckily what we've learned so far is going to serve us well here too. As discussed the octal system is a base 8 number system. This means that permissible numbers will be between 0..7.

To convert a binary number to octal requires a 2-step process. First, convert the binary number to it decimal (base 10) equivalent, then using the same "divide and conquer" technique used above, convert the decimal equivalent to octal. Only this time, instead of dividing by 2 we will be dividing by 8.

Let's use the same example we did above and convert 1101**2** to its octal equivalent. Therefore,



Now we divide by 8 to find the octal equivalent,



As we did before, since the result is 0 we need not perform any more operations and we read the resultant octal string top to bottom.

Therefore, 1101**2** = 15**8**

This same algorithm can be used to compute more complex binary conversions to octal strings.

PLC Basics – The Hexadecimal System

Next to binary, hexadecimal is probably the next most important number system you should be comfortable with. The hexadecimal system uses a base 16 number system, with integer values 0..9 and letter **A=10, B=11, C=12, D=13, E=14 and F=15**.

Hexadecimal numbers are commonly used in computing because they can express every byte as two consecutive hexadecimal digits versus eight bits. This is useful when identifying memory locations and is much more readable for humans.

For example, the binary byte 11101101**2** can be expressed in hexadecimal format by separating the binary string into its 4-bit nibbles. Then convert the 4-bit nibbles into its decimal (base 10) equivalent. Once it's in its decimal (base 10) form, convert it to its hexadecimal (base 16) equivalent. Let's try it!

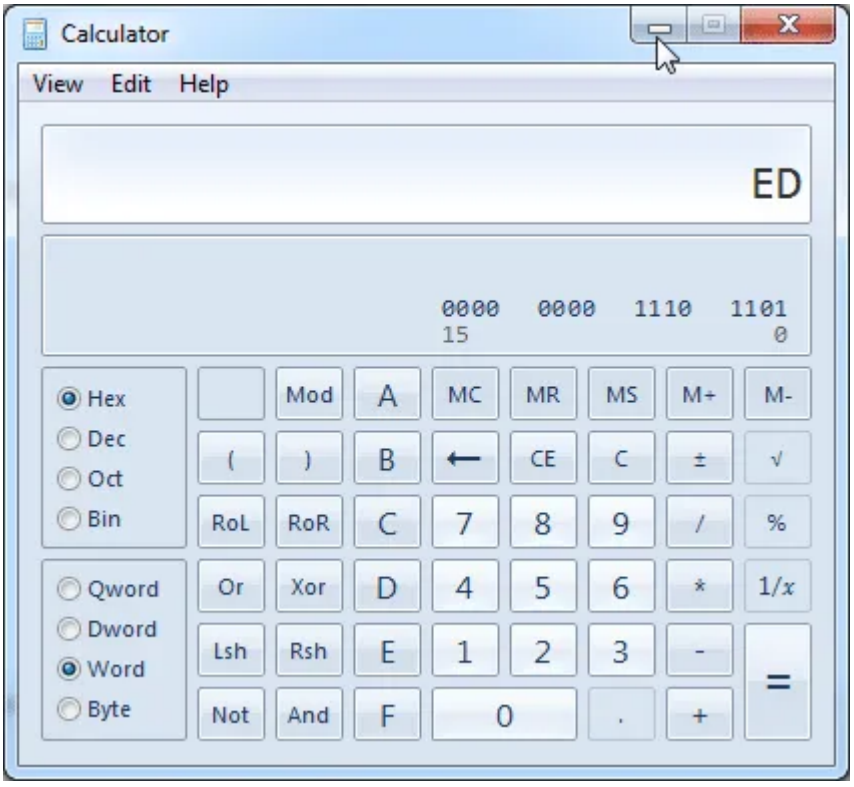
Convert 11101101**2** to its hexadecimal equivalent.

First, separate the binary string into its 4-bit nibbles: 1110 1101

Now treat each nibble separately, namely:
High-order nibble: 1110**2** = 14**10** = E**16**
Low-order nibble: 1101**2** = 13**10** = D**16**

Therefore, 11101101**2** = ED**16**

This can be confirmed using the calculator on your computer in "Programmer" mode as seen below.



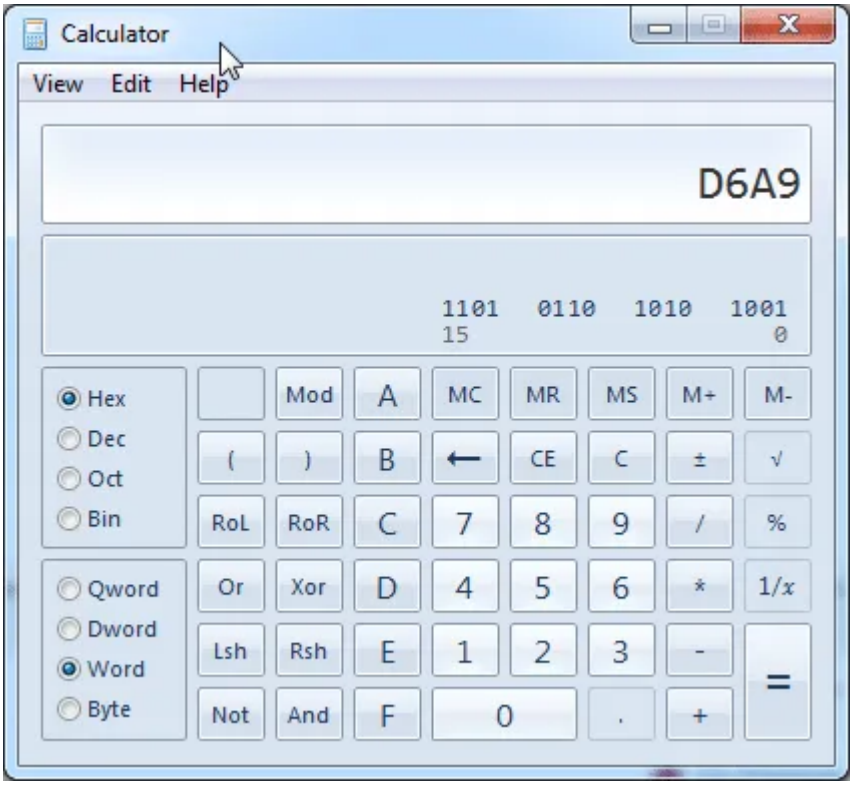
Because hexadecimal form is used so extensively in PLC's and computing let's try a more difficult example and then verify it using our calculator as we did above.

Convert 11010110101010012 to its hexadecimal equivalent.

First, separate the binary string into its 4-bit nibbles: 1101 0110 1010 1001

Now treat each nibble separately, namely:
11012 = 1310 = D16
01102 = 610 = 616
10102 = 1010 = A16
10012 = 910 = 916

Therefore, 11010110101010012 = D6A916



PLC Basics – A Quick Word about BCD…

BCD is known as **B**inary **C**oded **D**ecimal. It is very similar in structure to the hexadecimal system just discussed with some one difference. The binary coding limits us to using only number 0 through 9.

This type of encoding was widely used for devices such as Thumbwheel switches, 7-Segment Displays and Encoders.

We can use the same convention we used to “break-up” a 16-bit binary string for hexadecimal conversion to convert an binary number to its BCD equivalent.

Example: Convert the binary string 1001011100010011 to BCD format.

First we parse the 16-bit binary string into its 4-bit nibbles as follows,

1001 0111 0001 0011

Then computing each 4-bit nibble to its decimal equivalent:

1001 = **9**
0111 = **7**
0001 = **1**
0011 = **3**

Therefore, the BCD equivalent is **9713**. Binary Coded Decimal was created to provide a more human readable format than hexadecimal by limiting the usable values between 0..9 and omitting the letters A..F.

PLC Basics – Floating Point Numbers

Floating point numbers (also known as "Real" or "Decimal" numbers) give us the ability to represent fractional numbers with a finite precision. Depending on the architecture of your PLC, i.e., 16-bit or 32-bit, the large the memory the greater it will allow you to better approximate the fractional number you wish to represent.

Nearly all computers today follow the the **IEEE 754 standard** for representing floating-point numbers. This standard was largely developed by 1980 and it was formally adopted in 1985, though several manufacturers continued to use their own formats throughout the 1980's.

It should be noted that some numbers go on to infinity, for example pi. This number never ends, so it should be clear that while we can approximate these numbers to a high precision, that it is only an approximation.

PLC Basics – Discrete Inputs and Outputs

Discrete Inputs (Digital Inputs)

Discrete Input interface modules connects field input devices of the ON/OFF nature. This classification of I/O is related to *bit oriented* inputs and outputs – simply put – they can be described in the controllers memory using a 1 or 0.

Common voltage sources are 120 VAC and more commonly 24 VDC. Discrete PLC modules will specify whether it will accept AC, DC or both AC and DC, therefore, careful selection is required when sourcing these modules. A digital input card will receive an electrical signal (high or low) that will be interpreted as either ON or OFF.

Examples of Discrete Inputs are:

- Limit switches
- Proximity switches
- Pushbuttons
- Selector Switches

Discrete Outputs (Output Devices)

Discrete Output interface modules connect field output devices of the ON/OFF nature. A digital output module with either turn a device ON or OFF based on the logic that is controlling it and the input states that it depends on.

Examples of Discrete Outputs are:

- Control Relays
- Motor Contactors
- Pilot Lights
- Solenoid Valves

PLC Basics – Analog Inputs and Outputs

Analog Inputs

Analog Input interface modules convert a voltage or current (e.g. a signal that can be anywhere from 0 to 20mA) into a digitally equivalent number that can be understood by the CPU through an Analog- Digital Conversion (ADC) method known as *Quantization*. Analog input signals to a PLC can vary continuously over a range of voltage or current, providing detailed information about connected devices.

To input an analog voltage (into a PLC or any other computer) the continuous voltage value must be sampled and then converted to a numerical value by an ADC.

The time required to acquire the sample is called the sampling time. ADCs can only acquire a limited number of samples per second. The time between samples is called the sampling period, *T*, and the inverse of the sampling period is the sampling frequency (also called sampling rate).

The sampling time is often much smaller than the sampling period. The sampling frequency is specified when buying hardware, but for a PLC a maximum sampling rate might be 20Hz.

Resolution is another term you will often hear in the field when dealing with analog type instruments or sensors. Resolution is defined as the the smallest signal that can be represented by the ADC, or,

Resolution = Full Scale Value / 2n

Where n is the number of bits allocated to the conversion, which in most cases will be 16-, or 32-bits depending on your controller.

Example: If the analog input module you are using has a Full Scale Value = 10V, and n = 16 bits, then,
Resolution = 10V / 216 = 0.00015258789 V

This means we can be accurate, or detect a change in voltage to within 0.00015258789 V. Examples of Analog Inputs are:

- Linear Variable Displacement Transducers (LVDTs)
- Thermocouples
- Resistance Temperature Sensors (RTDs)
- Flow Sensors
- Potentiometers

Analog Outputs

Analog Output interface modules will convert a digital number sent by the CPU to it's real world voltage or current. Typical outputs signals can range from -10 VDC to +10 VDC, or 0-20mA and are used to drive mass flow controllers, pressure regulators and position controls.

Analog outputs are much simpler than analog inputs. To set an an analog output an integer is converted to a voltage. This process is very fast, however, analog outputs are subject to known as quantization errors. Examples of Analog Outputs are:

- Proportional Valves
- Servo Motors
- Heaters

Analog outputs control various output devices like motors, valves, and heaters, playing an essential role in managing industrial processes.

PLC Basics – Logic Gates & Boolean Expressions

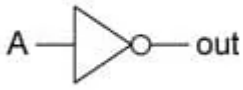
Logic gates in their basic form are electronic circuits that operate on one or more inputs to produce an output signal. They are the fundamental building blocks of any digital circuit. Most logic gates will accept two inputs and determine one output, however there are a few exceptions to this.

Let's focus our attention on the most common logic gates that will translate directly into PLC programming and Ladder Logic.

To evaluate logic gates is sometime useful to use something known as a Truth Table. Truth tables allow us to evaluate every combination of a logic gate or the combination of many logic gates built into a circuit.

The NOT function

The NOT function is perhaps the simplest of all gates. It's only purpose is to invert or flip whatever the input signal is to the output. For example, if the input is 1 the output is 0 and vice versa, if the input is 0 the output will become 1.

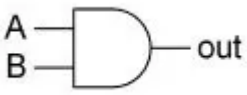


Input A	Output
0	1
1	0

The Ladder Logic equivalent of a NOT gate is:

The AND Function

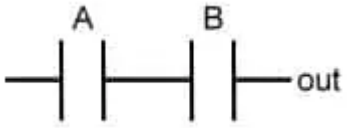
The AND function is a very practical uses in PLC programming. The AND function will output the AND'd result of two or more inputs on its input pins. Meaning, **the output will be true only when both inputs are true.** Let's look at this symbol little closer.



The truth table for the AND gate is:

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

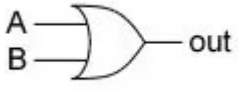
The Ladder Logic equivalent of the AND gate is:



It is very clear from the truth table that the output will only be true (1) when both inputs A and B are true (1). Please also take note how this is represented in Ladder Logic, it is read, "if A and B are true, then the output is true".

The OR Function

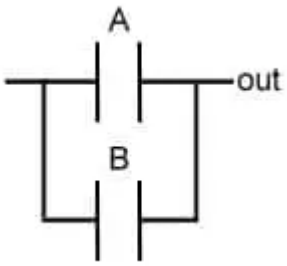
The OR function is another function that has very practical uses in PLC programming. The OR function will yield a true (1) output **when either A or B is true, or when both A and B are true**. Let's take a closer look at the OR function.



The truth table for the OR gate is:

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

The Ladder Logic equivalent of the OR gate is:



Looking at the truth table for the OR function it is very clear that when either A OR B are the output is also true. In addition the output will be true if both A AND B are true as well. So in short, as long as at least one of the inputs are true, the output will be true.

Also take not of how we represent a logical OR condition in the PLC using Ladder Logic. An OR condition is created by “branching” the inputs around each other as illustrated.

PLC Basics – Combining AND/OR Functions with a NOT Function

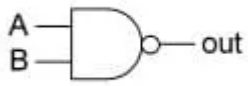
The NAND Function

This is where things start to get interesting. If we combine the AND function with the NOT function we get something called a NAND (NOT AND) function. I know, if things weren’t confusing enough right! As it turns the NAND gate has an important role in the PLC world.

Let’s first look at the truth table to help clarify:

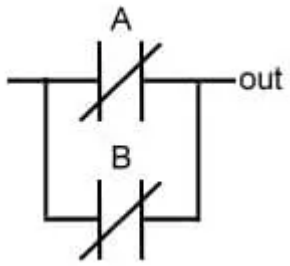
Input A	Input B	Output
0	0	1
1	0	1
0	1	1
1	1	0

Observing the truth table above we see that the output of the NAND function is essentially the inverted output of the AND function. So long as A **AND** B are NOT true, the output is true!



Notice the little circle at the end of what is the AND gate. That little circle is what differentiates the NAND gate from the AND gate letting the engineer know that this is NOT AND or NAND gate.

The Ladder Logic equivalent of the NAND gate is:



Pay particular attention to the Ladder Logic equivalent of the NAND. To build the NAND logic we must place the A and B outputs in parallel (or branched) as we did for the OR. This time however, we are examining the NOT state of each input.

Hopefully that's clear!

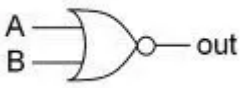
The NOR Function

This time if we combine the NOT and the OR function we get something called…you guessed it, the NOR function (NOT OR). Following the same reasoning as the NAND above, the output of the NOR function will yield the inverted result of the OR function. Let's take a look!

The truth table for the NOR function is:

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

Examining the truth table it is clear that the output for the NOR function is precisely the inverted output of the OR.



Similar to the NAND gate above, the little circle at the end of the OR is what tells us that it is NOT OR or the NOR gate.

The Ladder Logic equivalent of the NOR gate is:

PLC Basics – Let's Get Exclusive!

To complete our discussion of logic gates and boolean circuits there are two more functions we need to discuss. These functions are the XOR (Exclusive OR) and the XNOR (Exclusive NOT OR).

The XOR Function

Believe it's not as bad as at seems! The Exclusive OR, or XOR is one that we use almost every day in our decision making process. Let me explain.

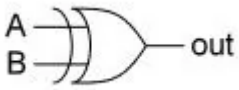
Imagine you are on a plane an the stewardess asks you if you would like a coffee **or** a tea. Let's add one further constraint to this in that you are only allowed one free beverage, **so your choice is either coffee OR tea BUT NOT both!**

That is how the XOR works, if A is true (1) OR B is true (1) the output is true (1) so long as both A AND B are NOT true (1). This is why we like truth tables…a picture says a 1000 words!

The truth table for the XOR function is:

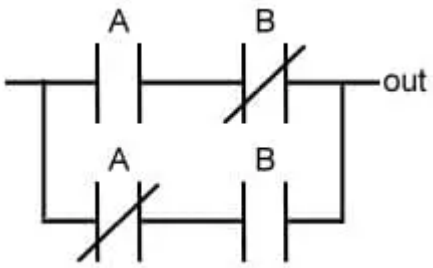
Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	0

As we said the output will be true (1), when either A OR B is true (1) but NOT when both A AND B are true (1).



Notice the additional curved line added to the OR gate. This indicates that it is the Exclusive OR (XOR).

The Ladder Logic equivalent of the XOR gate is:



Spend some time to think through the logic and hopefully it will be clear! This bit of logic is read as follows, “If A AND NOT B is true (1) then turn on the output”, OR, “If NOT A AND B is true (1), then turn on the output”. Notice that only one of these statements can be true at any given time!

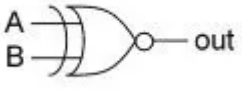
The XNOR Function

Last in our list of logic gates but certainly not least is the XNOR function (Exclusive NOT OR). If you’ve been very studious thus far you can probably predict what the output of this function is going to be???

If you said the inverted output of the XOR function then you would be absolutely right! The XNOR is precisely the inverted output of the XOR function. Let’s take a look!

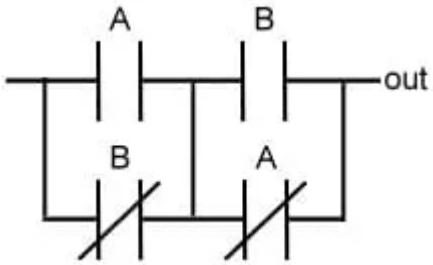
The truth table for the XNOR function is:

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



Notice that it looks very similar to the XOR gate with the addition of the circular NOT symbol on the output.

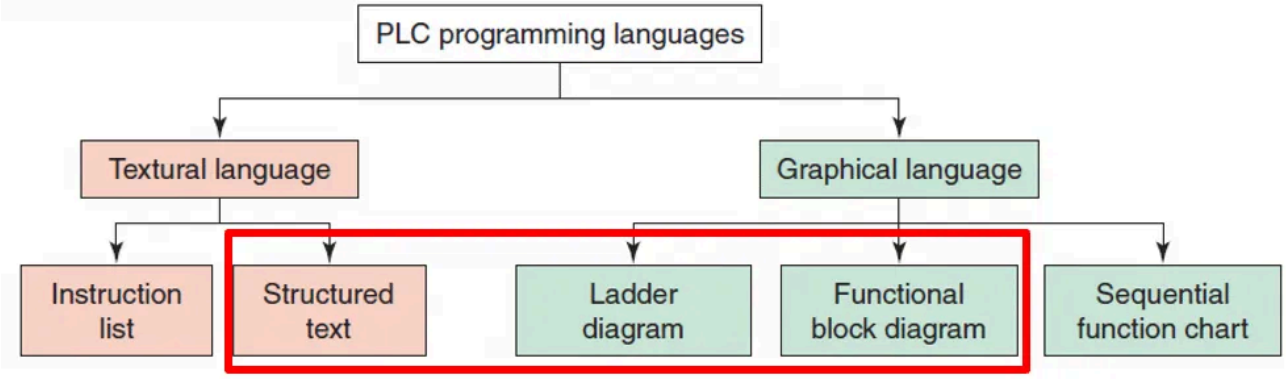
The Ladder Logic equivalent of the XNOR gate is:



PLC Basics – Programming Languages

PLC programming involves "downloading" a compiled sequence of binary coded numbers into the PLC system. There are various ways we can perform PLC programming tasks, however, we will focus on 3 of the most common ways indicated by a red box in the image below.

A programming device is essential for writing, entering, and monitoring the PLC's program. It communicates directly with the CPU and ensures the system functions correctly.



PLC Programming Methods

We will focus on three in particular: Ladder Logic, Function Block, Structured Text or Statement Logic

PLC Basics – Introducing Ladder Logic Programming

The most common PLC Programming "language" is something referred to as Ladder Logic. Ladder Logic has evolved from the days of controlling machines or processes using electro-mechanical relay devices or "relay logic" as it is referred.

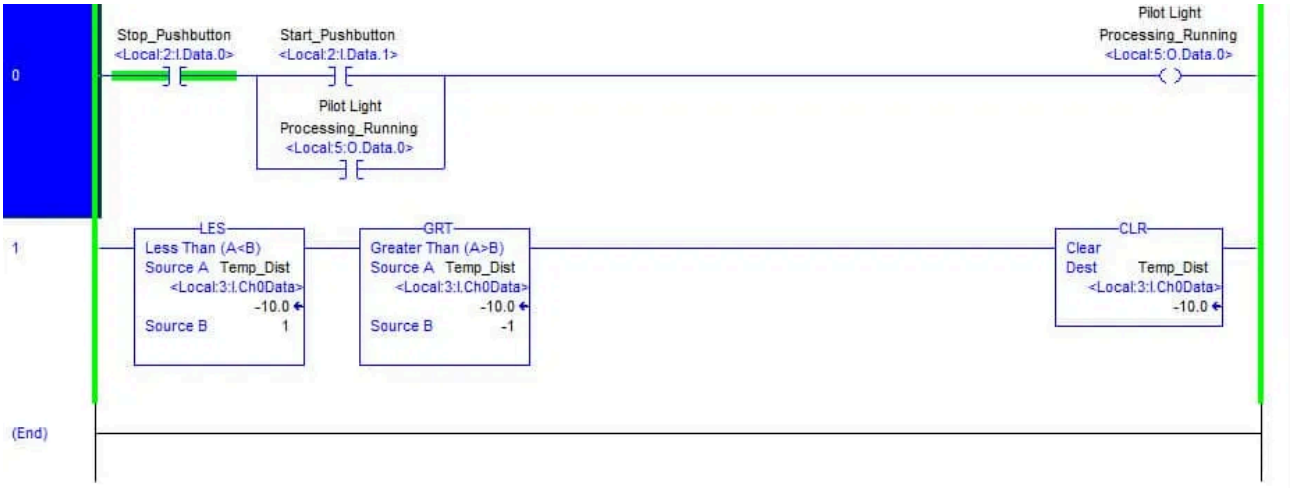
It is a graphical representation of the "relay contacts" that would have been used in a relay controlled system. Each rung of ladder typically has one coil at the far right and then logical "input contacts" to the left.

-[ ]- Normally open (Examine if Closed) contact.

-[ ]- Normally closed (Examine if Opened) contact.

-( )- Output Coil.

Here is a simple Start/Stop circuit with some compare instructions written in Ladder Logic.

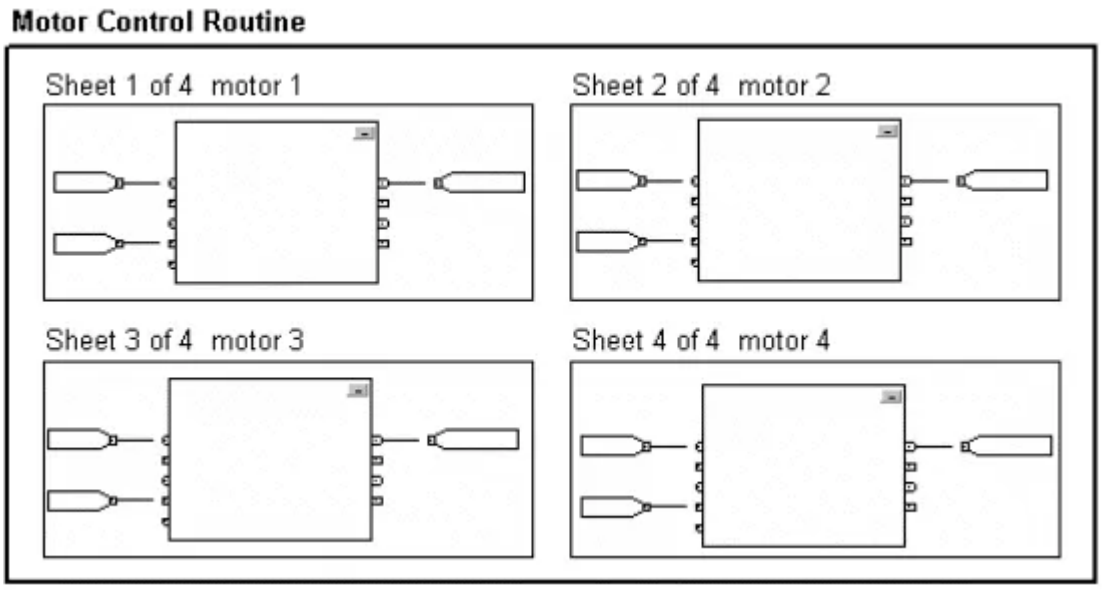


The input contacts are then arranged in a logical AND, OR type configuration to turn on an output as was discussed in the previous section. We provide a comprehensive PLC instruction set list below so keep reading!

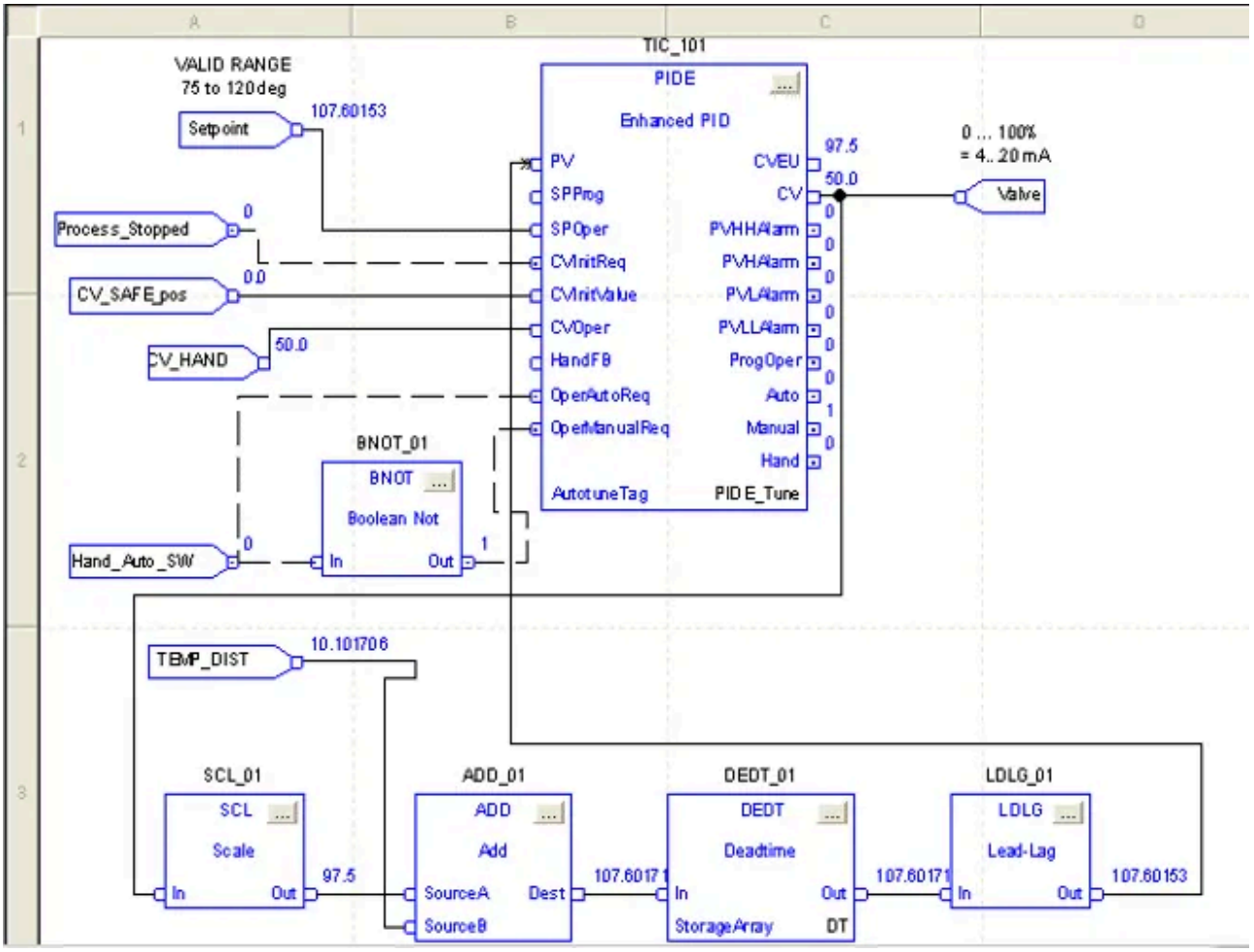
PLC Basics – Introducing Function Block Programming

Another common way to program a PLC is using Function Block. A Function Block Diagram (FBD) is a graphical depiction of process flow using simple and complex interconnecting blocks. FBD's are typically organized into multiple "sheets" allowing one to organize the instruction sets – typically one sheet per device.

See below a sample function block routines for 4 different motor controllers.



It's important to note that once the routine is executed all sheets in the list are executed as well. Below is a Function Block program created using Studio 5000 software, making use of a PIDE (Proportional Integral Derivative Enhanced) instruction to control a closed loop process. You can see that the instruction effectively get "wired up" using the various input and output tags.



PLC Basics – Introducing Structured Text Programming

Structured text is a textual programming language that uses statements to define what to execute. It follows more traditional programming conventions that you would find with procedural type languages, however, typically it is not case sensitive.

A series of statements (logic) is composed by formulating assignments and relationships using various operators as depicted.

Order	Operation
1.	( )
2.	function (…)
3.	**
4.	- (negate)
5.	NOT
6.	*, /, MOD
7.	+, - (subtract)
8.	<, <=, >, >=
9.	=, <>
10	&, AND
11.	XOR
12.	OR

In most cases where I've seen this type of PLC programming language used is when the programmer needs to handle those types of functions or operations that can't be defined clearly or efficiently using the other two methods.

One word of caution – as a PLC programmer we always have to be cognoscente of who will be maintaining our programs after the project is complete. In my experience, this often will be left to maintenance staff, namely, electrical maintenance.

This is why ladder logic programming is by far the most common language used in industry today, because as we mentioned early it evolves from the relay logic circuits of old that electricians in most facilities are familiar with.

PLC Basics – Common Instruction Sets

This is largely dependent on the PLC programming platform you are using, however, most PLC's today will include the following instruction sets or groups and this is not an exhaustive list by any means:

- **Bit Instructions** – Binary type ON/OFF instructions XIC, XIO, OTL, OTU, OTE, ONS
- **Timer/Counter Instructions** – TON, TOF, RTO, CTU, CTD, RES
- **Message/System Instructions** – MSG, GSV, SSV
- **Compare Instructions** – CMP, LIM, MEQ, EQU, LES, GRT, LEQ, GEQ
- **Math Instructions**– CPT, ADD, SUB, MUL, DIV, MOD, SQR, NEG, ABS
- **Move/Logical Instructions** – MOV, MVM, AND, OR, XOR, NOT, SWPF, CLR, BTD
- **File Manipulation Instructions** – FAL, FSC, COP, FLL, AVE, SRT, STD, SIZE, CPS
- **File/Shift Instructions** – BSL, BSR, FFL, FFU, LFL, LFU
- **Sequencer Instructions** – SQI, SQO, SQL
- **Program Control Instructions** – JMP, LBL, JSR, JXR, RET, SBR, TND, MCR, FOR, BRK
- **Motion Instructions** – for PLC's that support servo motion

- **Advanced Math/Trig** – for PLC's that support advanced mathematical operations

Compact or integrated PLCs, also known as unitary PLCs, are designed for straightforward applications with a fixed number of I/O points and an integrated CPU. Unitary PLCs, also known as compact PLCs, are the simplest type of PLC.

PLC Basics – Common PLC Acronyms

The following table shows a list of the some common acronyms or abbreviations you will see and hear when researching programmable logic controllers.

	Edit
Acronym	**Description**
ASCII	American Standard Code for Information Interchange
BCD	Binary Coded Decimal
CSA	Canadian Standards Association
DCS	Distributed Control System
DIO	Distributed I/O
EIA	Electronic Industries Association
EMI	ElectroMagnetic Interferance
HMI	Human Machine Interface
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronic Engineers
I/O	Input(s) and/or Output(s)
ISO	International Standards Organization
LL	Ladder Logic
LSB	Least Significant Bit
MMI	Man Machine Interface
MODICON	Modular Digital Controller
MSB	Most Significant Bit
NEC	National Electrical Code

PID	Proportional Integral Derivative (feedback control)
RF	Radio Frequency
RIO	Remote I/O
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
TCP/IP	Transmission Control Protocol/Internet Protocol

Conclusion

In conclusion, understanding the fundamentals of Programmable Logic Controllers (PLCs) is essential for anyone involved in industrial automation and control systems. PLCs have transformed the manufacturing landscape by offering flexible, reliable, and cost-effective solutions for complex industrial processes.

From their basic architecture and components to advanced programming techniques like ladder logic and function block diagrams, PLCs provide versatile tools for managing inputs and outputs, executing control logic, and ensuring efficient operation of industrial systems.

As technology advances, the role of PLCs continues to expand, integrating with the Industrial Internet of Things (IIoT) and other modern automation technologies. By mastering PLC basics, you can unlock the potential of automation and drive innovation in your industrial applications.
