Programmable Logic Controller Training Course

PLC Fundamentals and Applications

Ali T. Shaheen

University of Baghdad
Electrical Eng. Dept.
2011

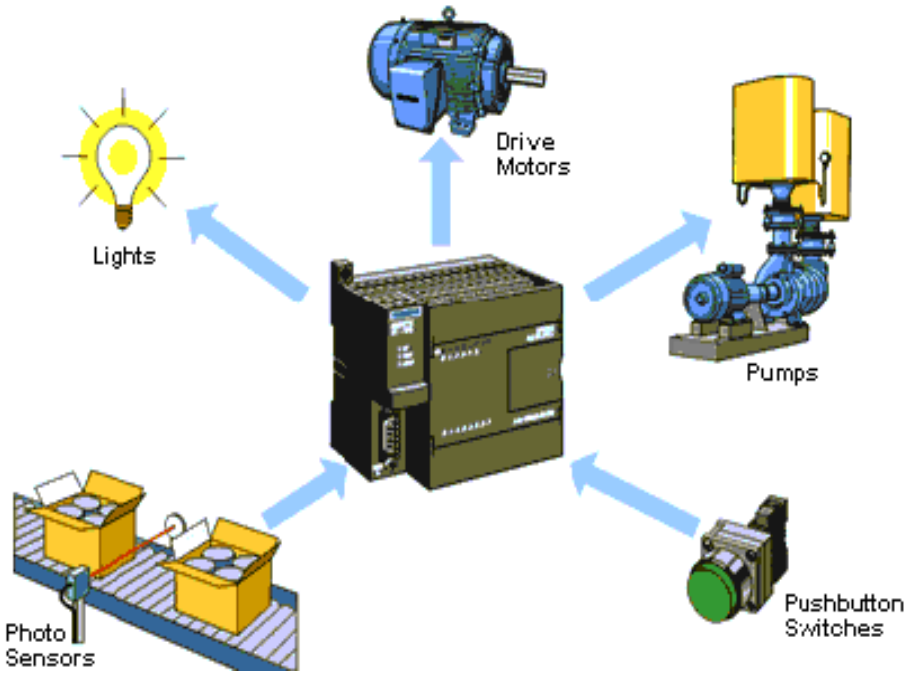
Lecture 1

Introduction to PLC and Types of Control System

A programmable controller, formally called the programmable logic controller (PLC) can be defined as a solid state device member of the computer family.

It is capable of storing instruction to implement control functions such as sequencing, timing, counting, arithmetic, data manipulation and communication to control industrial machines and processes.

- PLC can perform the same task as hard-wired devices
- Connections between field devices and relay contacts take place in the PLC
- Installation is less extensive
- Also more complex function.



History of PLC

During the Industrial Revolution of the 18th-and 19th-centuries, many traditionally manual processes were taken over by machines. These early machines relied on gears and pulleys to work and were, by our standards, extremely primitive. The first major breakthrough in the development of control systems came with the invention of electrically powered machines. The first control systems were developed in the early years of the 20th century and used sequential Relay Circuits for machine control. A major technical breakthrough in its day, and still used in some plants today, relay technology enabled machines to work faster and more safely.

Relay circuits performed their job very well, but they required large amounts of floor space, and huge amounts of energy. Adding to their drawbacks as the basis for a machine control system, relay circuits also took a long time to install, troubleshoot, and modify. Finally, in the early 1970s, a device was developed to replace sequential relay circuits: the Programmable Logic Controller (PLC).

As you will remember from reading about them in Module 24, PLCs are more reliable, faster, more flexible and more efficient than relay-based systems. For example, PLCs are cheaper and easier to wire and maintain than relays. Furthermore, when it comes to troubleshooting, PLCs are much quicker than relays at testing and debugging the program.

PLCs are used in all kinds of industries. In fact, almost any industrial process that uses electrical control needs a PLC. For example, let's assume that when a switch turns on we want to turn a solenoid on for 5 seconds and then turn it off regardless of how long the switch is on. We can do this with a simple

external timer. But what if the process included 10 switches and solenoids? We would need 10 external timers. What if the process also needed to count how many times the switches individually turned on?

We need a lot of external counters. With a PLC, however, we can dispense with those unwieldy timers and counters, and simply program the PLC to count its inputs and turn the solenoids on for the specified time.

Comparison of PLC with Other Control Systems :-

C\Cs	Relay systems	Digital Logics	Computers	PLC systems
Physical Size	**Bulky**	**Very Compact**	**Fairly Compact**	**Very Compact**
Operating Speed	**Slow**	**Very Fast**	**Fairly Fast**	**Fast**
Noise Immunity	**Excellent**	**Good**	**Fairly Good**	**Good**
Complex Operation	**None**	**Yes**	**Yes**	**Yes**
Ease of Changes	**Very Difficult**	**Difficult**	**Quite Simple**	**Very Simple**
Easy of Maintenance	**Poor-large No. Of Contacts**	**Poor if ICs Soldered**	**Poor-several Custom Boards**	**Good-few Standard Cards**

Advantages of PLCs: -

The same, as well as more complex tasks, can be done with a PLC. Wiring between devices and relay contacts is done in the PLC program. Hard-wiring, though still required to connect field devices, is less intensive. Modifying the application and correcting errors are easier to handle. It is easier to create and change a program in a PLC than it is to wire and rewire a circuit.

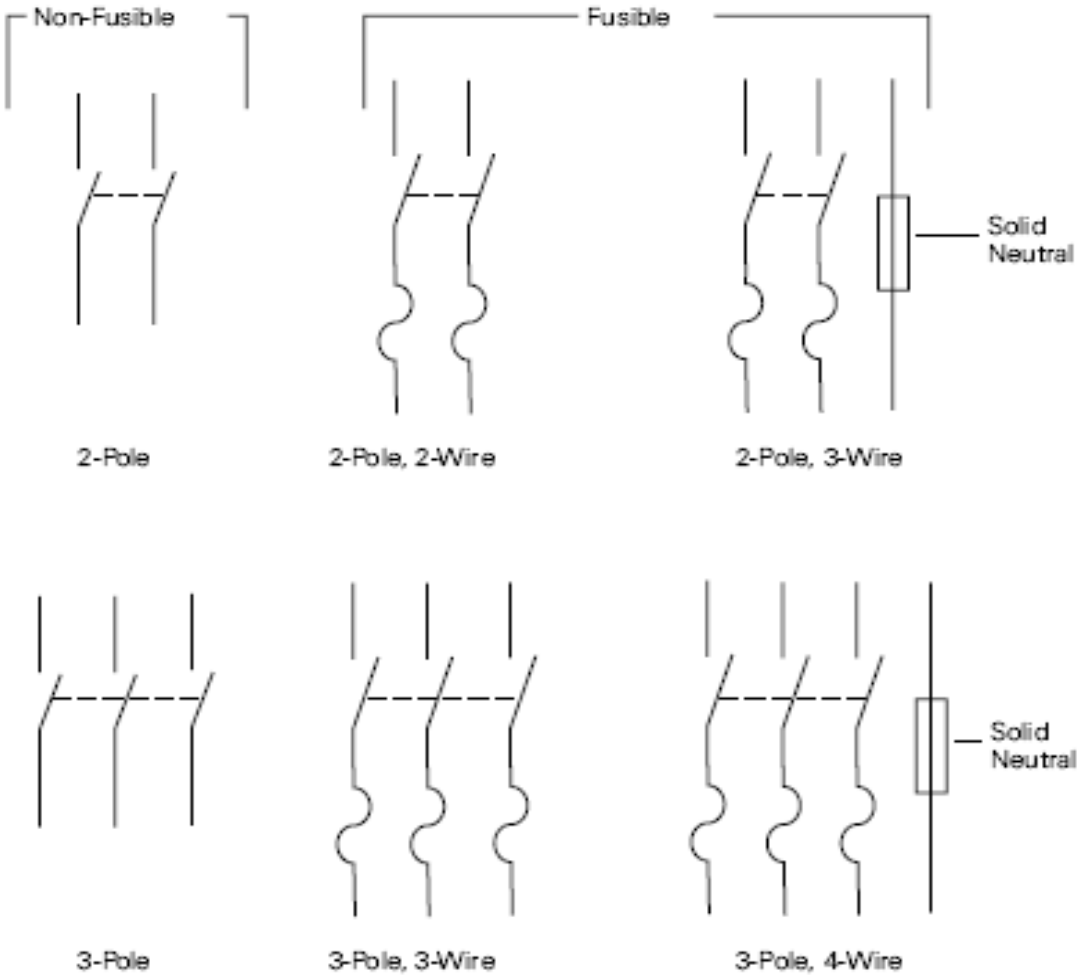
Following are just a few of the advantages of PLCs: -

- Smaller physical size than hard-wire solutions.
- Easier and faster to make changes.
- PLCs have integrated diagnostics and override functions.
- Diagnostics are centrally available.
- Applications can be immediately documented.
- Applications can be duplicated faster and less expensively.

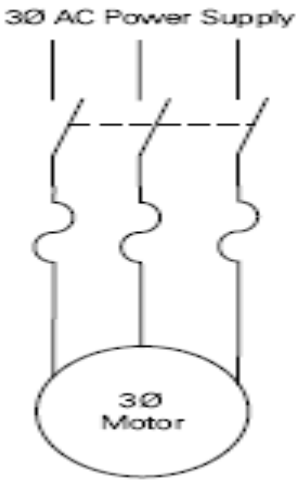
Basic elements of PLC and their functions

1.1- Switch Circuit Types : -

The Following diagrams are circuit configuration for 2- and 3-pole safety switches. Safety switches may be fusible, non-fusible, or fusible with a solid neutral.



The circuit configuration required depends on the load and on the power supply connected to it. For example, a three-phase motor needs a 3-pole switch to connect it to a three-phase power supply. If over current protection is required, a fusible 3-pole safety switch should be selected, as in the following example.



Selecting a Switch: -

There are three important features to consider when selecting a switch:

- Contacts (e.g. single pole, double throw)
- Ratings (maximum voltage and current)
- Method of Operation (toggle, slide, key etc.)

Switch Contacts: -

Several terms are used to describe switch contacts:

- Pole - number of switch contact sets.
- Throw - number of conducting positions, single or double.
- Way - number of conducting positions, three or more.
- Momentary - switch returns to its normal position when released.
- Open - off position, contacts not conducting.

- Closed - on position, contacts conducting, there may be several on positions.

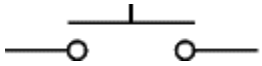
For example: the simplest on-off switch has one set of contacts (single pole) and one switching position which conducts (single throw). The switch mechanism has two positions: open (off) and closed (on), but it is called 'single throw' because only one position conducts.

Switch Contact Ratings: -

Switch contacts are rated with a maximum voltage and current, and there may be different ratings for AC and DC. The AC values are higher because the current falls to zero many times each second and an arc is less likely to form across the switch contacts.

For low voltage electronics projects the voltage rating will not matter, but you may need to check the current rating. The maximum current is less for inductive loads (coils and motors) because they cause more sparking at the contacts when switched off.

Standard Switches : -

Type of Switch	Circuit Symbol	Example
ON-OFF Single Pole, Single Throw = SPST A simple on-off switch. This type can be used to switch the power supply to a circuit. When used with mains electricity this type of switch *must* be in the live wire, but it is better to use a DPST switch to isolate both live and neutral.		 SPST toggle switch
(ON)-OFF Push-to-make = SPST Momentary A push-to-make switch returns to its normally open (off) position when you release the button, this is shown by the brackets around ON. This is the standard doorbell switch.		 Push-to-make switch
ON-(OFF)		 Push-to-break switch

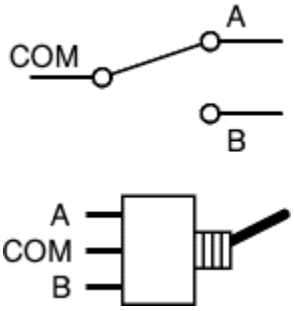
Push-to-break = SPST Momentary

A push-to-break switch returns to its normally closed (on) position when you release the button.

ON-ON
Single Pole, Double Throw = SPDT

This switch can be on in both positions, switching on a separate device in each case. It is often called a changeover switch. For example, a SPDT switch can be used to switch on a red lamp in one position and a green lamp in the other position.

A SPDT toggle switch may be used as a simple on-off switch by connecting to COM and one of the A or B terminals shown in the diagram. A and B are interchangeable so switches are usually not labeled.



SPDT toggle switch

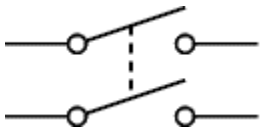
SPDT slide switch
(PCB mounting)

ON-OFF-ON
SPDT Centre Off
A special version of the standard SPDT switch. It has a third switching position in the centre which is off. Momentary (ON)-OFF-(ON) versions are also available where the switch returns to the central off position when released.

SPDT rocker switch

Dual ON-OFF
Double Pole, Single Throw = DPST

A pair of on-off switches which operate together (shown by the dotted line in the circuit symbol).



DPST rocker switch

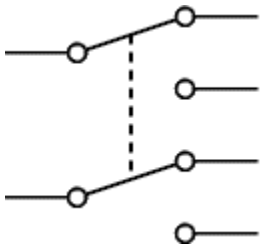
A DPST switch is often used to switch mains electricity because it can isolate both the live and neutral connections.

Dual ON-ON
Double Pole, Double Throw = DPDT

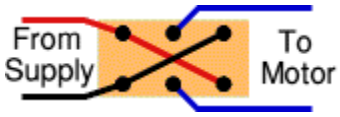
A pair of on-on switches which operate together (shown by the dotted line in the circuit symbol).

DPST slide switch

A DPDT switch can be wired up as a reversing switch for a motor as shown in the diagram.



ON-OFF-ON
DPDT Centre Off
A special version of the standard SPDT switch. It has a third switching position in the centre which is off. This can be very useful for motor control because you have forward, off and reverse positions. Momentary (ON)-OFF-(ON) versions are also available where the switch returns to the central off position when released.



Wiring for Reversing Switch

Special Switches : -

Type of Switch **Example**

Push-Push Switch (e.g. SPST = ON-OFF)

This looks like a momentary action push switch but it is a standard on-off switch: push once to switch on, push again to switch off. This is called a latching action.

Micro switch (usually SPDT = ON-ON)

Micro switches are designed to switch fully open or closed in response to small movements. They are available with levers and rollers attached.

Key switch

A key operated switch. The example shown is SPST.

Tilt Switch (SPST)

Tilt switches contain a conductive liquid and when tilted this bridges the contacts inside, closing the switch. They can be used as a sensor to detect the position of an object. Some tilt switches contain mercury which is poisonous.

Reed Switch (usually SPST)

The contacts of a reed switch are closed by bringing a small magnet near the switch. They are used in security circuits, for example to check that doors are closed. Standard reed switches are SPST (simple on-off) but SPDT (changeover) versions are also available.

Warning: reed switches have a glass body which is easily broken!

DIP Switch (DIP = Dual In-line Parallel)

This is a set of miniature SPST on-off switches, the example shown has 8 switches. The package is the same size as a standard DIL (Dual In-Line) integrated circuit.

This type of switch is used to set up circuits, e.g. setting the code of a remote control.

Multi-pole Switch

The picture shows a 6-pole double throw switch, also known as a 6-pole changeover switch. It can be set to have momentary or latching action. Latching action means it behaves as a push-push switch, push once for the first position, push again for the second position etc.

Multi-way Switch

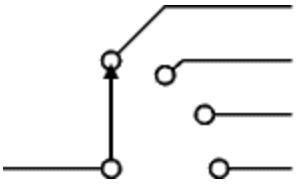
Multi-way switches have 3 or more conducting positions. They may have several poles (contact sets). A popular type

has a rotary action and it is available with a range of contact arrangements from 1-pole 12-way to 4-pole 3 way.

The number of ways (switch positions) may be reduced by adjusting a stop under the fixing nut. For example if you need a 2-pole 5-way switch you can buy the 2-pole 6-way version and adjust the stop.

Contrast this multi-way switch (many switch positions) with the multi-pole switch (many contact sets) described above.

Multi-way rotary switch



1-pole 4-way switch symbol

Sensors:-

Generally there are 5 steps to determine which switch type is best suited to the application. This depends on the material properties of the target to be detected.

Step (1) : - type of sensor .

Step (2) : - Housing design .

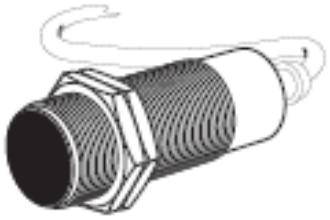
Step (3) : - Sensing range (mm)

Step (4) : - Electrical data and connections

Step (5) : - General specifications

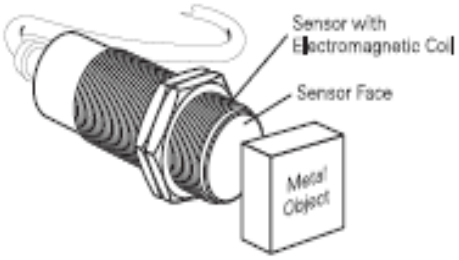
- **Proximity Sensor:**

A type of sensing switch that detects the presence or absence of an object without physical contact



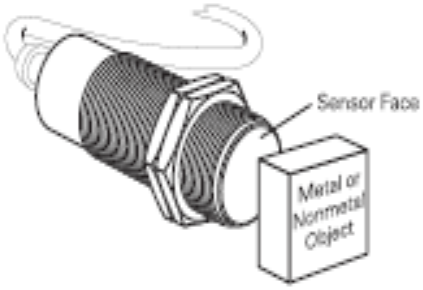
- Inductive Proximity Sensor:-

A type of *sensing switch* that uses an electromagnetic coil to detect the presence of a metal object without coming into physical contact with it, Inductive proximity sensors ignore nonmetallic objects.



- Capacitive Proximity Sensor :-

A type of *sensing switch* that produces an electrostatic field to detect the presence of metal and nonmetallic objects without coming into contact with them

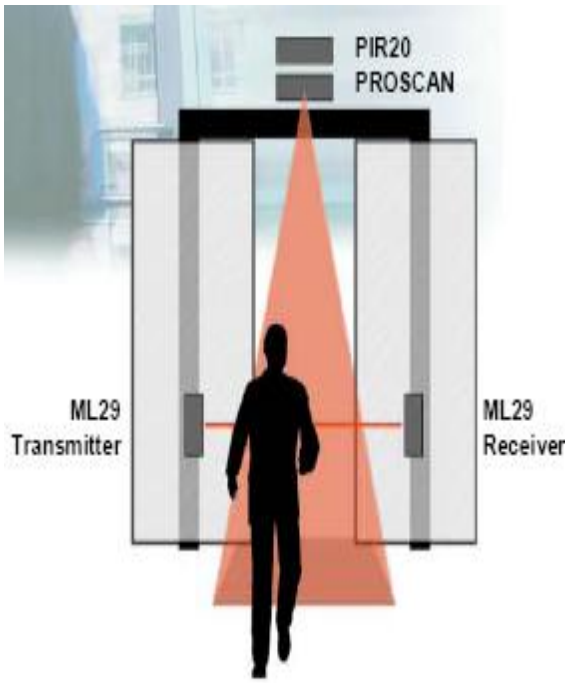


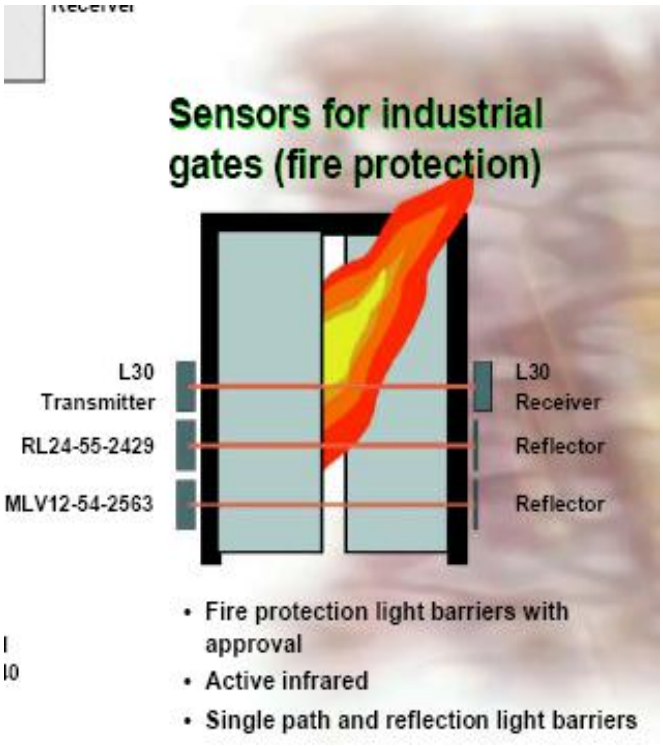
- Ultrasonic Sensor

A type of sensing switch that uses high frequency sound to detect the presence of an object without coming into contact with the object

- Photoelectric Sensor : -

Recognition, detection, positioning, classification, counting, notification and monitoring. Nowadays, these processes are largely handled by non-contact photoelectric sensors. Applications range from the automobile industry, mechanical engineering, and assembly automation, through warehousing and conveyor systems and packaging applications, to the printing and paper industries, and naturally include monitoring and safety systems.


PIR20
PROSCAN
ML29
Transmitter
ML29
Receiver

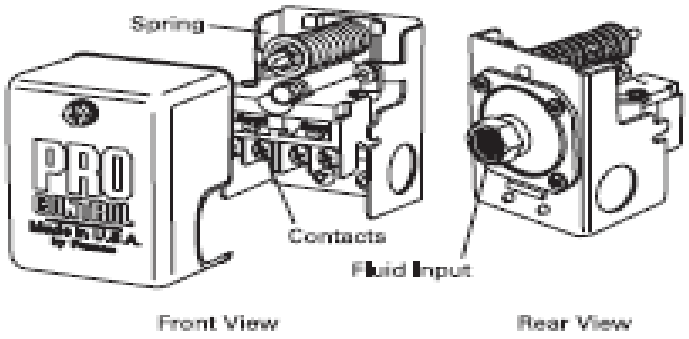

Sensors for industrial gates (fire protection)
L30
Transmitter
RL24-55-2429
MLV12-54-2563
L30
Receiver
Reflector
Reflector
• Fire protection light barriers with approval
• Active infrared
• Single path and reflection light barriers


Proscan-T
FLT-8-HW
ML29T


Sensors for automatic doors in buses and trains
ML29T

- Pressure Switch : -

A control device that opens or closes its contacts in response to a change in the pressure of a liquid or gas



- Sensing Switches :-

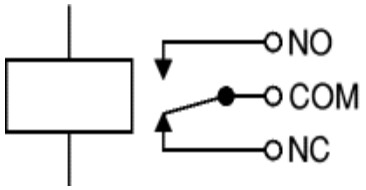
A device, often called a sensor, used to provide information on the presence or absence of an object. Examples include a limit switch, photoelectric sensor, inductive proximity sensor, capacitive proximity sensor, and ultrasonic proximity sensor.

Sensors	Advantages	Disadvantage	Applications
Limit Switch	• **High Current Capability** • **Low Cost** • **Familiar " Low-Tech " Sensing**	• **Require Physical Contact** • **Very Slow Response** • **Contact Bounce**	• **Interlocking** • **Basic End Travel Sensing**

Photoelectric	• **Senses all Kinds of Materials** • **Long Life** • **Largest Sensing Range** • **Very Fast Response Time**	• **Lens Subject to Contamination.** • **Sensing Range Affected by Color and Reflectivity**	• **Packaging** • **Material Handling** • **Parts Detection**
Inductive	• **Resistant to Harsh Environments** • **Very Predictable** • **Long Life** • **Easy to Install**	• **Distance Limitations** • **Senses Metal Only**	• **Industrial and Machines.** • **Machine Tools**
Capacitive	• **Can Detect Non-Metallic** • **Detects Through Some Containers**	• **Very Sensitive to Extreme Environmental Changes**	• **Level Sensing**
Ultrasonic	• **Senses all Materials**	• **Sensitive to Temperature Changes.**	• **Level Control** • **Doors** • **Anti-Collision**

Electromagnetic Relay : -

Relay is an electrically operated switch. Current flowing through the coil of the relay creates a magnetic field which attracts a lever and changes the switch contacts. The coil current can be on or off so relays have two switch positions and they are double throw (changeover) switches.

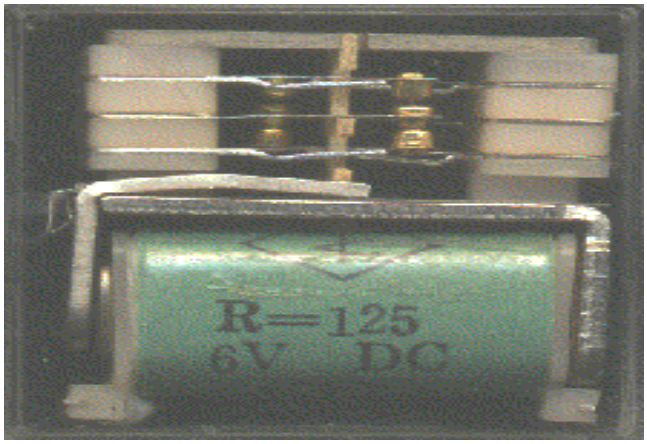


Circuit symbol for a relay

Relays allow one circuit to switch a second circuit which can be completely separate from the first. For example a low voltage battery circuit can use a relay to switch a 230V AC mains circuit. There is no electrical connection inside the relay between the two circuits, the link is magnetic and mechanical. The coil of a relay passes a relatively large current, typically 30mA for a 12V relay, but it can be as much as 100mA for relays designed to operate from lower voltages

Relays are usually SPDT or DPDT but they can have many more sets of switch contacts, for example relays with 4 sets of changeover contacts are readily available.

The animated picture shows a working relay with its coil and switch contacts. You can see a lever on the left being attracted by magnetism when the coil is switched on. This lever moves the switch contacts. There is one set of contacts (SPDT) in the foreground and another behind them, making the relay DPDT.



The relay's switch connections are usually labeled COM, NC and NO:

- COM = Common, always connect to this, it is the moving part of the switch.
- NC = Normally Closed, COM is connected to this when the relay coil is off.
- NO = Normally Open, COM is connected to this when the relay coil is on.

- Connect to COM and NO if you want the switched circuit to be on when the relay coil is on.
- Connect to COM and NC if you want the switched circuit to be on when the relay coil is off.

Choosing a relay : -

You need to consider several features when choosing a relay:

1. Physical size and pin arrangement

   If you are choosing a relay for an existing PCB you will need to ensure that its dimensions and pin arrangement are suitable. You should find this information in the supplier's catalogue.
2. Coil voltage

   The relay's coil voltage rating and resistance must suit the circuit powering the relay coil. Many relays have a coil rated for a 12V supply but 5V and 24V relays are also readily available. Some relays operate perfectly well with a supply voltage which is a little lower than their rated value.
3. Coil resistance

   The circuit must be able to supply the current required by the relay coil. You can use Ohm's law to calculate the current:

$$\text{Relay coil current} = \frac{\text{supply voltage}}{\text{coil resistance}}$$

4. For example: A 12V supply relay with a coil resistance of 400Ω passes a current of

30mA.

5. Switch ratings (voltage and current)
   The relay's switch contacts must be suitable for the circuit they are to control. You will need to check the voltage and current ratings. Note that the voltage rating is usually higher for AC, for example: "5A at 24V DC or 125V AC".

6. Switch contact arrangement (SPDT, DPDT etc)
   Most relays are SPDT or DPDT which are often described as "single pole changeover" (SPCO) or "double pole changeover" (DPCO)

example).

Advantages of relays:

- Relays can switch AC and DC, transistors can only switch DC.
- Relays can switch high voltages, transistors cannot.
- Relays are a better choice for switching large currents (> 5A).
- Relays can switch many contacts at once.

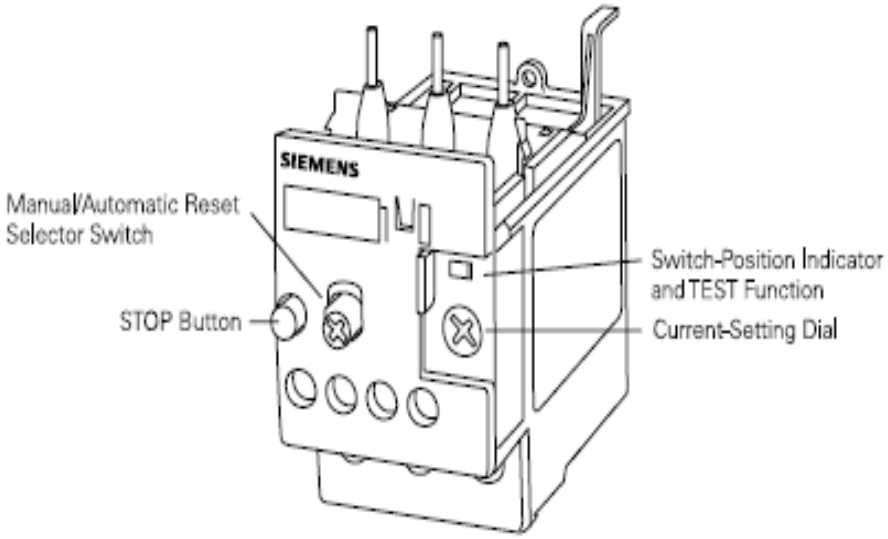
Disadvantages of relays:

- Relays are bulkier than transistors for switching small currents.
- Relays cannot switch rapidly (except reed relays), transistors can switch many times per second.
- Relays use more power due to the current flowing through their coil.
- Relays require more current than many chips can provide, so a low power transistor may be needed to switch the current for the relay's coil.

Relays can generate a very high voltage across the coil when switched off. This can damage other components in the circuit. To prevent this a diode is connected across the coil. The cathode of the diode is connected to the most positive end of the coil.

- Overload Relay

A device used to protect a motor from damage resulting from an *overcurrent*.



- Overcurrent

A *current* in excess of the rated current for a device or *conductor*. An overcurrent can result from an *overload*, *short circuit*, or *ground fault*.

- Overload

Can refer to an operating condition in excess of a full-load rating or a *current* high enough to cause damage if it is present long enough. An overload does not refer to a *short circuit* or *ground fault*.

Lecture 2

Digital Logic Concepts

Number systems

Since a PLC is a computer, it stores information in the form of On or Off conditions (1 or 0), referred to as binary digits (bits). Sometimes binary digits are used individually and sometimes they are used to represent numerical values.

Decimal System Various number systems are used by PLCs. All number systems have the same three characteristics: digits, base, weight. The decimal system, which is commonly used in everyday life, has the following characteristics: Ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 Base 10 Weights 1, 10, 100, 1000,

Binary System The binary system is used by programmable controllers. The binary system has the following characteristics:

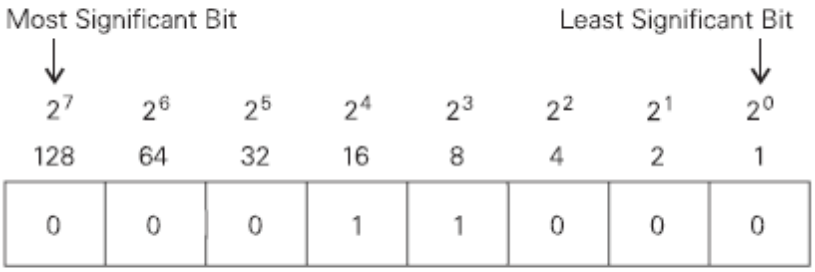
Two digits 0, 1

Base 2

Weights Powers of base 2 (1, 2, 4, 8, 16, ...)

In the binary system 1s and 0s are arranged into columns. Each column is weighted. The first column has a binary weight of

20. This is equivalent to a decimal 1. This is referred to as the least significant bit. The binary weight is doubled with each succeeding column. The next column, for example, has a weight of 21, which is equivalent to a decimal 2. The decimal value is doubled in each successive column. The number in the far left hand column is referred to as the most significant bit. In this example, the most significant bit has a binary weight of 27. This is equivalent to a decimal 128.

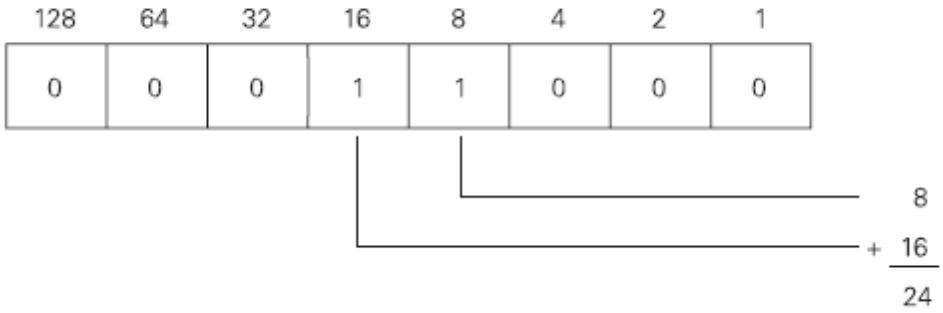


Converting Binary to Decimal

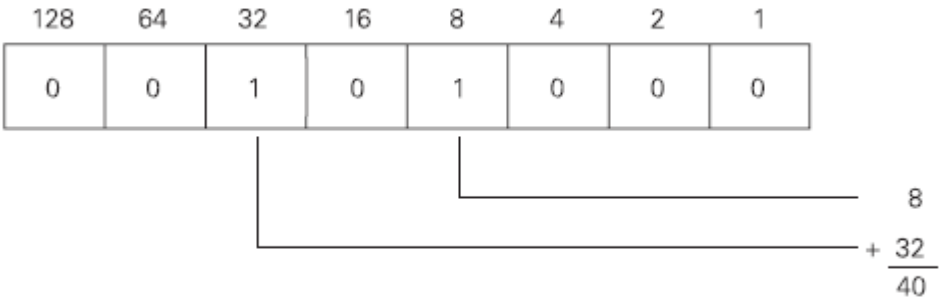
The following steps can be used to interpret a decimal number from a binary value.

1) Search from least to most significant bit for 1s.
2) Write down the decimal representation of each column containing a 1.
3) Add the column values.

In the following example, the fourth and fifth columns from the right contain a 1. The decimal value of the fourth column from the right is 8, and the decimal value of the fifth column from the right is 16. The decimal equivalent of this binary number is 24. The sum of all the weighted columns that contain a 1 is the decimal number that the PLC has stored.



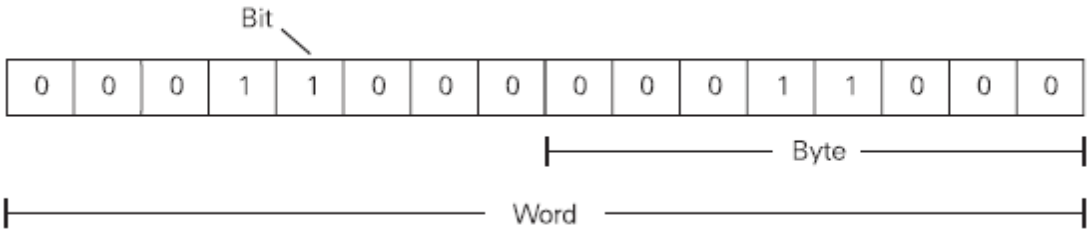
In the following example the fourth and sixth columns from the right contain a 1. The decimal value of the fourth column from the right is 8, and the decimal value of the sixth column from the right is 32. The decimal equivalent of this binary number is 40.



Bits, Bytes, and Words

Each binary piece of data is a bit. Eight bits make up one byte.

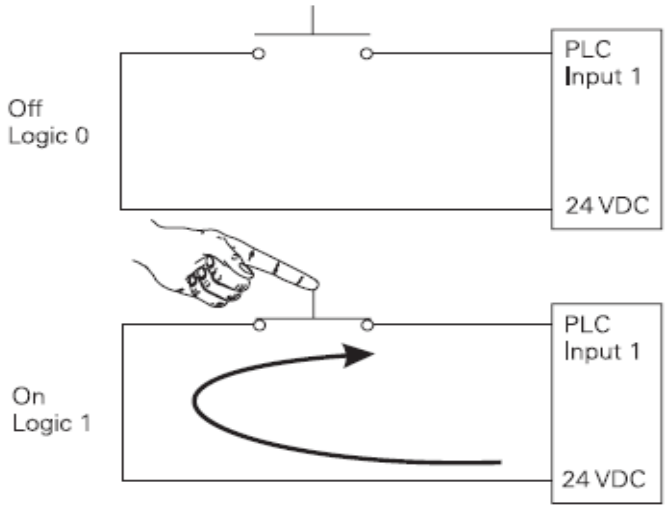
Two bytes, or 16 bits, make up one word.



Programmable controllers can only understand a signal that is On or Off (present or not present). The binary system is a system in which there are only two numbers, 1 and 0. Binary 1 indicates that a signal is present, or the switch is On. Binary 0 indicates that the signal is not present, or the switch is Off.

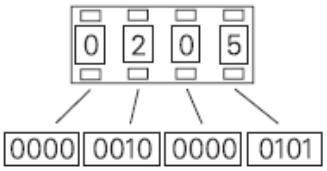
Logic 0, Logic 1

Programmable controllers can only understand a signal that is On or Off (present or not present). The binary system is a system in which there are only two numbers, 1 and 0. Binary 1 indicates that a signal is present, or the switch is On. Binary 0 indicates that the signal is not present, or the switch is Off.



BCD

Binary-Coded Decimal (BCD) numbers are decimal numbers where each digit is represented by a four-bit binary number. BCD is commonly used with input and output devices. A thumbwheel switch is one example of an input device that uses BCD. The binary numbers are broken into groups of four bits, each group representing a decimal equivalent. A four-digit thumbwheel switch, like the one shown here, would control 16 (4 x 4) PLC inputs.



Decimal Numbers	BCD Numbers
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Hexadecimal

Hexadecimal is another system used in PLCs. The hexadecimal system has the following characteristics:

16 digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Base 16

Weights Powers of base 16 (1, 16, 256, 4096 ...)

The ten digits of the decimal system are used for the first ten digits of the hexadecimal system. The first six letters of the alphabet are used for the remaining six digits.

A = 10	D = 13
B = 11	E = 14
C = 12	F = 15

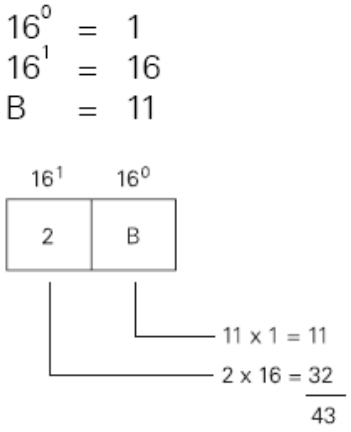
The hexadecimal system is used in PLCs because it allows the status of a large number of binary bits to be represented in a small space such as on a computer screen or programming device display. Each hexadecimal digit represents the exact status of four binary bits. To convert a decimal number to a hexadecimal number the decimal number is divided by the base of 16. To convert decimal 28, for example, to hexadecimal:

$$16\overline{)\,28}\quad 1 \text{ r } 12$$

Decimal 28 divided by 16 is 1 with a remainder of 12. Twelve is equivalent to C in hexadecimal. The hexadecimal equivalent of decimal 28 is 1C.

The decimal value of a hexadecimal number is obtained by multiplying the individual hexadecimal digits by the base 16 weight and then adding the results. In

the following example the hexadecimal number 2B is converted to its decimal equivalent of 43.



Conversion of Numbers

The following chart shows a few numeric values in decimal, binary, BCD, and hexadecimal representation.

Decimal	Binary	BCD	Hexadecimal
0	0	0000	0
1	1	0001	1
2	1	0010	2
3	11	0011	3
4	100	0100	4
5	101	0101	5
6	110	0110	6
7	111	0111	7
8	1000	1000	8
9	1001	1001	9
10	1010	0001 0000	A
11	1011	0001 0001	B
12	1100	0001 0010	C
13	1101	0001 0011	D
14	1110	0001 0100	E
15	1111	0001 0101	F
16	1 0000	0001 0110	10
17	1 0001	0001 0111	11
18	1 0010	0001 1000	12
19	1 0011	0001 1001	13
20	1 0100	0010 0000	14
.	.	.	.
.	.	.	.
126	111 1110	0001 0010 0110	7E
127	111 1111	0001 0010 0111	7F
128	1000 0000	0001 0010 1000	80
.	.	.	.
.	.	.	.
510	1 1111 1110	0101 0001 0000	1FE
511	1 1111 1111	0101 0001 0001	1FF
512	10 0000 0000	0101 0001 0010	200

BOOLEAN ALGEBRA

Boolean algebra was developed in the 1800's by James Bool, an Irish mathematician. It was found to be extremely useful for designing digital circuits, and it is still heavily used by electrical engineers and computer scientists. The techniques can model a logical system with a single equation. The equation can then be simplified and/or manipulated into new forms. The same techniques developed for circuit designers adapt very well to ladder logic programming.

Boolean equations consist of variables and operations and look very similar to normal algebraic equations. The three basic operators are AND, OR and NOT; more complex operators include exclusive or (EOR), not and (NAND), not or (NOR). Small truth tables for these functions are shown in Figure 6.1. Each operator is shown in a simple equation with the variables A and B being used to calculate a value for X. Truth tables are a simple (but bulky) method for showing all of the possible combinations that will turn an output on or off.

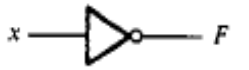
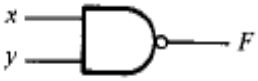
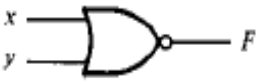
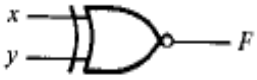
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = xy$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$
OR		$F = x + y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 1$
Inverter		$F = x'$	$x \mid F$ $0 \mid 1$ $1 \mid 0$
Buffer		$F = x$	$x \mid F$ $0 \mid 0$ $1 \mid 1$
NAND		$F = (xy)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
NOR		$F = (x + y)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 0$
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= x \odot y$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$

Figure 6.1 Boolean Operations with Truth Tables and Gates

Note: The symbols used in these equations, such as + for OR are not universal standards and some authors will use different notations.

Note: The EOR function is available in gate form, but it is more often converted to its equivalent, as shown below.

$$X = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$

In a Boolean equation the operators will be put in a more complex form as shown in Figure 6.2. The variable for these equations can only have a value of 0 for false, or 1 for true. The solution of the equation follows rules similar to normal algebra. Parts of the equation inside parenthesis are to be solved first. Operations are to be done in the sequence NOT, AND, OR. In the example the NOT function for C is done first, but the NOT over the first set of parentheses must wait until a single value is available. When there is a choice the AND operations are done before the OR operations. For the given set of variable values the result of the calculation is false.

given

$$X = \overline{(A + B \cdot C)} + A \cdot (B + \overline{C})$$

assuming A=1, B=0, C=1

$$X = \overline{(1 + 0 \cdot 1)} + 1 \cdot (0 + \overline{1})$$

$$X = \overline{(1 + 0)} + 1 \cdot (0 + 0)$$

$$X = \overline{(1)} + 1 \cdot (0)$$

$$X = 0 + 0$$

$$X = 0$$

Figure 6.2 A Boolean Equation

The equations can be manipulated using the basic axioms of Boolean shown in Figure 6.3. A few of the axioms (associative, distributive, commutative) behave like normal algebra, but the other axioms have subtle differences that must not be ignored.

Idempotent

$A + A = A$ $A \cdot A = A$

Associative

$(A + B) + C = A + (B + C)$ $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

Commutative

$A + B = B + A$ $A \cdot B = B \cdot A$

Distributive

$A + (B \cdot C) = (A + B) \cdot (A + C)$ $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$

Identity

$A + 0 = A$ $A + 1 = 1$

$A \cdot 0 = 0$ $A \cdot 1 = A$

Complement

$A + \overline{A} = 1$ $\overline{(\overline{A})} = A$

$A \cdot \overline{A} = 0$ $\overline{1} = 0$

DeMorgan's

$\overline{(A + B)} = \overline{A} \cdot \overline{B}$ $\overline{(A \cdot B)} = \overline{A} + \overline{B}$

Duality

interchange AND and OR operators, as well as all Universal, and Null sets. The resulting equation is equivalent to the original.

Figure 6.3 The Basic Axioms of Boolean Algebra

An example of equation manipulation is shown in Figure 6.4. The distributive axiom is applied to get equation (1). The idempotent axiom is used to get equation (2).

Equation (3) is obtained by using the distributive axiom to move C outside the parentheses, but the identity axiom is used to deal with the lone C. The identity axiom is then used to simplify the contents of the parentheses to get equation (4). Finally the Identity axiom is used to get the final, simplified equation. Notice that using Boolean algebra has shown that 3 of the variables are entirely unneeded.

$$A = \overline{\overline{B}} \cdot (C \cdot (\overline{D} + E + C) + \overline{\overline{F}} \cdot C)$$

$$A = \overline{\overline{B}} \cdot (\overline{D} \cdot C + E \cdot C + C \cdot C + \overline{\overline{F}} \cdot C) \quad (1)$$

$$A = \overline{\overline{B}} \cdot (\overline{D} \cdot C + E \cdot C + C + \overline{\overline{F}} \cdot C) \quad (2)$$

$$A = \overline{\overline{B}} \cdot C \cdot (\overline{D} + E + 1 + \overline{\overline{F}}) \quad (3)$$

$$A = \overline{\overline{B}} \cdot C \cdot (1) \quad (4)$$

$$A = \overline{\overline{B}} \cdot C \quad (5)$$

Note: When simplifying Boolean algebra, OR operators have a lower priority, so they should be manipulated first. NOT operators have the highest priority, so they should be simplified last. Consider the example from before.

$$X = \overline{(A + B \cdot C)} + A \cdot (B + \overline{C})$$

$$X = \overline{(A) + (B \cdot C)} + A \cdot (B + \overline{C})$$ ← The higher priority operators are put in parenthases

$$X = \overline{(A)} \cdot \overline{(B \cdot C)} + A \cdot (B + \overline{C})$$ ← DeMorgan's theorem is applied

$$X = \overline{A} \cdot (\overline{B} + \overline{C}) + A \cdot (B + \overline{C})$$ ← DeMorgan's theorem is applied again

$$X = \overline{A} \cdot \overline{B} + \overline{A} \cdot \overline{C} + A \cdot B + A \cdot \overline{C}$$ ← The equation is expanded

$$X = \overline{A} \cdot \overline{B} + (\overline{A} \cdot \overline{C} + A \cdot \overline{C}) + A \cdot B$$ ← Terms with common terms are collected, here it is only NOT C

$$X = \overline{A} \cdot \overline{B} + \overline{C} \cdot (\overline{A} + A) + A \cdot B$$ ← The redundant term is eliminated

$$X = \overline{A} \cdot \overline{B} + \overline{C} + A \cdot B$$ ← A Boolean axiom is applied to simplify the equation further

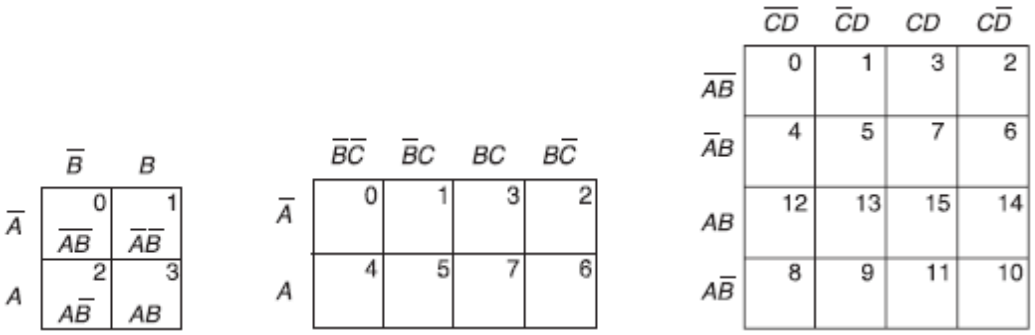
Combinational logic circuits

Logic circuits are classified into two categories: combinational and sequential. In a combinational logic circuit the output is a function of the present input only. It does not depend on the past values of the inputs. If the output is a function of past inputs (memory) as well as the present inputs, then the circuit is known as a sequential logic circuit.

The main objective of combinational circuit design is to construct a circuit utilizing the minimum number of gates and inputs from the behavioral specification of the circuit. The first step in the design process is to construct a truth table of the circuit from its specification. The sum-of-products or product-of-sums form of the Boolean expression is then derived from the truth table and simplified where possible. The simplified expression is then implemented into the actual circuit by using appropriate gates.

Karnaugh Maps

Boolean expressions can be graphically depicted and simplified with the use of Karnaugh maps. In a Karnaugh map 2^n possible minterms of an n-variable Boolean function are represented by means of separate squares or cells on the map. For example, the Karnaugh map of two variables A and B will consist of 2^2 squares, one for each possible combination of A and B as shown in Figure below. Each square of the Karnaugh map is designated by a decimal number written on the right-hand upper corner of the square. The decimal number corresponds to the minterm number of the Boolean function. The figure below shows Karnaugh map for a two, three and four -variable Boolean function.



One can derive the simplified logical function from the Karnaugh map as in the following example:

Let us consider the following four variable logical function

$$f(A,B,C,D) = \Sigma m(0,1,4,5,7,8,9,12,13,15)$$

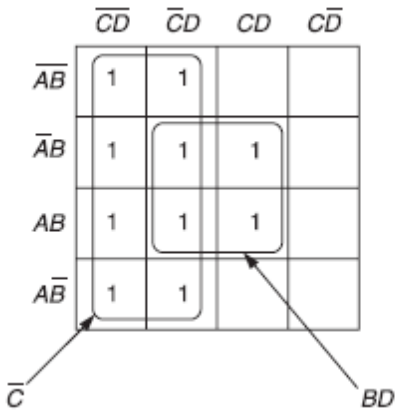
Then we need to simplify this function using K. Map

The Karnaugh map for the function is shown in Figure 3.12. The reduced form of the

function can be derived directly from the Karnaugh map:

$$f(A,B,C,D) = \bar{C} + BD$$

In four-variable Karnaugh maps, the top and bottom rows are logically adjacent and so are the left and right columns. We saw one example of grouping four cells that were not physically adjacent .



Don't Care Conditions

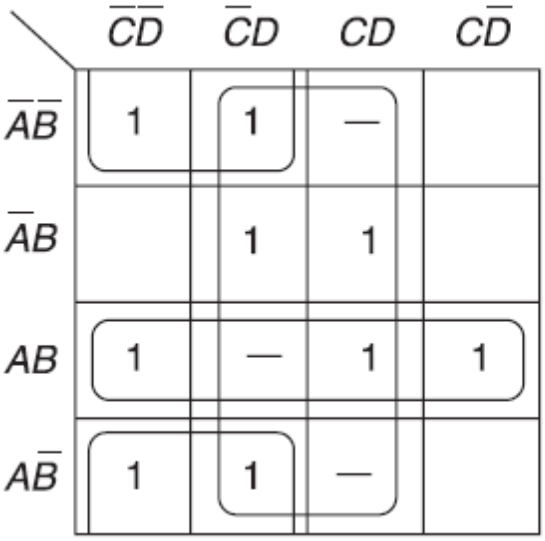
In certain Boolean functions it is not possible to specify the output for some input combinations. It means that these particular input combinations have no relevant effect on the output. These input combinations or conditions are called don't care conditions, and the minterms corresponding to these input combinations are called don't care terms. Functions that include don't care terms are said to be incompletely specified functions. The don't care minterms are labeled d instead of *m*.

Ex: Let us minimize the following Boolean function using a Karnaugh map:

$$f(A, B, C, D) = \Sigma m(0, 1, 5, 7, 8, 9, 12, 14, 15) + d(3, 11, 13)$$

The Karnaugh map is shown in Figure 3.15. From this the minimized function is given by

$$f(A, B, C, D) = D + AB + \bar{B}\bar{C}$$



Example: Design a combinational circuit with three inputs, x, y and z, and the three outputs, A, B, and C. when the binary input is 0, 1, 2, or 3, the binary output is one greater than the input. When the binary input is 4, 5, 6, or 7, the binary output is one less than the input.

Solution:

Design procedure:

1. Derive the truth table that defines the required relationship between inputs and outputs.

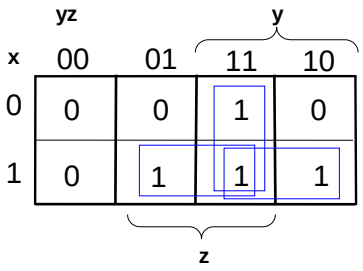
x	Y	z	A	B	C
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

2. Obtain the simplified Boolean functions for each output as a function of the input variables.

Map for output A:

The simplified expression from the map is:

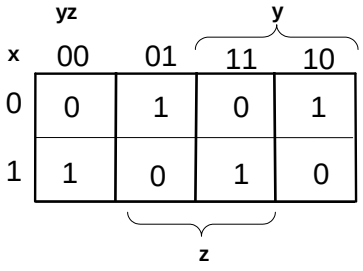
$$A = xz + xy + yz$$



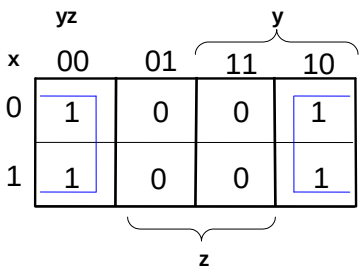
Map for output B:

The simplified expression from the map is:

$$B = x'y'z + x'yz' + xy'z' + xyz$$



Map for output C:



The simplified expression from the map is:

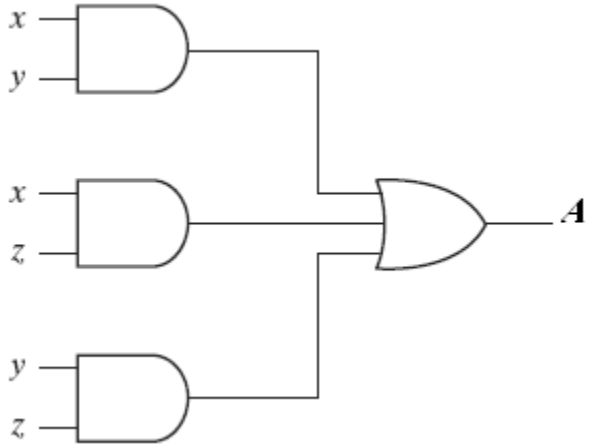
$C = z'$

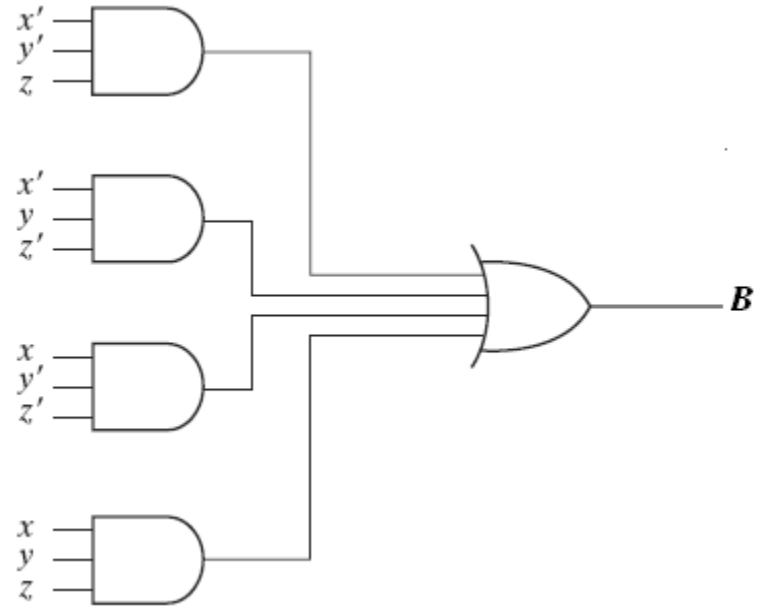
3. Draw the logic diagram.

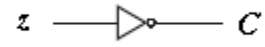
$A = xy + xz + yz$

$B = x'y'z + x'yz' + xy'z' + xyz$

$C = z'$


x
y
x
z
y
z
A


x′
y′
z
x′
y
z′
x
y′
z′
x
y
z
B


z
C

Lecture 3

Basic PLC Operation

PLC structure and its operation

The PLC mainly consists of a CPU, memory areas, and appropriate circuits to receive input and output data. We can consider the PLC to be a box full of hundreds or thousands of separate relays, counters, timers and data storage locations. These counters, timers, etc. don't "physically" exist but instead are simulated and can be considered software counters, timers, etc. These internal relays are simulated through bit locations in registers.

A programmable controller, as illustrated in Figure 1-5, consists of two basic sections:

• The central processing unit

• The input/output interface system

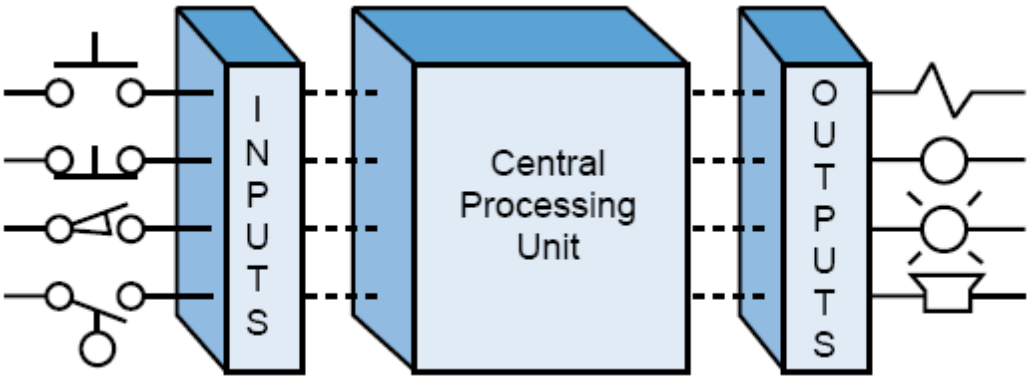


Figure 1-5. Programmable controller block diagram.

The central processing unit (CPU) governs all PLC activities. The following three components, shown in Figure 1-6, form the CPU:

• The processor
• The memory system
• The system power supply

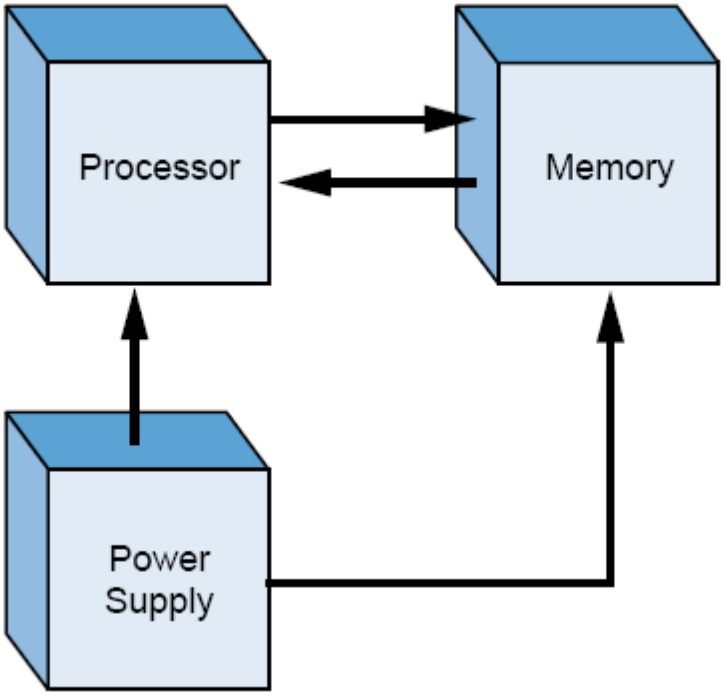


Figure 1-6. Block diagram of major CPU components.

The operation of a programmable controller is relatively simple. The input/ output (I/O) system is physically connected to the field devices that are encountered in the machine or that are used in the control of a process. These field devices may be discrete or analog input/output devices, such as limit switches, pressure transducers, push buttons, motor starters, solenoids, etc.

The I/O interfaces provide the connection between the CPU and the information providers (inputs) and controllable devices (outputs).

During its operation, the CPU completes three processes: (1) it reads, or accepts, the input data from the field devices via the input interfaces, (2) it executes, or performs, the control program stored in the memory system, and (3) it writes, or updates, the output devices via the output interfaces. This process of sequentially reading the inputs, executing the program in memory, and updating the outputs is known as scanning. Figure 1-7 illustrates a graphic representation of a scan.

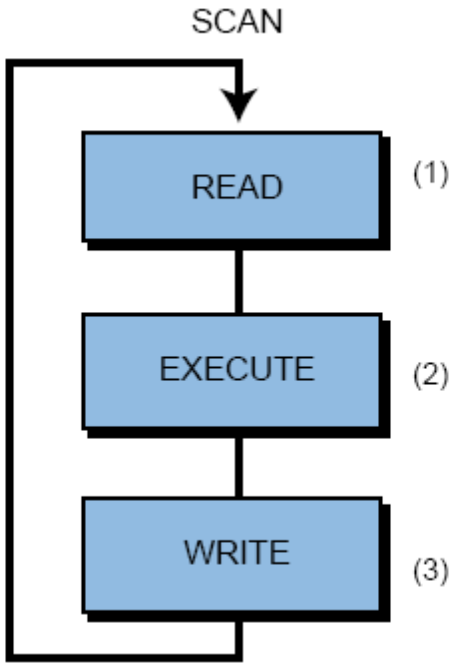


Figure 1-7. Illustration of a scan.

The input/output system forms the interface by which field devices are connected to the controller (see Figure 1-8). The main purpose of the interface is to condition the various signals received from or sent to external field devices. Incoming signals from sensors (e.g., push buttons, limit switches, analog sensors, selector switches, and thumbwheel switches) are wired to terminals on the input interfaces. Devices that will be controlled, like motor starters, solenoid valves, pilot lights, and position valves, are connected to the terminals of the output interfaces. The system power supply provides all the voltages required for the proper operation of the various central processing unit sections.

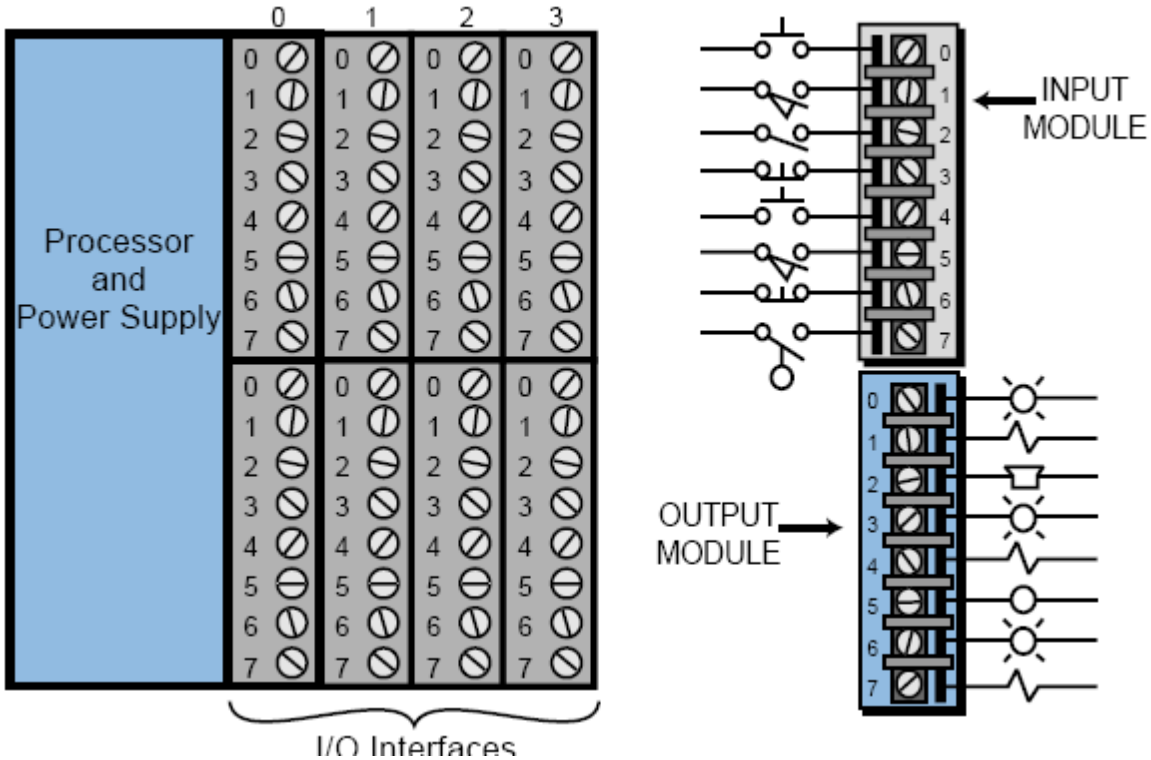
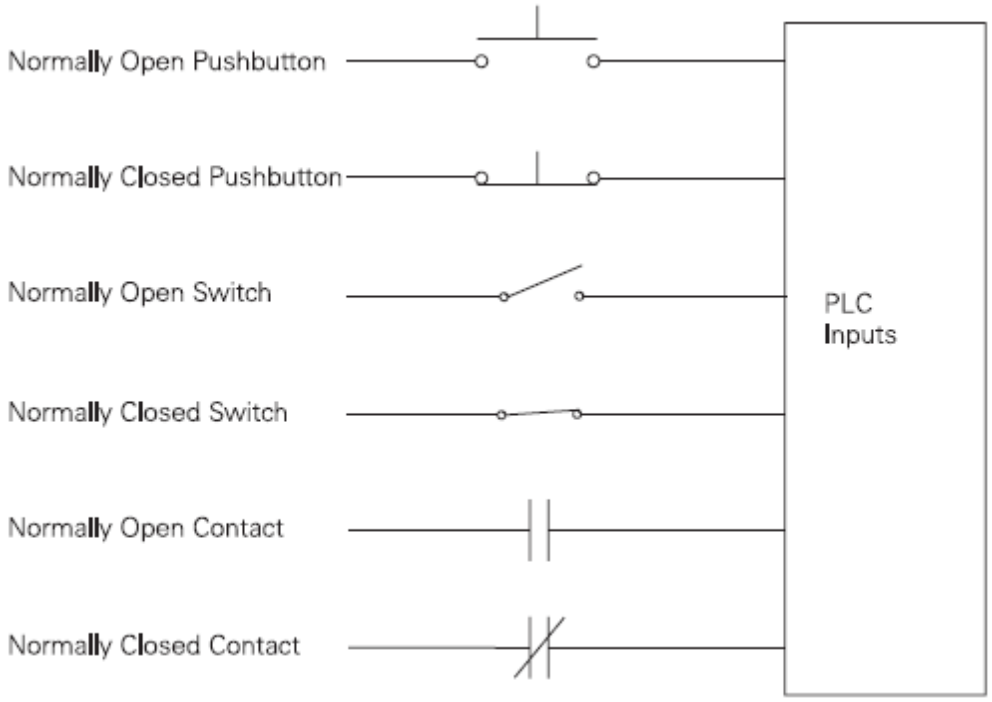


Figure 1-8. Input/output interface.

Although not generally considered a part of the controller, the programming device, usually a personal computer or a manufacturer's miniprogrammer unit, is required to enter the control program into memory.

Discrete Input

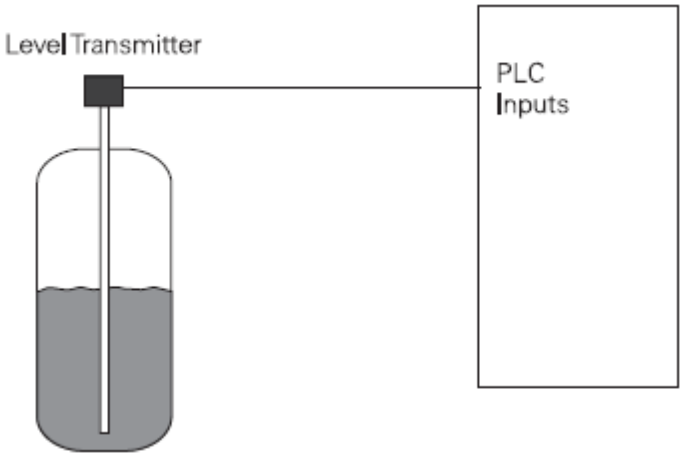
A discrete input also referred to as a digital input, is an input that is either in an ON or OFF condition. Pushbuttons, toggle switches, limit switches, proximity switches, and contact closures are examples of discrete sensors which are connected to the PLCs discrete or digital inputs. In the ON condition a discrete input may be referred to as a logic 1 or a logic high. In the OFF condition a discrete input may be referred to as a logic 0 or a logic low.



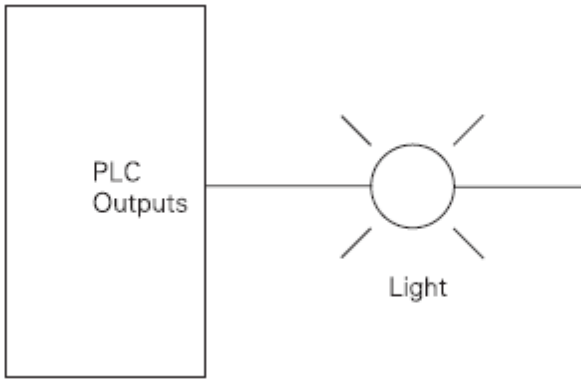
Analog Inputs

An analog input is an input signal that has a continuous signal. Typical analog inputs may vary from 0 to 20 milliamps, 4 to 20 milliamps, or 0 to 10 volts. In the

following example, a level transmitter monitors the level of liquid in a tank. Depending on the level transmitter, the signal to the PLC can either increase or decrease as the level increases or decreases.

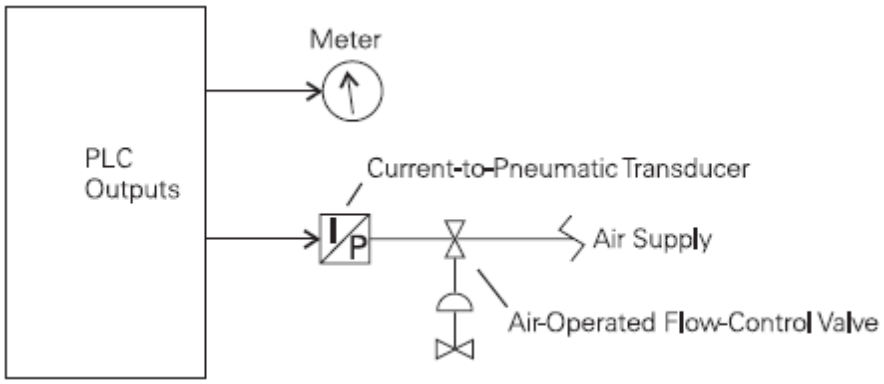


Discrete Outputs A discrete output is an output that is either in an ON or OFF condition. Solenoids, contactor coils, and lamps are examples of actuator devices connected to discrete outputs. Discrete outputs may also be referred to as digital outputs. In the following example, a lamp can be turned on or off by the PLC output it is connected to.



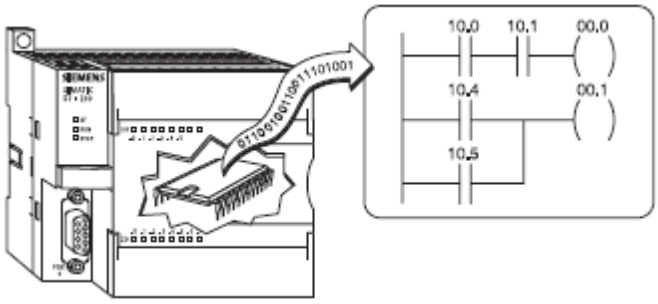
Analog Outputs

An analog output is an output signal that has a continuous signal. The output may be as simple as a 0-10 VDC level that drives an analog meter. Examples of analog meter outputs are speed, weight, and temperature. The output signal may also be used on more complex applications such as a current-topneumatic transducer that controls an air-operated flow-control valve.



CPU

The central processor unit (CPU) is a microprocessor system that contains the system memory and is the PLC decision-making unit. The CPU monitors the inputs and makes decisions based on instructions held in the program memory. The CPU performs relay, counting, timing, data comparison, and sequential operations.



Like other computerized devices, there is a CPU in a PLC, the CPU which is the brain of the PLC is able to do the following operation: -

- Updating inputs and outputs this function allows PLC to read the status of the input terminal and energize or de energize output terminals.
- Performing logic and arithmetic operation CPU conducts all the mathematical and logic operation involving in PLC.
- Communication with memory. PLC's program and data are stored in memory. When a PLC is operating, its CPU reads or changes the contents of memory location.
- Scanning application program which is called ladder diagram this scanning allow PLC to execute the application program as specified by the programmer.
- The CPU controls and supervises all operation within PLC, carrying out programmed instructions stored in the memory.
- An internal communications highway or bus system carries information to and from CPU, memory and I/O units, under CPU control.

- The CPU is supplied with a clock frequency by a quartz crystal or RC oscillator with speed depending on the microprocessor type.
- The clock determines the operating speed of the PLC and provides timing/synchronization.

The programming languages of PLC

The three types of programming languages used in PLCs are:

• Ladder

• Boolean

• Grafcet

The ladder and Boolean languages essentially implement operations in the same way, but they differ in the way their instructions are represented and how they are entered into the PLC. The Grafcet language implements control instructions in a different manner, based on steps and actions in a graphicoriented program.

LADDER LANGUAGE

The ladder diagram language is a symbolic instruction set that is used to create PLC programs. The ladder instruction symbols can be formatted to obtain the desired control logic, which is then entered into memory.

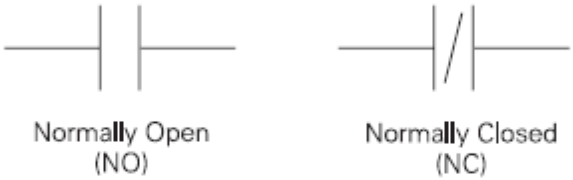
Symbols

In order to understand the instructions a PLC is to carry out, an understanding of the language is necessary. The language of

PLC ladder logic consists of a commonly used set of symbols that represent control components and instructions.

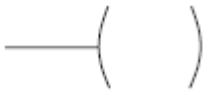
Contacts

One of the most confusing aspects of PLC programming for first-time users is the relationship between the device that controls a status bit and the programming function that uses a status bit. Two of the most common programming functions are the normally open (NO) contact and the normally closed
(NC) contact. Symbolically, power flows through these contacts when they are closed. The normally open contact (NO) is true (closed) when the input or output status bit controlling the contact is 1. The normally closed contact (NC) is true (closed) when the input or output status bit controlling the contact is 0.



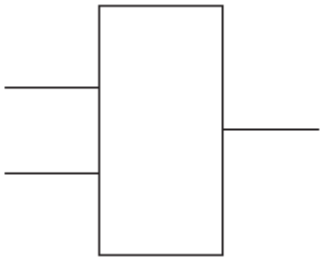
Coils

Coils represent relays that are energized when power flows to them. When a coil is energized, it causes a corresponding output to turn on by changing the state of the status bit controlling that output to 1. That same output status bit may be used to control normally open and normally closed contacts elsewhere in the program.



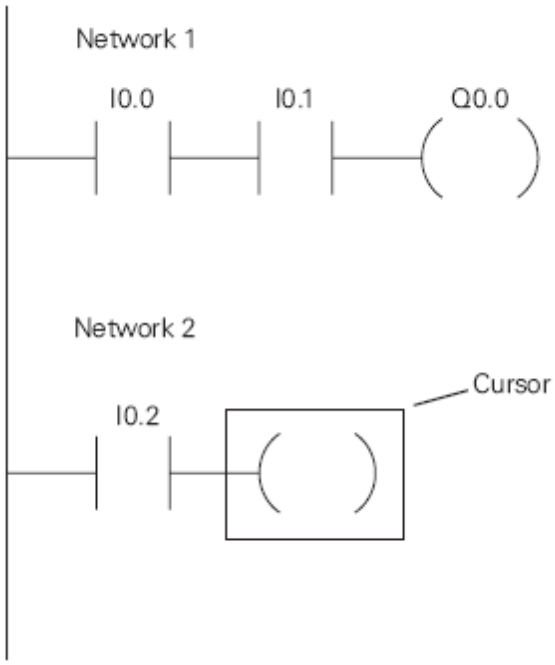
Boxes

Boxes represent various instructions or functions that are executed when power flows to the box. Typical box functions are timers, counters, and math operations.



Entering Elements

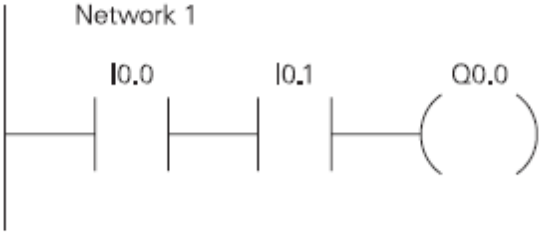
Control elements are entered in the ladder diagram by positioning the cursor and selecting the element from a lists. In the following example the cursor has been placed in the position to the right of I0.2. A coil was selected from a pulldown list and inserted in this position.



An AND Operation

Each rung or network on a ladder represents a logic operation. The following programming example demonstrates an AND operation. Two contact closures and one output coil are placed on network 1. They were assigned addresses I0.0, I0.1, and Q0.0. Note that in the statement list a new logic operation always begins with a load instruction (LD). In this example I0.0 (input 1) and (A in the statement list) I0.1 (input 2) must be true in order for output Q0.0 (output 1) to be true. It can also be seen That I0.0 and I0.1 must be true for Q0.0 to be true by looking at the function block diagram representation.

Ladder Diagram Representation



Statement List Representation

```
Network 1
LD      I0.0
A       I0.1
=       Q0.0
```

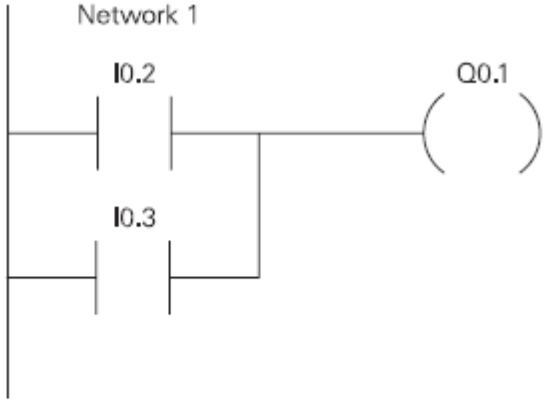
Function Block Diagram Representation

Network 1

An OR Operation

In this example an OR operation is used in network 1. It can be seen that if either input I0.2 (input 3) or (O in the statement list) input I0.3 (input 4), or both are true, then output Q0.1 (output 2) will be true.

Ladder Diagram Representation

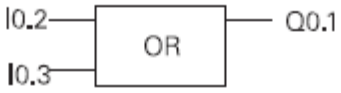


Statement List Representation

```
Network 1
LD      I0.2
O       I0.3
=       Q0.1
```

Function Block Diagram Representation

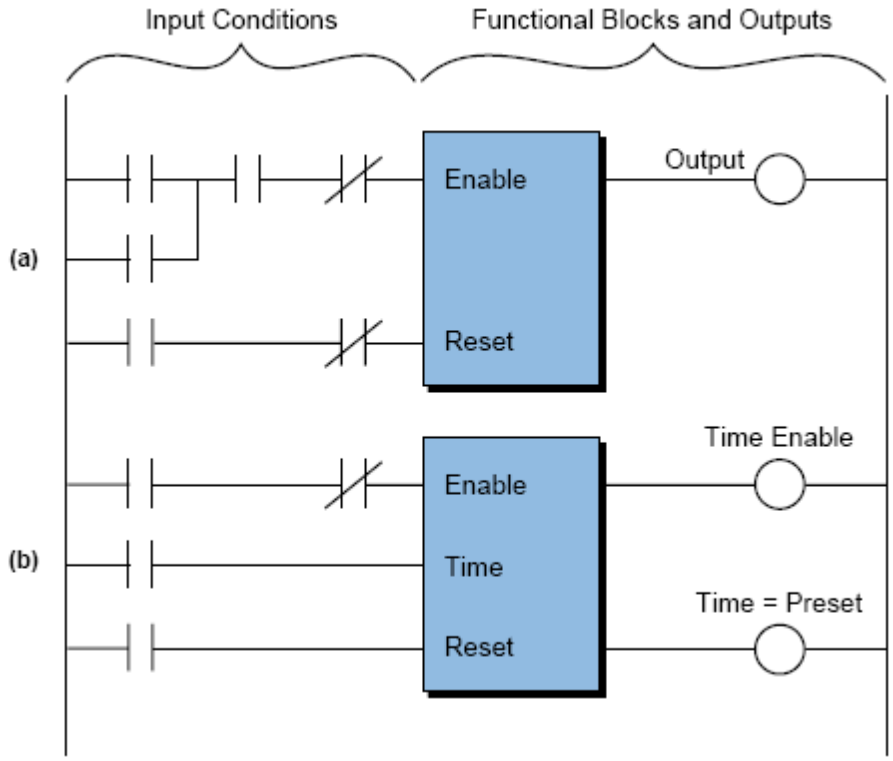
Network 1



Hint:

When a ladder diagram contains a functional block, contact instructions are used to represent the input conditions that drive (or *enable*) the block's logic. A functional block can have one or more enable inputs that control its operation. In addition, it can have one or more output coils, which signify the status of the function being performed. For example, the block shown in

Figure 9-9a has an enable block line, which when energized (i.e., continuity exists), will activate the block to perform the instruction. Thus, this instruction says: IF the enable is ON because the desired logic has continuity, THEN execute the block instruction. Depending on the instruction, other enable lines (see Figure 9-9b) may drive the block using reset or other control functions.



Instruction List (IL)

Series of instructions, each one must start on a new line.

One instruction = operator + one or more operations separated by commas.

Function Blocks lunched using a special operator.

Label	Operator	Operation	Comment
Run:	LD	%IX1	(*pushbutton*)
	ANDN	%MX5	
	ST	%QX2	(*run*)

The kinds of PLC Commands

Applicable Commands in LG PLC K Series :

- Sequence Command:

  Basic Commands for creating Sequence Logic Circuits.
- Comparison Command:

  Application Commands to execute the Comparison Operations.
- Arithmetic Command:

  Application Commands to execute the Arithmetic Operations.
- Logical Operation Command:

  Application Commands to execute the Logical Operations.
- Rotate/Shift Command:

  Application Commands to rotate or shift Data.
- Increment/Decrement:

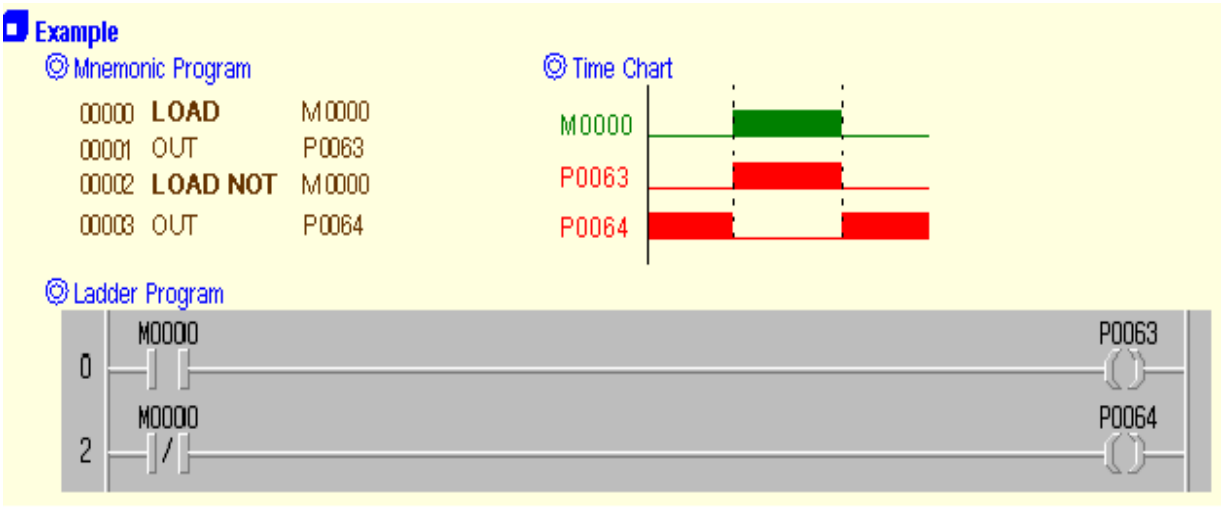
  Application Commands to add or subtract ‘ 1 ’ to the data.
- Conversion Command:

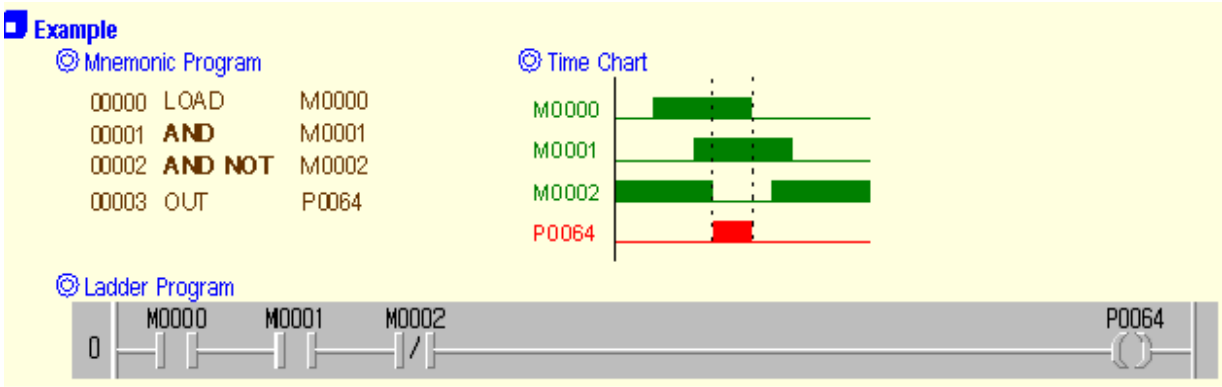
  Application Commands to change the type of Data.
- Transfer Command:

  Application Commands to copy, exchange or transfer Data between the internal devices.
- Timer/Counter Command:

  Basic Commands to use Timer and Counter.
- Jump/Interrupt Command:

Application Commands to execute Interrupt, Call or Jump with the specified program.



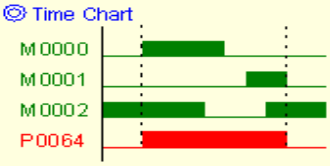


Example

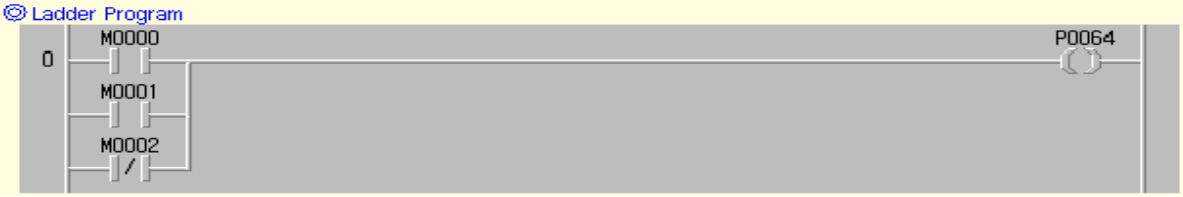
Mnemonic Program

```
00000  LOAD      M0000
00001  OR        M0001
00002  OR NOT    M0002
00003  OUT       P0064
```

Time Chart



Ladder Program



Example

Mnemonic Program

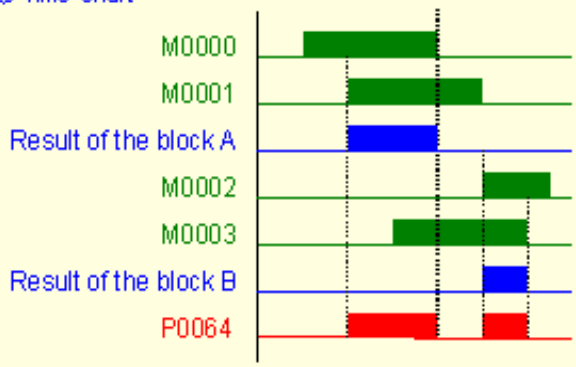
Time Chart

Example

Mnemonic Program

```
00000  LOAD      M 0000   Block A
00001  AND       M 0001   ;
00002  LOAD      M 0002   Block B
00003  AND       M 0003   ;
00004  OR LOAD
00005  OUT       P0064
```

Time Chart



Ladder Program

Lecture 4

PLC Simulation Software (LogixPro) with some applications

What exactly is LogixPro?

LogixPro is actually 3 major programs in one. First LogixPro contains a RSLogix look-alike editor which allows you to create and edit Ladder Logic programs using many of the same basic programming instructions utilized by Allen Bradley's RSLogix500. The look, feel and operation of the ladder rung editor so closely mimics Allen Bradley's that many will need a second look to be sure who's editor they're using.

Secondly, LogixPro contains a software PLC emulator which we simply call "The PLC". The PLC has much of the same functionality that an actual Allen Bradley PLC has. You may download your Ladder Logic programs to it just as you would with an actual PLC. Place The PLC into the "Run" mode, and it scans the I/O and executes your program just as you would expect of the real thing. You can't take a screwdriver to it, but you can't accidentally break it either, and you never have to replace it's batteries.

Thirdly, LogixPro contains the ProSimII simulations package. This is a collection of software simulations of real-world equipment that are graphically depicted on your computer's screen. Select the "Silo Simulator" with it's conveyor, proximity switches and solenoid controlled filling station, and you now have a real-world process that you can attempt to control. To do so, you of course must first write a proper Ladder Logic program, then download it to The PLC, and lastly place The PLC into the "Run" mode. If you did everything correctly, then the boxes will run along the conveyor being filled as they pass through the filling station. If you didn't

do everything correctly, then you may end up with a plant full of product, a scolding from "Merlin", and some serious troubleshooting ahead of you. Just like the real world, but a lot less costly!!...and you can always tell Merlin to just go away.

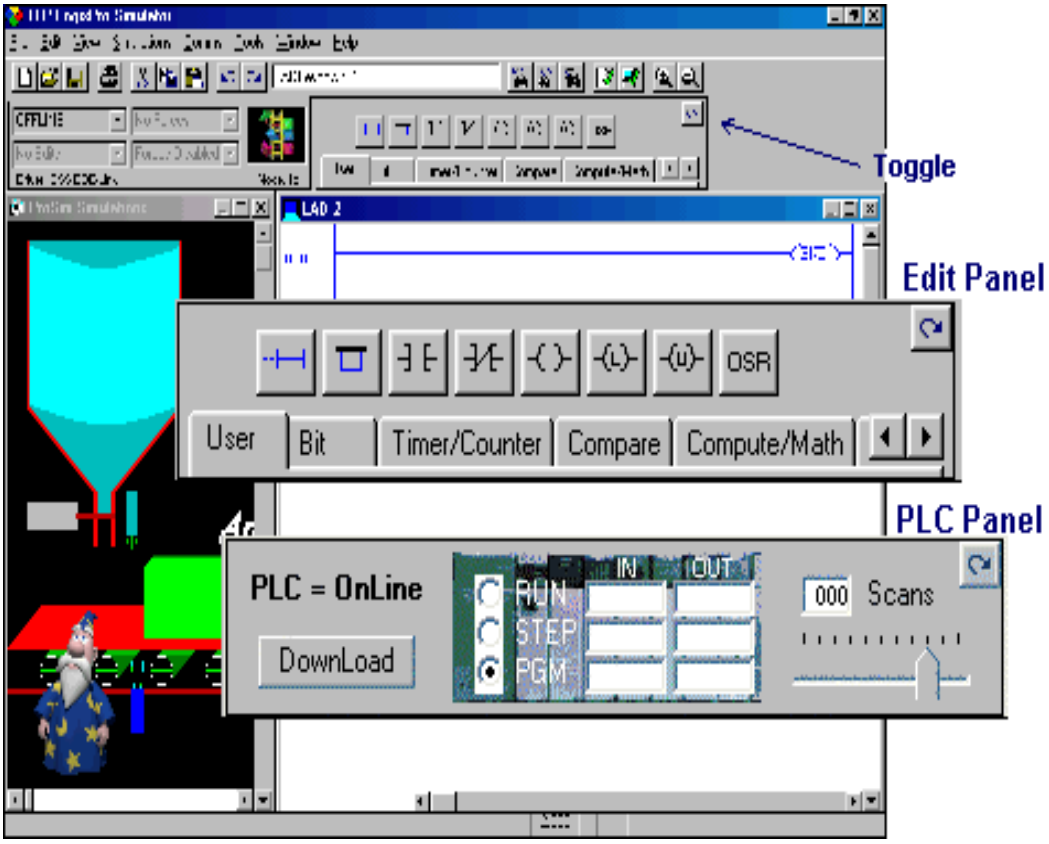
LogixPro is actually far superior to many custom designed PLC training stations that employ actual PLCs and utilize the full RSLogix software package. These dedicated stations commonly employ a handful of switches and lights to represent the imagined real-world process, and are incapable of responding in a synchronous fashion. If you turn on a light which supposedly represents a conveyor, nothing actually moves. You must then manually indicate that the conveyed object is in position by toggling a switch. With LogixPro however, your computer graphically simulates the full motion and operation of the process equipment just as would happen with real equipment. The synchronous and interactive nature of a real industrial processes is retained, and presents the student with a far more realistic and challenging programming experience. The fact that you can accomplish all this using just a computer, makes LogixPro ideal for PLC training wherever you go.

The LogixPro Screen

The most commonly used elements of LogixPro are displayed below. The **Edit Panel** provides easy access to all the RSLogix instructions and they may be simply dragged and dropped into your program.

Once your program is ready for testing, clicking on the "Toggle Button" of the Edit Panel will bring the **PLC Panel** into view. From the PLC Panel you can download

your program to the "PLC" and then place it into the "RUN" mode. This will initiate the scanning of your program and the I/O of your chosen simulat

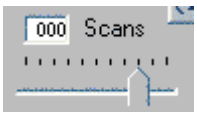


Editing Your Program

If your familiar with Windows and how to use a mouse, then you are going to find editing a breeze. Both Instructions and Rungs are selected simply by clicking on them with the left mouse button. Deleting is then just a matter of hitting the Del key on your keyboard.

Double Clicking with the left mouse button allows you to edit an instruction's address while right clicking (right mouse button) displays a pop-up menu of related editing commands.

Click on an Instruction or Rung with the left mouse button and keep it held down and you will be able to drag it wherever you please. Let go of it on any of the tiny locating boxes that you will see, and the Instruction or Rung will cling to it's new home. Isn't Windows Grand!

Debugging Your Program 

If you take a look at the PLC Panel you'll notice an adjustable Speed Control. This is not a component of normal PLCs, but is provided with LogixPro so that you may adjust the speed of the simulations to suit your particular computer.

When the simulation is slowed, so is the PLC scanning. You can use this to good effect when trying to debug your program. Set the scan slow enough and you can easily monitor how your program's instructions are responding. This capability may not be typical of real PLCs, but for Training Purposes, you will find that it is an invaluable debugging tool.

RSLogix Documentation

Be sure to check out the entries listed under "RSLogix/LogixPro Reference Documents & Links" on the lower half of the LogixPro Index page. Also, if you have the space on your hard drive, then seriously consider installing the "AB SLC® Instruction Set Reference Manual". An installation program has been provided which will integrate this Manual into the LogixPro Help menu and give you high speed access to the wealth of information it contains.

This is release v1.6.1 of the LogixPro Simulator.

About: LogixPro Simulator combine our prosim-ii programmable process simulations with a plc editor/emulator which mimics allen bradley's (rockwell) rslogix 500, and you have logixpro; a complete stand-alone plc training system without the expense of a plc. logixpro is the ideal tool for learning the fundamentals of rslogix ladder logic programming. the look, feel and operation of our ladder rung editor so closely mimics allen bradley's world renown software offering, that many need a second look to be sure who's editor they're using. of course the give-away is the window containing one of our prosim-ii simulations. this is where logixpro really out-shines typical plc training setups employing a plc connected to a handful of switches and lights. by graphically simulating process equipment such as conveyors etc. in software, the synchronous and interactive nature of real industrial processes, presents the student with a far more realistic and challenging programming experience.

logixpro allows you to practice and develop your rslogix programming skills where and when you want. it replaces the plc, ladder rung editor, and all the electrical components that have until now, been required to learn rslogix. it doesn't however, replace instructors, texts, tutorials or plc documentation manuals etc. which are so essential when learning about plcs and rslogix.

See DT.nfo for install information. This is a complete working version of LogixPro. While operating in the Trial or Evaluation mode, the File Save and Ladder Print functions are disabled. In addition, only the I/O, Door, and Silo simulations have been made fully available for user programming. These restrictions are removed once LogixPro is registered. To familiarize yourself with LogixPro and RSLogix programming, please go to the page titled "LogixPro ....

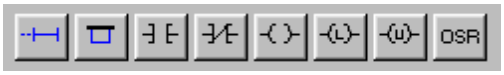
Student Exercises and Documentation" which is listed on the homepage of TheLearningPit.com. Once there, select the page "Getting Started with RSLogix and LogixPro ..... Please Read..!"

For those unfamiliar with RSLogix, I've included a program file for the

Silo Simulation to get you started. When you have LogixPro running select the Silo Simulation, then click Load in the File Menu and select the file "silo.rsl". Once it is loaded and you can see it, you can then "DownLoad" it to the PLC. At this point you can place the PLC into the "RUN" mode. If all goes well, just clicking on the simulations START pushbutton should get the whole process going. We don't support online editing, so remember to place the PLC in the "PGM" mode to edit and then "Download" to the PLC before attempting to "RUN" again. If you need help with the RSLogix addressing or instructions, go to the "LogixPro .... Student Exercises and Documentation" page entry which is listed on TheLearningPit.com Home Page. Also remember to try clicking on rungs, instructions etc with the right mouse button, to locate popup editing menus etc. We will continue to add additional instructions, commands etc over the coming months, but we must rely heavily on your feedback to track down installation problems, errors and unsuitable functionality.

RSLogix Relay Logic Instructions

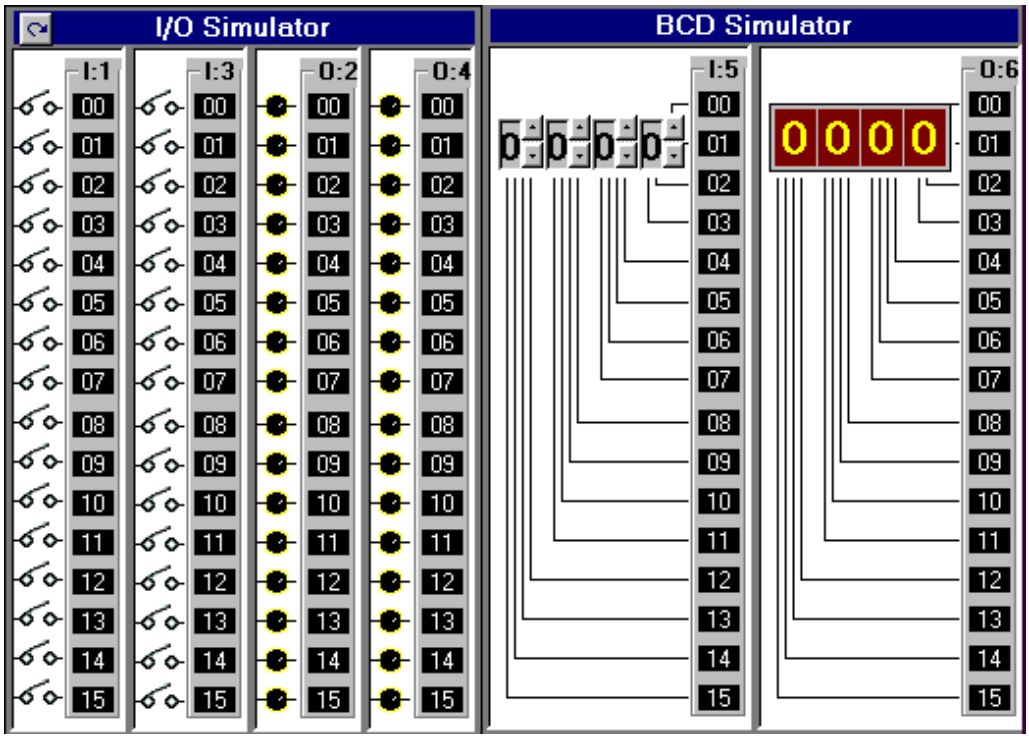
This exercise is designed to familiarize you with the operation of LogixPro and to step you through the process of creating, editing and testing simple PLC programs utilizing the Relay Logic Instructions supported by RSLogix.



From the Simulations Menu at the top of the screen, Select the I/O Simulation and ensure that the User Instruction Bar shown above is visible

The program editing window should contain a single rung. This is the End of Program rung and is always the last rung in any program. If this is the only rung visible then your program is currently empty.

If your program is not empty, then click on the File menu entry at the top of the screen and select "New" from the drop-down list. A dialog box will appear asking for you to select a Processor Type. Just click on "OK" to accept the default TLP LogixPro selection



The I/O Simulator

The simulator screen shown above, should now be in view. For this exercise we will be using the I/O simulator section, which consists of 32 switches and lights. Two groups of 16 toggle switches are shown connected to 2 Input cards of our simulated PLC. Likewise two groups of 16 Lights are connected to two output cards of our PLC. The two input cards are addressed as "I:1" and "I:3" while the output cards are addressed "O:2" and "O:4". Use your mouse to click on the various switches and note the change in the status color of the terminal that the switch is connected to. Move your mouse slowly over a switch, and the mouse cursor should change to a hand symbol, indicating that the state of switch can be altered by clicking at this location. When you pass the mouse over a switch, a "tool-tip" text box also appears and informs you to "Right Click to Toggle Switch Type". Click your right mouse button on a switch, and note how the switch type may be readily changed.

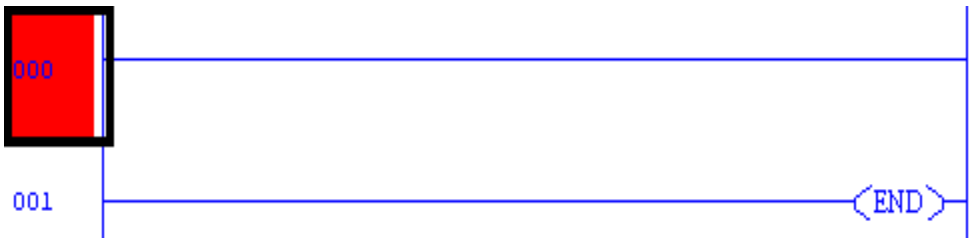
RSLogix Program Creation

Collapse the I/O simulation screen back to its normal size by clicking on the same (center) button you used to maximize the simulation's window. You should now be able to see both the simulation and program windows again. If you wish, you can adjust the relative size of these windows by dragging the bar that divides them with your mouse.

I want you to now enter the following single run program, which consists of a single Input instruction (XIC - Examine If Closed) and a single Output instruction

(OTE - Output Energize). There's more than one-way to accomplish this task, but for now I will outline what I consider to be the most commonly used approach

 First click on the "New Rung" button in the User Instruction Bar. It's the first button on the very left end of the Bar. If you hold the mouse pointer over any of these buttons for a second or two, you should see a short "ToolTip" which describes the function or name of the instruction that the button represents.

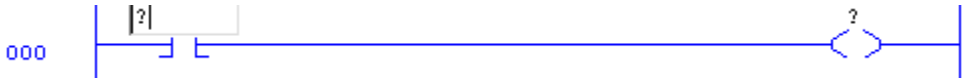


You should now see a new Rung added to your program as shown above, and the Rung number at the left side of the new rung should be highlighted. Note that the new Rung was inserted above the existing (END) End Of Program Rung. Alternatively you could have dragged (left mouse button held down) the Rung button into the program window and dropped it onto one of the locating boxes that would have appeared.

 Now click on the XIC instruction with your left mouse button (Left Click) and it will be added to the right of your highlighted selection. Note that the new XIC instruction is now selected (highlighted). Once again, you could have alternatively dragged and dropped the instruction into the program window. Note:

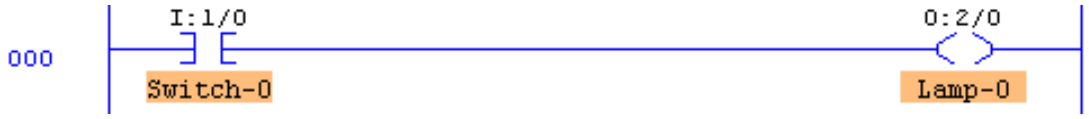
If you accidentally add an instruction, which you wish to remove, just Left Click on the instruction to select it, and then press the "Del" key on your keyboard. Alternatively, you may right click on the instruction and then select "Cut" from the drop-down menu that appears.

 Left Click on the OTE output instruction and it will be. Added to the right of your current selection.



Double Click (2 quick left mouse button clicks) on the XIC instruction's Question Mark and a textbox should appear which will allow you to enter the address (I:1/0) of the switch we wish to monitor. Begin typing address the question mark will be writing over or use the Backspace key to get rid of the "?" currently in the textbox. Once you type in the address, click anywhere else on the instruction (other than the textbox) and the box should close

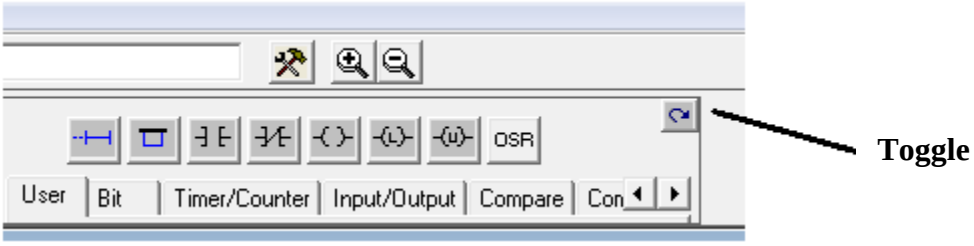
Enter the address the OTE instruction, before continuing however, Double check that the addresses of your instructions are correct.



Right Click on the XIC instruction and select "Edit Symbol" from the drop-down menu that appears. Another textbox will appear where you can type in a name (Switch-0) to associate with this address. As before, a click anywhere else will close the box. Add symbol Lamp-0 to the OTE address O:2/0.

Testing your Program

It's now time to "Download" your program to the PLC. First click on the "Toggle" button at the top right corner of the Edit Panel, which will bring the PLC Panel into view.



The PLC panel will viewed as

Click on the "DownLoad" button to initiate the downloading of your program to the PLC. Once complete, click inside the "RUN" option selection circle to start the PLC scanning.

Enlarge the Simulation window so that you can see both the Switches and Lamps, by dragging the bar that separates the Simulation and Program windows to the right with your mouse. Now click on Switch I:1/00 in the simulator and if all is well, Lamp O:2/00 should illuminate.

Toggle the Switch On and Off a number of times and note the change in value indicated in the PLC Panel's status boxes, which are being updated constantly as the PLC Scans. Try placing the PLC back into the "PGM" mode and then toggle the simulator's Switch a few times and note the result. Place the PLC back into the "Run" mode and the Scan should resume.

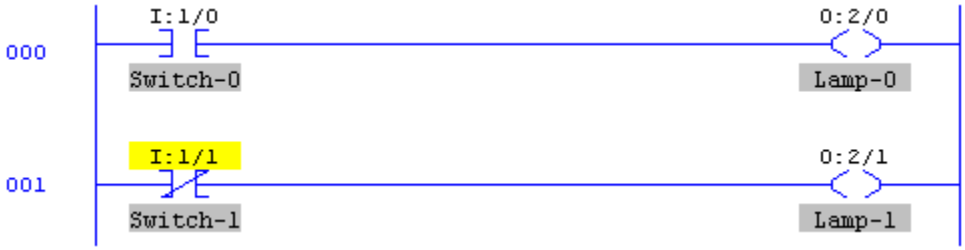
We are usually told to think of the XIC instruction as an electrical contact that allows electrical flow to pass when an external switch is closed. We are then told that the OTE will energize if the flow is allowed to get through to it. In actual fact the XIC is a conditional instruction that tests any bit that we address for Truth or logic (1).

Editing your Program

Click on the "Toggle" button of the PLC Panel, which will put the PLC into the PGM mode and bring the Edit Panel back into view.

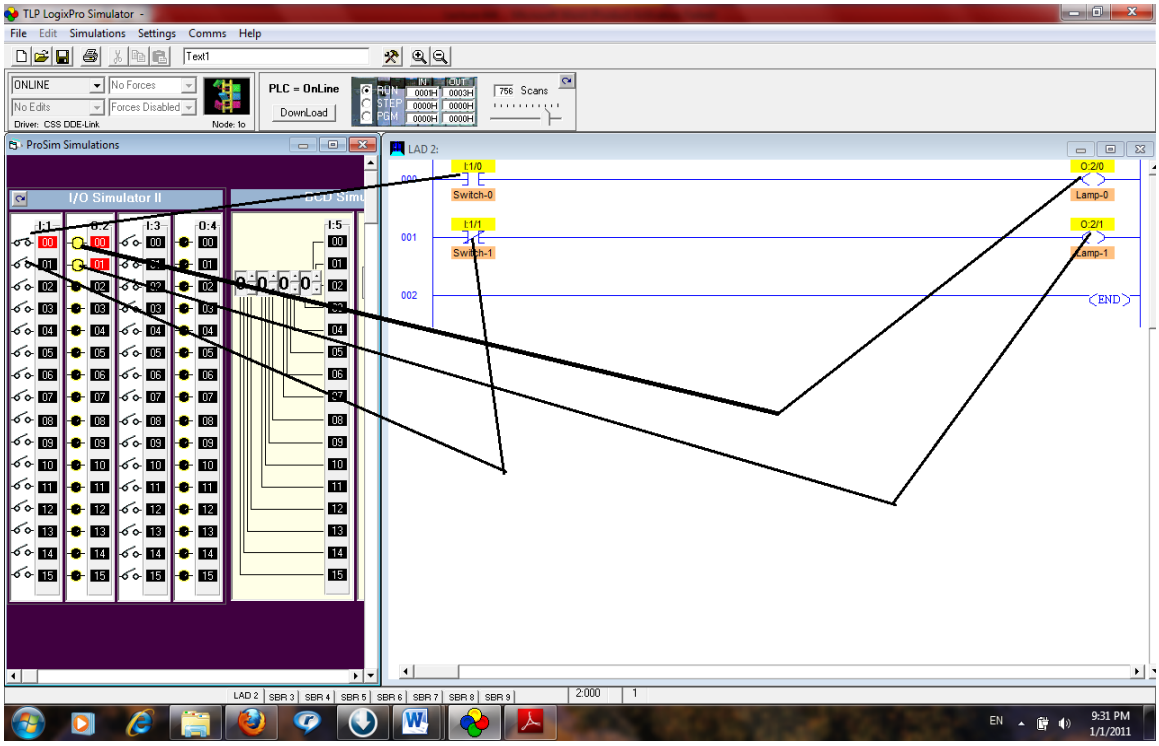
Now add a second rung to your program as shown below. This time instead of entering the addresses as you did before, try dragging the appropriate address which is displayed in the I/O simulation and dropping it onto the instruction.

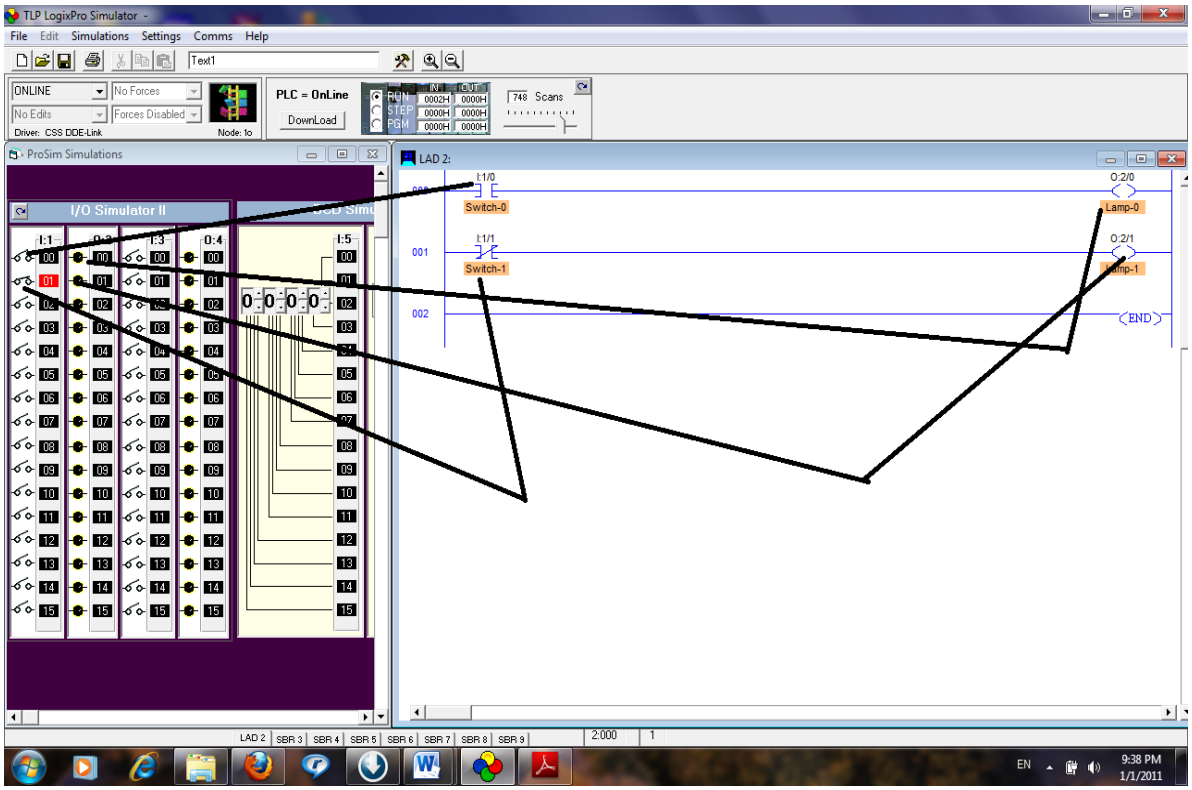
Note that the XIC instruction, which Tests for Zero or False has, it's address highlighted in yellow. This indicates that the instruction is True, which in the case of an XIC, means that the bit addressed is currently a Zero or False



This is probably a good time to practice your drag drop skills. Try moving instructions from rung to rung by holding the left mouse button down while over an instruction, and then while keeping the mouse button down, move the mouse (and instruction) to a new location. Try doing the same with complete rungs by dragging the box at the left end of the rung and dropping it in a new location.

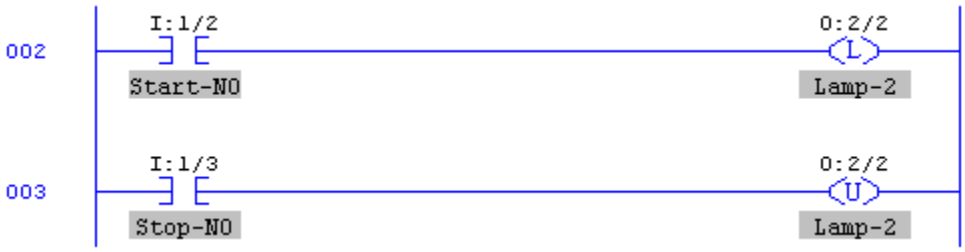
Once you feel comfortable with drag drop, ensure that your program once again looks like the one pictured above, Now download your program to the PLC and place the PLC into the Run Mode. Toggle both Switch-0 and Switch-1 on and off a number of times and observe the effects this has on the lamps. Ensure that you are satisfied with the operation of your program before proceeding further.


TLP LogixPro Simulator
File Edit Simulations Settings Comms Help
PLC = OnLine
DownLoad
ProSim Simulations
I/O Simulator II
LAD 2:
I:1/0
Switch-0
I:1/1
Switch-1
O:2/0
Lamp-0
O:2/1
Lamp-1
END
9:31 PM
1/1/2011


TLP LogixPro Simulator
File Edit Simulations Settings Comms Help
PLC = OnLine
DownLoad
ProSim Simulations
I/O Simulator II
LAD 2:
I:1/0
Switch-0
I:1/1
Switch-1
O:2/0
Lamp-0
O:2/1
Lamp-1
END
9:38 PM
1/1/2011

Stop/Start utilizing OTL and OUT – Output Latch and Unlatch

For this exercise we need two Normally Open momentary switches. Using your right mouse button, click on switch "I:1/2" and "I:1/3", changing them to N.O. pushbuttons. Now add the following two rungs to your program. Once you have the rungs entered correctly, download and run your modified program.



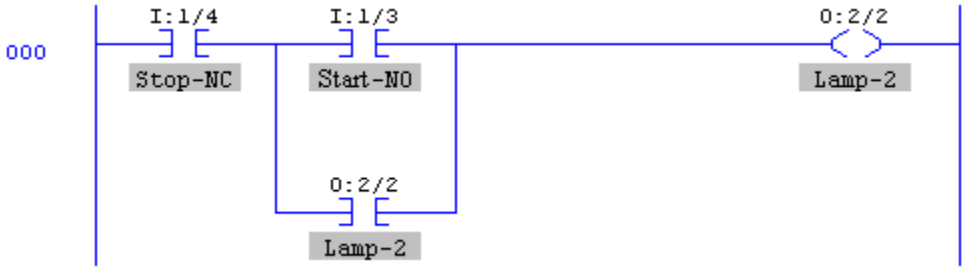
Activate the Start and Stop switches and ensure that the OTL and OTL output instructions are responding as outlined in your text. Once you have the lamp ON, could you turn it off if power was lost in the Stop Switch circuit?

Now modify your program so that it operates correctly when you substitute the N.O. Stop switch (I:1/03) with a Normally Closed Switch. If we now lost power on the N.C. Stop switch circuit, what would happen to the state of Lamp (O:2/02)?

Emulating Standard Stop/Start Control

Erase your program by selecting "New" from the "File" menu selection at the top of the screen. When the dialog box appears just click on "OK" to select the default

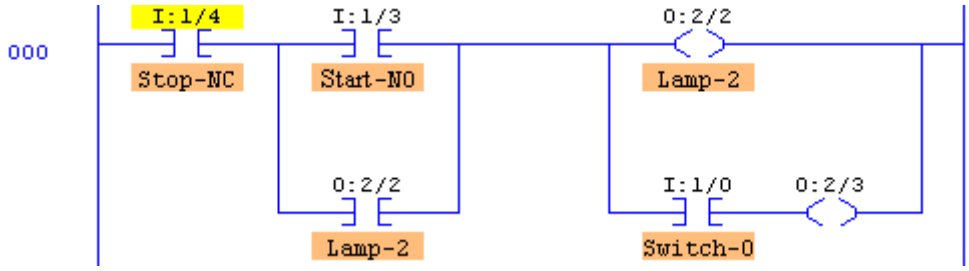
PLC type. Now enter the following program. To enter a branch, just drag the branch (button) onto the rung and then insert or drag instructions into the branch.



Before you download and run this program, take a careful look at our use of a XIO instruction to test the state of the N.C. Stop Switch. When someone presses the Stop Switch, will bit I:1/04 go True or False? Will the XIC instruction go True or False when the Switch is pressed? Is this the logic we are seeking in this case? .... Run the program and see if you're right! .... If we loose power in the Stop Switch circuit, what state will the lamp go to? .... Why do you think that most prefer this method rather than the OTL/OTU method of implementing Stop/Start Control?

Output Branching with RSLogix

Modify your program so that it matches the following.



Download and Run the program. Operate the Stop and Start switches several times with Switch-0 open, and again with Switch-0 closed. Remove the XIC instruction from the Output branch and note what happens to Lamp-3 when you Start and Stop

the circuit. Try moving the Lamp-3 OTE instruction so that it is in series with the Lamp-2 OTE instruction. Download, Run and observe how both lamps still light even with the empty branch (short?) in place. It may look like an electrical circuit but in fact we know that it isn't and therefore obeys a somewhat different set of rules. Remove the empty branch, Download, Run and see if this has any effect on the logic or operation of the rung.

Controlling One Light from two Locations

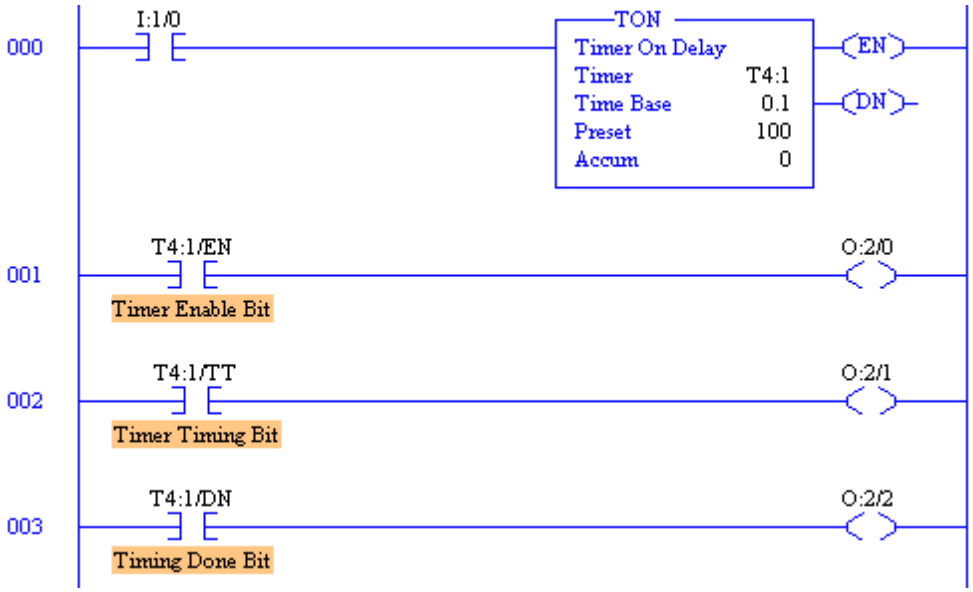
Create, enter and test a program, which will perform the common electrical function of controlling a light from two different locations. Clear your program and utilize toggle switch (I:1/00) and switch (I:1/01) to control Lamp (O:2/00)... (Hint: If both switches are On or if both switches are Off, then the Lamp should be On! This of course is just one approach to solving this problem)

Lecture 5

Introduction To RSLogix Timers in the LogixPro Simulator

The TON Timer.... (Timer ON Delay)

- From the LogixPro Simulations Menu, select the I/O Simulation.
- Clear out any existing program by selecting the "New" entry in the File menu, and then select the "Clear Data Table" entry in the Simulations menu.
- Now enter the following program being careful to enter the addresses exactly as shown.
- Confirm that you have entered the number 100 as the timer's preset value. This value represents a 10 second timing interval (10x0.1) as the timebase is fixed at 0.1 seconds:



- Once you have your program entered, and have ensured that it is correct, download it to the PLC.

- Ensure that Switch I:1/0 is Open, and then place the PLC into the Run mode.
- Right click on the Timer instruction, and select "GoTo DataTable" from the drop-down menu.
- Note the initial value of timer T4:1's accumulator and preset in the spaces below. Also indicate the state of each of the timer's control bits in the spaces provided: Initial State (Switch I:1/0=Open):

  T4:1.ACC = _____ T4:1.PRE = _____ T4:1/EN = ____ T4:1/TT = ____ T4:1/DN = ____
- Close switch I:1/0, and carefully observe the incrementing of the timer's accumulator, and the state of each of it's control bits.
- Once the Timer stops incrementing, note the final value of timer T4:1's accumulator, preset, and the state of it's control bits below:

  Final State (Switch I:1/0=Closed):

  T4:1.ACC = _____ T4:1.PRE = _____ T4:1/EN = ____ T4:1/TT = ____ T4:1/DN = ____

- Toggle the state of switch I:1/0 a number of times, and observe the operation of the Timer in both the DataTable display and in the Ladder Rung program display.
- Confirm that when the rung is taken false, the accumulator and all 3 control bits are reset to zero. This type of timer is a non-retentive instruction, in that the truth of the rung can cause the accumulator and control bits to be reset (=0).

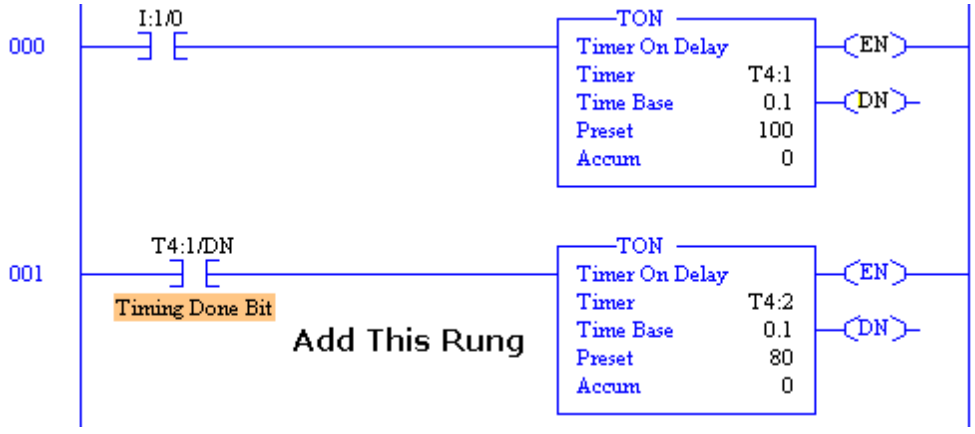
Conclusions:

Use the TON instruction to turn an output on or off after the timer has been on for a preset time interval. This output instruction begins timing when its rung goes "true". It waits the specified amount of time (as set in the PREset), keeps track of the accumulated intervals which have occurred (ACCumulator), and sets the DN (done) bit when the ACC (accumulated) time equals the PRESET time.

As long as rung conditions remain true, the timer adjusts its accumulated value (ACC) each evaluation until it reaches the preset value (PRE). The accumulated value is reset when rung conditions go false, regardless of whether the timer has timed out.

Cascaded TON Timers

- Insert a new rung containing a second timer just below the first rung as shown below. This second timer T4:2 will be enabled when the first timer's Done bit T4:1/DN goes true or high (1).



- Once you have completed this addition to your program, download your program to the PLC and select RUN.

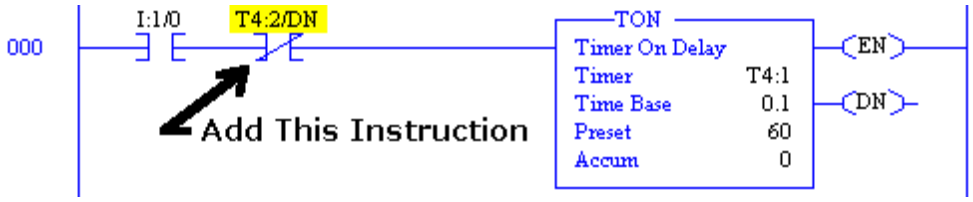
- Toggle the state of switch I:1/0 to ON and observe the operation of the timers in your program.
- Bring the DataTable display into view, and pay particular attention to the way in which the timers are cascaded (one timer starts the next).
- Try changing the value of one of the timer presets by double clicking on the preset value in the DataTable display, and then entering a new value.
- Run the timers through their timing sequence a number of times. Don't move on until you are satisfied that the timers are working as you would expect.

In this exercise we have utilized just two timers, but there is nothing stopping us from sequencing as many timers as we wish. The only thing to remember is; to use the DN (done) bit of the previous timer to enable the next timer in the sequence. Obviously locating the timers on consecutive rungs, and employing consecutive numbering will make such a program much easier to read and trouble-shoot.

Self Resetting Timers

- Place the PLC into the PGM mode, and modify the first rung of your program as depicted below.
- Once you have modified your program, download it to the PLC and place the PLC into the RUN mode.
- Close switch I:1/0 and observe the operation of the timers. The timers should now be operating in a continuous loop with Timer1 starting Timer2, and then when Timer2 is done, Timer1 is reset by Timer2's done bit. As before,

when Timer1 is reset, it in turn resets Timer2 which causes Timer2's done to go low (T4:2/DN=0).



Once Timer2's done bit is low, the sequence is back to where it originally began, and the timing sequence will start over once again on the very next scan.

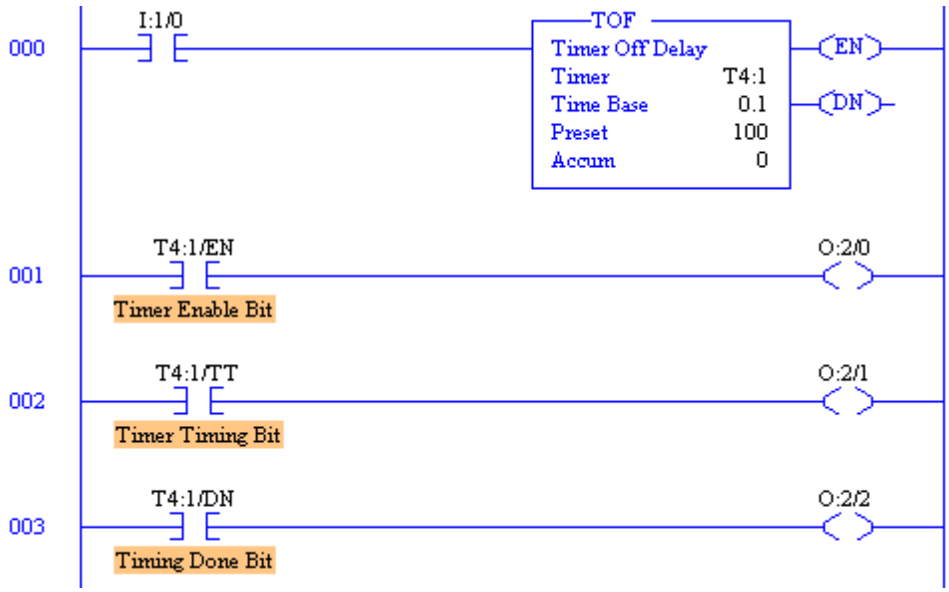
- Remove the first instruction (switch XIC I:1/0) from rung zero of your program.
- Download and RUN this modified version of your program
- Does the timing operation continuously sequence as before? It should!
- Can you stop the timing sequence? Not without taking the PLC out of the RUN mode! In many applications there may never be a need to stop such a timing sequence, so a switch might not be used or needed.

In this exercise we cascaded two timers, but as before there is nothing to stop us from cascading as many timers as we wish. The thing to remember here is; utilize the DN (XIC or "NOT"done) bit of the last timer in the sequence to reset the first timer in the sequence. Once again, consecutive rungs, and numbering will make a program much easier to read and trouble-shoot.

The TOF Timer.... (Timer OFF Delay)

In Allen Bradley PLC programming, the TON timer is by far the most commonly used type of timer. Most people consider TON timers to be simple to use and understand. In comparison, many people find the operation of the Allen Bradley TOF (Timer OFF delay) timer to be less intuitive, but I'm going to let you decide for yourself.

- Make sure that switch I:1/0 is Closed, and then enter or modify your existing program to match the one shown below.



- Once you have your program entered, and have ensured that it is correct, download it to the PLC.
- Ensure that Switch I:1/0 is Closed, and then place the PLC into the Run mode.
- Right click on the Timer instruction, and select "GoTo DataTable" from the dropdown menu.

- Note the initial value of timer T4:1's accumulator and preset in the spaces below.

Also indicate the state of each of the timer's control bits in the spaces provided:
Initial State (Switch I:1/0=Closed):
T4:1.ACC = _____ T4:1.PRE = _____ T4:1/EN = ____ T4:1/TT = ____ T4:1/DN = ____

- Open switch I:1/0, and carefully observe the incrementing of the timer's accumulator, and the state of each of it's control bits.
- Once the Timer stops incrementing, note the final value of timer T4:1's accumulator, preset, and the state of it's control bits below:

Final State (Switch I:1/0=Open):
T4:1.ACC = _____ T4:1.PRE = _____ T4:1/EN = ____ T4:1/TT = ____ T4:1/DN = ____

- Toggle the state of switch I:1/0 a number of times, and observe the operation of the Timer in both the DataTable display and in the Ladder Rung program display.
- Confirm that when the rung is taken true, the accumulator and all 3 control bits are reset to zero. The TOF timer like the TON timer is also a non-retentive instruction and can be reset by changing the truth of the rung.

Conclusions:

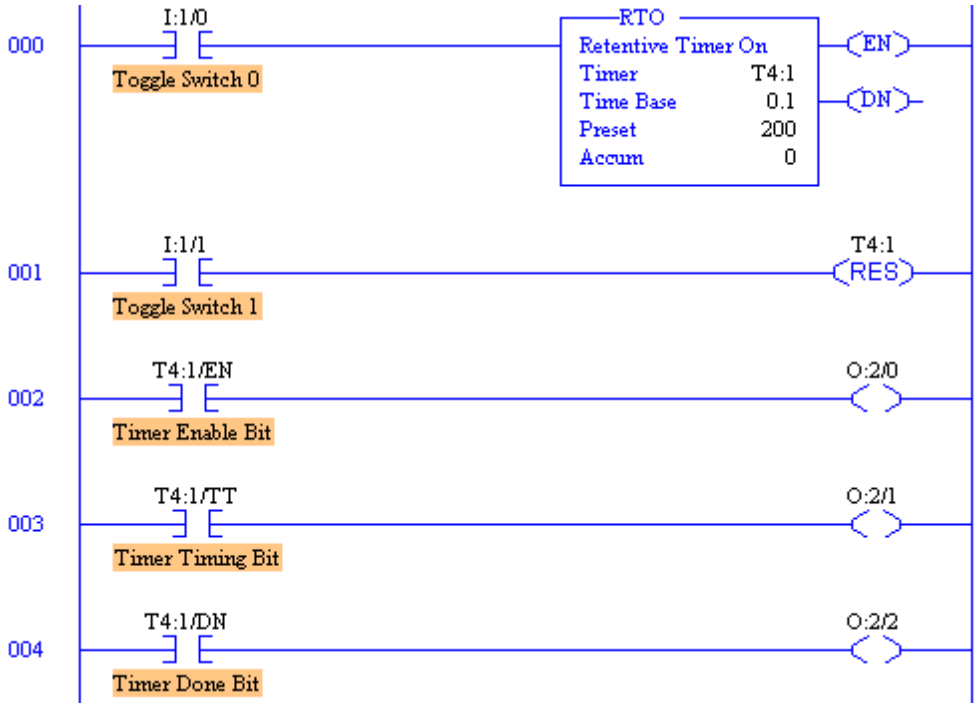
Use the TOF instruction to turn an output on or off after its rung has been off for a preset time interval. This output instruction begins timing when its rung goes "false." It waits the specified amount of time (as set in the PRESET), keeps track

of the accumulated intervals which have occurred (ACCUM), and resets the DN (done) bit when the ACCUM (accumulated) time equals the PRESET time.

The Accumulated value is reset when rung conditions go true regardless of whether the timer has timed out.

The RTO Timer.... (Retentive Timer ON)

- Make sure that switch I:1/0 is Open, and then replace the TOF timer in your program with a RTO retentive timer.
- Now insert a new rung below the timer, and add the XIC,I:1/1 and RES,T4:1 instructions.
- Your program should now match the one shown below:



- Once you have your program entered, and have ensured that it is correct, download it to the PLC.

- Ensure that both Switches are Open, and then place the PLC into the Run mode.
- Right click on the Timer instruction, and select "GoTo DataTable" from the dropdown menu.
- Note the initial value of timer T4:1's accumulator preset and control bits. Are we starting off with the same values we had in the TON exercise? You should be answering Yes!
- Close switch I:1/0 for 2 or 3 seconds and then Open it again.
- Note that the timer stopped timing when the rung went false, but the accumulator was not reset to zero.
- Close the switch again and leave it closed which will allow the timer to time-out (ACC=PRE).
- Once timed out, note the state of the control bits
- Open the switch, and once again note the state of the control bits.
- Now close Switch I:1/1 and leave it closed. This will cause the Reset instruction to go true.
- Close switch I:1/0 momentarily to see if the timer will start timing again. It should not!
- Open Switch I:1/1 which will cause the Reset instruction return to false.
- Now toggle switch I:1/0 several times and note that the timer should again start timing as expected.
- Repeat the foregoing steps, until you are satisfied that you clearly understand the operation of both the RTO timer, and the Reset instruction.

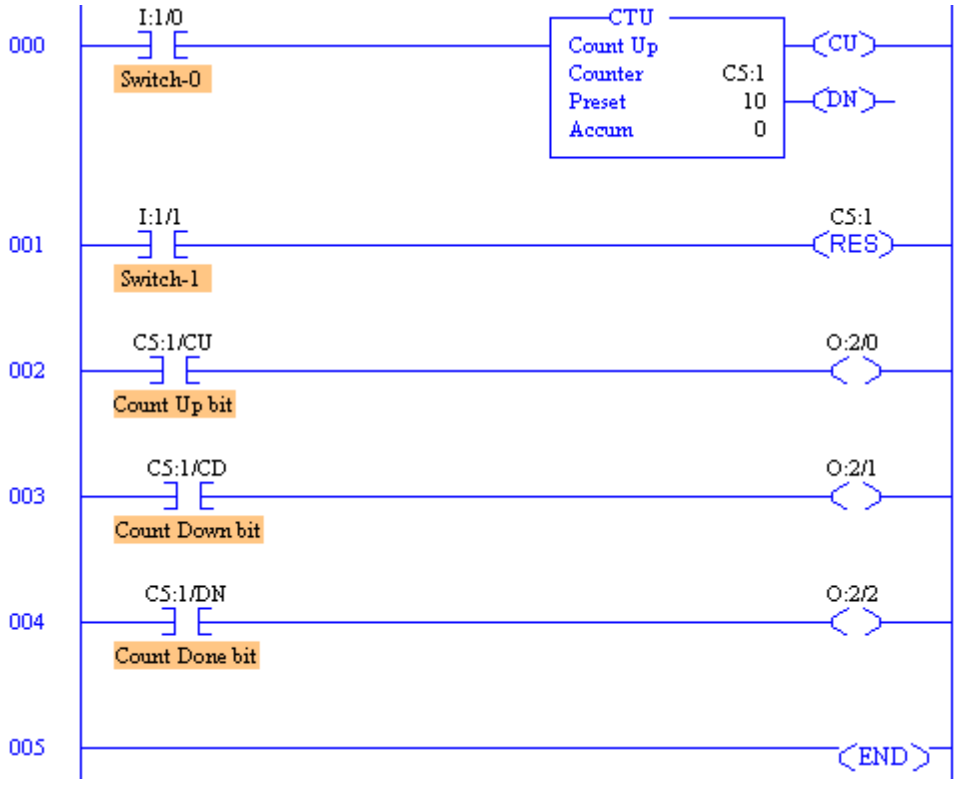
Conclusions:

An RTO timer functions the same as a TON with the exception that once it has begun timing, it holds its count of time even if the rung goes false, a fault occurs, the mode changes from RUN to PGM, or power is lost. When rung continuity returns (rung goes true again), the RTO begins timing from the accumulated time which was held when rung continuity was lost. By retaining its accumulated value, retentive timers measure the cumulative period during which rung conditions are true.

Introduction To RSLogix Counters

The CTU and RES ..... Counter Instructions

- From the LogixPro Simulations Menu, select the I/O Simulation.
- Clear out any existing program by selecting the "New" entry in the File menu, and then select the "Clear Data Table" entry in the Simulations menu.
- Now enter the following program being careful to enter the addresses exactly as shown.
- Confirm that you have entered the number 10 as the counter's preset value. This value is optionally used to set the point at which the counter's Done Bit will be Set, indicating that the count is complete.



- Once you have your program entered, and have ensured that it is correct, download it to the PLC.
- Ensure that Switch I:1/0 and I:1/1 are Open, and then place the PLC into the Run mode.
- Right click on the CTU instruction, and select "GoTo DataTable" from the drop-down menu.
- Note the initial value of Counter C5:1's accumulator and preset in the spaces below. Also indicate the state of each of the Counter's primary control bits in the spaces provided:

  Initial State (Switch I:1/0=Open): C5:1.ACC = _____ C5:1.PRE = _____ C5:1/CU = ___ C5:1/CD = ___ C5:1/DN = ___

- Open and Close switch I:1/00 a number of times and carefully observe the incrementing of C5:1's accumulator and the operation of the enable and done bits.
- Close switch I:1/01 and observe the effect that the "RES" instruction has on the counter.
- Attempt to increment the counter while switch I:1/01 is closed. You should not be able to increment the counter while the "RES" instruction is held "True".
- Open switch I:1/01 to allow the "RES" instruction to go false, and then increment the counter until the accumulator matches the preset.
- Increment the counter 2 or 3 more times and note the final value of C5:1's accumulator, preset and status bits in the spaces below.

  Final State (Switch I:1/0=Closed):
  C5:1.ACC = _____ C5:1.PRE = _____ C5:1/CU = ___ C5:1/CD = ___
  C5:1/DN = ___

Conclusions:

The CTU output instruction counts up for each false-to-true transition of conditions preceding it in the rung and produces an output (DN) when the accumulated value reaches the preset value. Rung transitions might be triggered by a limit switch or by parts traveling past a detector etc.
The ability of the counter to detect a false-to-true transitions depends on the speed (frequency) of the incoming signal. The on and off duration of an incoming signal

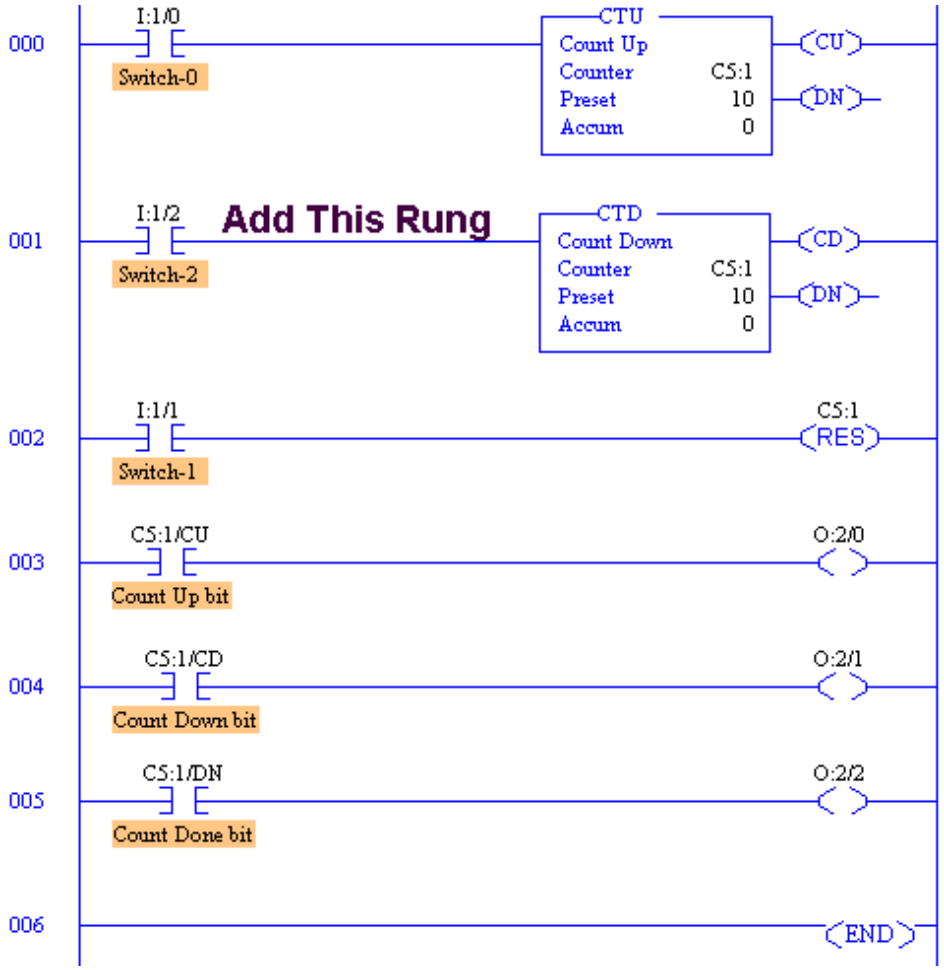
must not be faster than the scan time.

Each count (accumulator) is retained when the rung conditions again become false, permitting counting to continue beyond the preset value. This way you can base an output on the preset but continue counting to keep track of inventory/parts, etc.

Use a RES (reset) instruction with the same address as the counter, or another instruction in your program to overwrite the value of the accumulator and control bits. The on or off status of counter done, overflow, and underflow bits is retentive. The accumulated value and control bits are reset when a RES is enabled.

The CTD ..... Count Down Instruction

Ensure that switch I:1/00 and I:1/01 are open; then place the PLC into the Program mode, and Insert a new rung containing a CTD instruction just below the first rung in your program.



- Once you have completed this addition to your program, download your program to the PLC and select RUN.
- Toggle the state of switch I:1/0 continuously until the accumulator of C5:1 exceeds the preset.
- Now toggle switch I:1/02 and decrement counter C5:1 while carefully observing the status bits of the counter. Increment and decrement the counter from below zero to beyond the preset a number of times.

Conclusions:

The CTD output instruction counts down for each false-to-true transition of conditions preceding it in the rung and produces an output when the accumulated

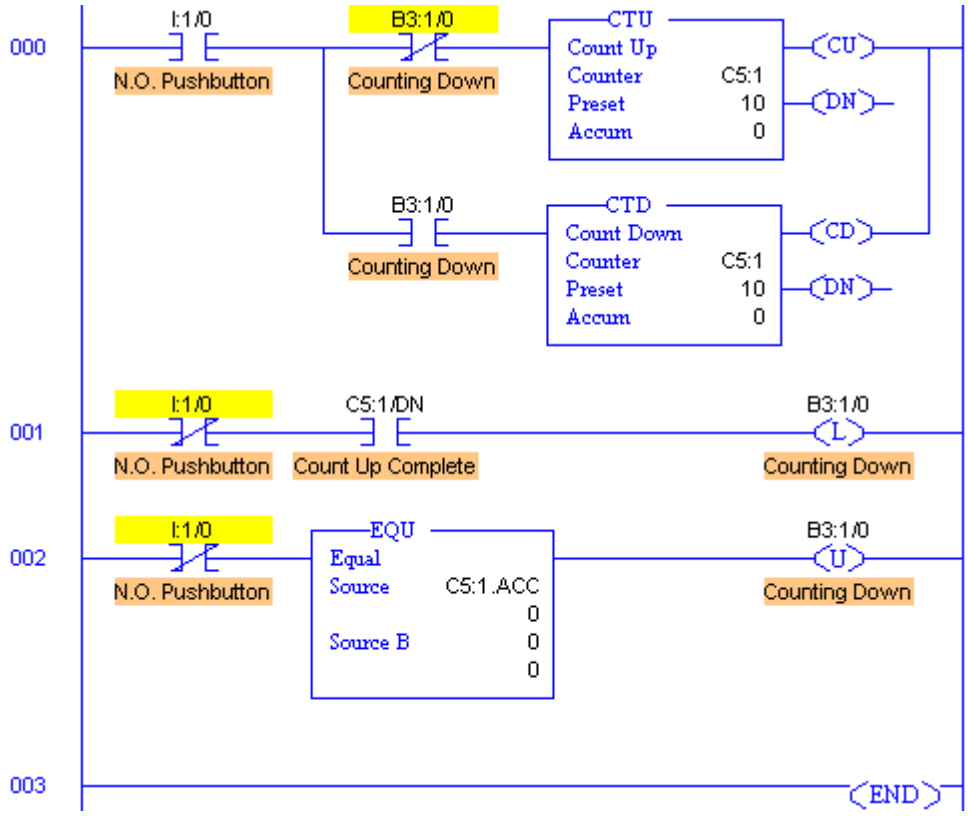
value reaches the preset value. Rung transitions might be triggered by a limit switch or by parts traveling past a detector.

Each count is retained when the rung conditions again become false. The count is retained until a RES (reset) instruction with the same address as the counter is enabled, or if another instruction in your program overwrites the value.

The accumulated value is retained after the CTU or CTD instruction goes false, and when power is removed from and then restored to the processor. Also, the on or off status of counter done, overflow, and underflow bits is retentive. The accumulated value and control bits are reset when a RES is enabled.

Applying Counter Instructions .... An Up/Down Sequence Example

- Ensure that the I/O Simulation is still selected.
- Clear your existing program by selecting the "New" entry in the File menu, and then select the "Clear Data Table" entry in the Simulations menu.
- Note the use of the "EQU" instruction in rung 2 of the following program. This input instruction will go true if the value referenced by the Source entry is "Equal" to the value contained in the Source B entry. In this example, the instruction will go true if the accumulator of the counter is equal to zero.
- Now enter the following program being careful to enter the addresses exactly as shown.



- Once you have your program entered, and have ensured that it is correct, download it to the PLC.
- Confirm that you have configured switch I:1/0 as a N.O. pushbutton then place the PLC into the "Run" mode.
- Continuously Open and Close switch I:1/00 while carefully observing the incrementing of C5:1's accumulator.
- If you have entered your program correctly, the accumulator should increment until the count of 10 is reached, and then start to decrement. When the count reaches zero, the B3:1/0 flag bit should be cleared and the up/down sequence should then repeat.
- Ensure that your program is operating as described, and carefully note how bit B3:1/0 is being employed to track and control the direction of the counting sequence.

- Note the conditions that must be present in order for bit B3:1/0 to be latched On. The first XIC instruction ensures that the latching only occurs when the pushbutton switch is released
- Delete this XIC instruction from rung 1, and then download and run your program again.
- Without the XIC instruction, the latching will occur as soon as the count of ten is reached and the CTD instruction will immediately decrement the counter back to a count of 9.
- Set the scan speed to it's lowest value, and you should be able to see that the count does reach 10, but it is then immediately decremented.

Conclusions:

The CTU is by far the most commonly used counter instruction. It can, and is utilized in almost a limitless number of counting applications, and is typically very easy to understand and employ.

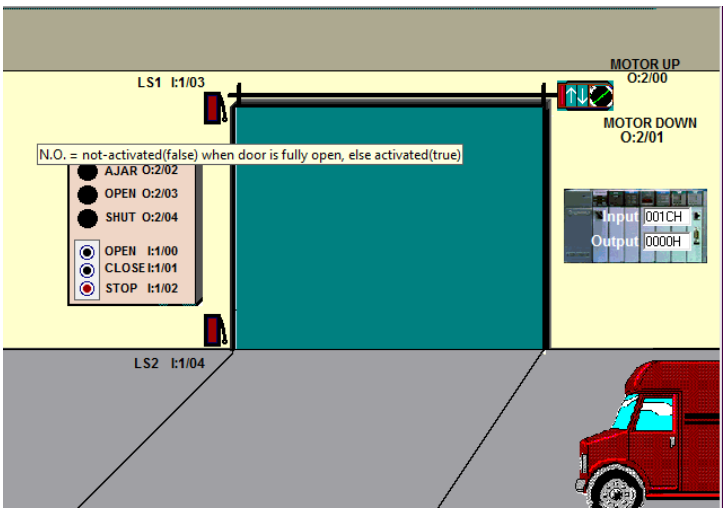
The CTD instruction is less widely employed. It is extremely useful however when paired with a CTU, where up/down counting operations are required. Cars entering and leaving a parking lot, containers being filled and then emptied are just 2 examples of where paired CTU/CTD counters might be employed.

The elegance of the CTU/CTD pairing can extract a price however in terms of ease of use and program clarity. As the last exercise highlighted, one requires a very clear understanding of the operation of these instructions and the PLC's scan sequence, in order to employ them effectively.

Lecture 6

PLC application laboratory using LogixPro

The ProSim-II Door Simulation



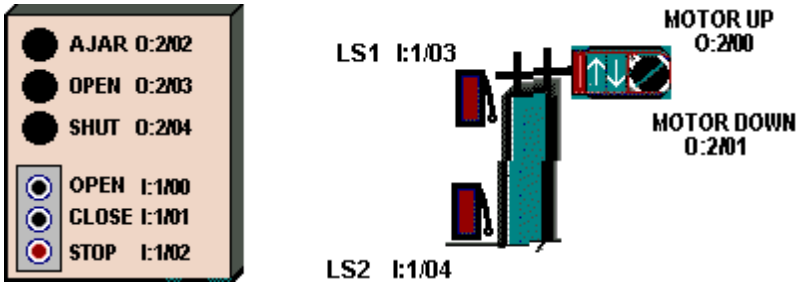
From the Simulations Menu at the top of the screen, Select the Door Simulation.

Take the time to familiarize yourself with the components used in the Door system, and take particular note of the current state of the limit switches. When the door is in the closed position, both limit switches are in their activated state (Not Normal). Run your mouse over each switch, and you should see a tool-tip text box appear, which denotes that the selected switch is wired using a set of Normally Open contacts. With the door fully closed, what signal level would you expect to see at the limit switch inputs I:1/03 and I:1/04?

To confirm your assessment of the current limit switch states, place the PLC into the RUN mode which will initiate scanning. Now open the Data Table display by

clicking on the Data Table icon located on the toolbar (3rd from right) at the top of the screen.

When you have the Data Table showing, select the "Input Table" from the drop down Table list box. You should now be able to see the current state of each bit associated with input card I:1. You should also note that bit I:1/02 is also in a High or True state. Use your mouse to press the Stop switch on the Control Panel a few times, and note the results. Don't continue on with the exercise until you are confident that you understand the rational of the observed results



Student Programming Exercise #1:

In this exercise we want you to apply your knowledge of Relay Logic Instructions to design a program which will control the ProSim-II Door. The Door System includes a Reversible Motor, a pair of Limit Switches and a Control Panel, all connected to your PLC. The program you create will monitor and control this equipment while adhering to the following criteria:

- In this exercise the Open and Close pushbuttons will be used to control the movement of the door. Movement will not be maintained when either switch is released, and therefore the Stop switch is neither required nor used in this

exercise. However, all other available Inputs and Outputs are employed in this exercise.

- Pressing the Open Switch will cause the door to move upwards (open) if not already fully open. The opening operation will continue as long as the switch is held down. If the switch is released, or if limit switch LS1 opens, the door movement will halt immediately.
- Pressing the Close Switch will cause the door to move down (close) if not already fully closed. The closing operation will continue as long as the switch is held down. If the switch is released, or if limit switch LS2 closes, the door movement will halt immediately.
- If the Door is already fully opened, Pressing the Open Switch will Not energize the motor.
- If the Door is already fully closed, Pressing the Close Switch will Not energize the motor.
- Under no circumstance will both motor windings be energized at the same time.
- The Open Lamp will be illuminated if the door is in the Fully Open position.
- The Shut Lamp will be illuminated if the door is in the Fully Closed position.

It is your responsibility to fully design, document, debug, and test your Program. Avoid the use of OTL or OTU latching instructions, and make a concerted effort to minimize the number of rungs employed.

Ensure that you have made effective use of both instruction and rung comments to clearly document your program. All I/O components referenced within your program should be clearly labeled, and rung comments should be employed to add additional clarity as required.

Student Programming Exercise #2:

In this exercise we want you to apply your knowledge of Relay Logic Instructions to design a program which will maintain the appropriate door movement once initiated by the operator. The Opening or Closing operation of the door will continue to completion even if the operator releases the pushbutton which initiated the movement. The program will adhere to the following criteria:

- Door movement will halt immediately when the Stop Switch is initially pressed, and will remain halted if the switch is released.
- Pressing the Open Switch will cause the door to Open if not already fully open. The opening operation will continue to completion even if the switch is released.
- Pressing the Close Switch will cause the door to Close if not already fully shut. The closing operation will continue to completion even if the Switch is released.

- If the Door is already fully opened, Pressing the Open Switch will Not energize the motor.
- If the Door is already fully closed, Pressing the Close Switch will Not energize the motor.
- Under no circumstance will both motor windings be energized at the same time.
- The Ajar Lamp will be illuminated if the door is NOT in either the fully closed or fully opened position.
- The Open Lamp will be illuminated if the door is in the Fully Open position.
- The Shut Lamp will be illuminated if the door is in the Fully Closed position.

It is your responsibility to fully design, document, debug, and test your Program. Avoid the use of OTL or OTU latching instructions, and make a concerted effort to minimize the number of rungs employed.

As before, ensure that you have made effective use of both instruction and rung comments to clearly document your program.

Student Programming Exercise #3:

In this exercise we want to introduce you to a simple programming technique for adding a bit of "Flash" to your program. We want you to make use of the PLC's

Free Running Timer which can be viewed in the Data Table Display at location S2:4. This integer word contains a count which is incremented continuously by the PLC when it is in the Run mode, and it can come in quite handy at times for variety of purposes. In this exercise we want you to utilize this word as follows:

With the PLC in the Run mode, Display word S2:4 utilizing the Data Table display. Ensure that the Radix is set to Binary so that you can view the individual bits within the word. You should see a binary count in progress where the rate of change of each bit is directly related to it's position within the word. Bit 0 will have the highest rate, while Bit 1 will be 1/2 as fast as Bit 0, and Bit 2 half as fast as 1 etc. etc.

We want you to add a Lamp Flasher to your program by monitoring the state of one of these bits with an XIC instruction. I'm going to suggest using Bit 4 for this purpose, but depending upon the speed of your computer you may elect to substitute another Bit. With an actual AB PLC, the rate is consistent, but with LogixPro it varies from computer to computer.

Place an XIC instruction addressed to S:4/4 on the rung which controls either the Open or Shut Lamp in your previous program. Now download and Run this modified program to see the flashing effect achieved. The Lamp should flash at a reasonable rate whenever your program energizes the selected Lamp.

Now modify your program so that the following criteria is met:

- If the Door is fully open, the Open lamp will be energized but not flashing as was the case before.
- If the Door is opening, the Open lamp will flash while the door is in motion.

- If the Door is fully closed, the Shut lamp will be energized but not flashing as was the case before.
- If the Door is closing, the Shut lamp will flash while the door is in motion.
- The Ajar Lamp will flash if the door is stationary, and is not in the fully open or fully closed position. The Ajar Lamp will flash at a slower rate (1/4) then the other lamps.
- The Ajar Lamp will be illuminated in a steady state if the door is in motion.

As before, ensure that you have made effective use of both instruction and rung comments to clearly document your program.

Supplemental Programming Exercise #4:

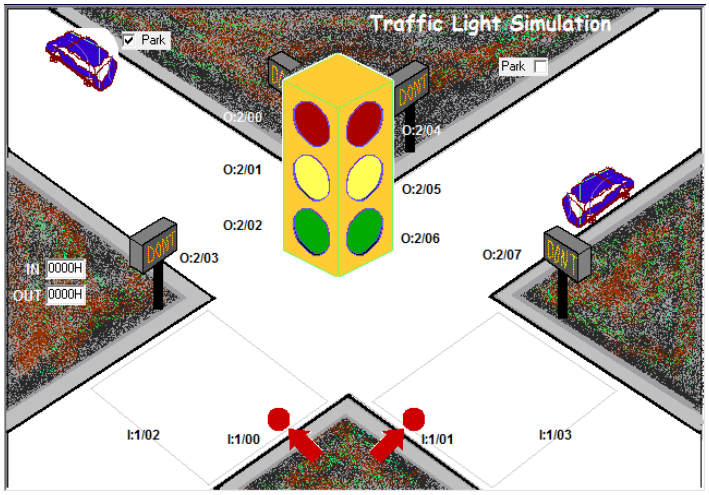
We do not recommend proceeding with this exercise if you do not have an instructor or experienced PLC programmer to call upon for assistance.

In this exercise we want you to modify your program so that it adheres to this additional criteria:

- If the door is currently opening, pressing the Close Switch will immediately halt movement. Door movement will remain halted when the switch is released.

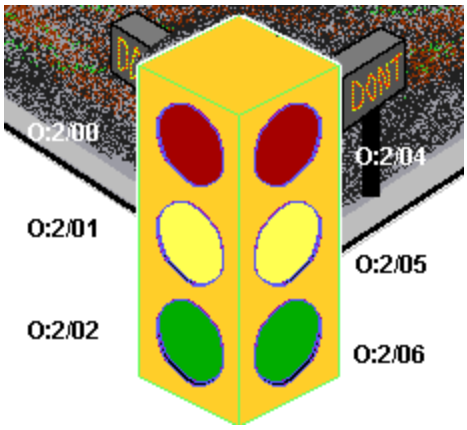
- If the door is currently closing, pressing the Open Switch will immediately halt movement. Door movement will remain halted when the switch is released.
- Once movement is halted by the either of the foregoing actions, the operating criteria associated with the previous exercise will again take effect.
- The utilization of Binary or Integer Table bits to Flag specific conditions within your program would be appropriate. Also, the retentive OTL and OTU instructions may be utilized freely at your discretion

Traffic Control Exercises Utilizing TON Timers



Exercise #1 -- Traffic Control using 3 Lights

From the Simulations Menu at the top of the screen, Select the Traffic Light Simulation



Using your knowledge of cascading timers, develop a ladder logic program which will sequence a set of green, amber and red lights in the following manner:

Sequence of Operation:

1. Light O:2/00 (Red) = 12 seconds ON
2. Light O:2/02 (Green) = 8 seconds ON
3. Light O:2/01 (Amber) = 4 seconds ON
4. The sequence now repeats with Red = ON.

RED	GREEN	AMBER
12 Sec.	8 Sec.	4 Sec.

Exercise #2 -- Traffic Control using 6 Lights

Modify your program so that the 3 lights which represent the other traffic direction are also controlled. It is tempting to use six timers for this task, but the job can be done with just four, and you'll end up with a much cleaner program as a bonus.

Red = O:2/00		Green = O:2/02	Amber = O:2/01
Green = O:2/06	Amber = O:2/05	Red = O:2/04	
8 Sec.	4 Sec.	8 Sec.	4 Sec.

Still getting the odd Crash? Well it's pretty obvious that these drivers aren't paying much attention to Amber Lights! No need for any more wiring however. You can solve this problem, but it's going take a little more programming.

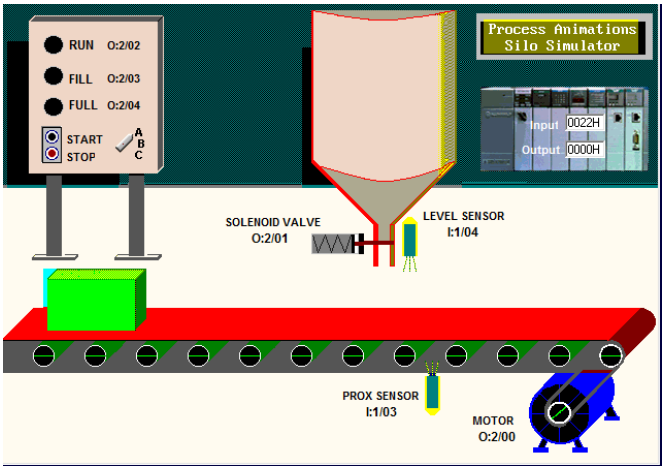
Exercise #3 -- Traffic Light With Delayed Green

Modify your program so that there is a 1 second period when both directions will have their RED lights illuminated. Note that the timing diagram below only shows one of these 1 second intervals but two are actually required. Work the problem out, and try to keep the Timer count down to six.

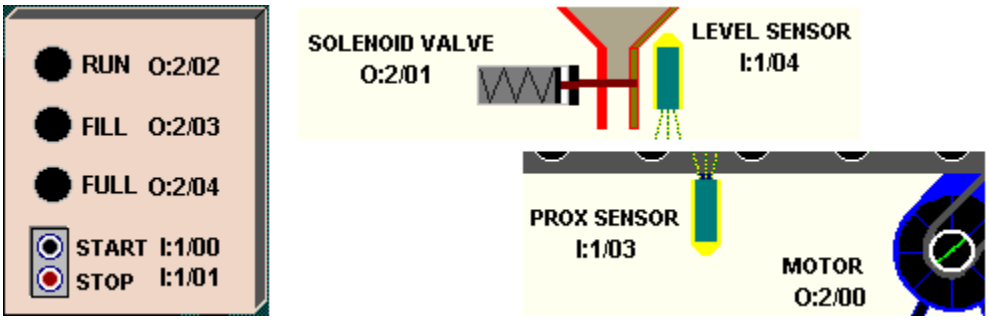
Red = O:2/00			Green = O:2/02	Amber = O:2/01
Green = O:2/06	Amber = O:2/05	Red = O:2/04		
8 Sec.	4 Sec.	1	8 Sec.	4 Sec.

If a one second delay is not enough to get these drivers under control then just go ahead and jack the delay up to two!

The Silo Lab Utilizing Relay Logic



From the Simulations Menu at the top of the screen, Select the Silo Simulation



Exercise #1 -- Continuous Operation

Completely design and de-bug a ladder control circuit which will automatically position and fill the boxes which are continuously sequenced along the conveyor. Ensure that the following details are also met:

- The sequence can be stopped and re-started at any time using the panel mounted Stop and Start switches.

- The RUN light will remain energized as long as the system is operating automatically.
- The RUN light, Conveyor Motor and Solenoid will de-energize whenever the system is halted via the STOP switch.
- The FILL light will be energized while the box is filling.
- The FULL light will energize when the box is full and will remain that way until the box has moved clear of the prox-sensor.

Exercise #2 -- Container Filling with Manual Restart

Alter or re-write your program so that it incorporates the following changes:

- Stop the conveyor when the right edge of the box is first sensed by the prox-sensor.
- With the box in position and the conveyor stopped, open the solenoid valve and allow the box to fill. Filling should stop when the Level sensor goes true.
- The FILL light will be energized while the box is filling.
- The FULL light will energize when the box is full and will remain that way until the box has moved clear of the prox-sensor.
- Once the box is full, momentarily pressing the Start Switch will move the box off the conveyor and bring a new box into position. Forcing the operator

to hold the Start button down until the box clears the prox-sensor is not acceptable.

Exercise #3 -- Selectable Mode of Operation

Alter or re-write your program so that the panel mounted Selector switch can be utilized to select one of 3 different modes of operation. The 3 modes shall operate as follows:

- When the selector switch is in position "A", the system shall operate in the "Continuous" mode of operation. This is the mode of operation which was used in Exercise #1.
- When the selector switch is in position "B", the system shall operate in the "Manual Restart" mode of operation. This is the mode of operation which was used in Exercise #2.
- When the selector switch is in position "C", the system shall operate in the "Fill Bypass" mode of operation. In this mode, the boxes will simply move down the conveyor continuously and bypass the fill operation. As in the other modes, the Start and Stop pushbuttons will control the conveyor motion and the Run Lamp will operate as expected.

References for more reading

1. PROGRAMMABLE CONTROLLERS- THEORY AND IMPLEMENTATION, 2e, by L. A. Bryan
2. PROGRAMMABLE CONTROLLERS, 3e, by Frank D.Petruzella
3. Automating Manufacturing Systems with PLCs, 2005, by Hugh Jack
4. Programmable Logic Controllers, 4th Edition, 2006 by: W. Bolton
5. The Student RSLogix Programming Exercises Manual -The Learning Pit