

NEAR EAST UNIVERSITY



Faculty of Engineering

**Department of Electrical and Electronic
Engineering**

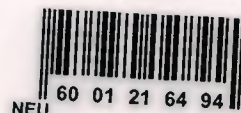
PROGRAMMABLE LOGIC CONTROLLERS

**Graduation Project
EE- 400**

Student: Abed Yahya TAFESH (20001276)

Supervisor: Mr. Özgür ÖZERDEM

Nicosia - 2003





ACKNOWLEDGMEN

For the time being I would like to thank my parents and all my family for endless support, and for their encouragement and believing in my abilities to complete my studies and to become an engineer.

I would like to express my deepest gratitude to Mr. Özgür ÖZERDEM to be my supervisor, where the guiding of his successfully helped to overcome many and to learn a lot about PLC. I also wish to extend my thanks to the staff members of NEAR EAST UNIVERSITY.

Finally, I must say a special thanks to my friends: Deniz, Yousef Baalousha, Ashraf Farah (debus), Ahmad Abu Ayyash, IBO and Loay, Majdi. I will never forget our nice time, I hope to them all more success in their future life. I also want to thank my friends in NEU: Musa, Hassan, Ata, and Alaa...

TABLE OF CONTENTS

ACKNOWLEDGMENT	i
INTRODUCTION	ii
1. PLC STRUCTURE AND OPERATION	1
1.1. What Is The PLC?	1
1.2. First Programmable Controllers	2
1.3. Why Use PLCs?	3
1.4. PLCs - Hardware Design	5
1.4.1. The Central Processing Unit	6
1.4.2. Memory	7
1.4.3. Power Supply	8
1.4.4. PLC Controller Input	8
1.4.5. PLC Controller Output	9
1.4.6. Extension Lines	10
1.5. PLC Operation	11
1.5.1. Response Time	12
1.5.2. Data Areas	13
1.5.3. Data Object	14
2. PROGRAMMING THE PLC	15
2.1. Ladder Diagram	15
2.2. Normally Open and Normally Closed Contacts	17
2.3. Combination Logic	21
2.4. Complex Gate Form	23
2.5. Multiplexes	23
2.6. Instruction Lists	25
2.7. Brach Code	27
2.8. Timers	28
2.9. Counters	31

3. PLC COMMUNICATION	36
3.1. Introduction	36
3.2. Parallel Communication	36
3.3. Serial Communication	36
3.3.1. RS 232	36
3.3.2. RS 423 and RS 485	42
3.4. Local Area Network	42
3.4.1. Response Time of Network	43
3.4.2. Network Standards	44
4. APPLIED PROGRAM	45
4.1. Introduction	45
4.2. Working process	46
4.3. Ladder diagram of the system	47
4.4. Statement list of the system	50
Conclusion	52
References	53

INTRODUCTION

A programmable logic controller (PLC), which was defined by Capiel in 1982, is used in many “real world” applications. If there is industry present, chances are good that there is a PLC present. If you are involved in machining, packaging, material handling, automated assembly or countless other industries you are probably already using them. If you are not, you are wasting money and time. Almost any application that needs some type of electrical control has a need for a PLC.

In a traditional industrial control system, all control devices are wired directly to each other according to how the system is supposed to operate. In a PLC system, however, the PLC replaces the wiring between the devices. Thus, instead of being wired directly to each other, all equipment is wired to the PLC. Then, the control program inside the PLC provides the “wiring” connection between the devices.

However, this project consists of three chapters.

Chapter one introduces the PLC structure and operation, in this chapter we presented the development of PLC, its advantages and the basic elements of PLC controller.

Chapter two discusses PLC programming, contactors, combination gates, ladder diagram and statement list

Chapter three studies PLC communications interfaces, options of communicating devices, local area networks and network standards.

Chapter four shows the applied program, which is about production line in a factory. Ladder diagram and the instruction list were provided.

1. PLC STRUCTURE AND OPERATION

1.1 What is a PLC?

Industry has begun to recognize the need for quality improvement and increase in productivity in the sixties and seventies. Flexibility also became a major concern (ability to change a process quickly became very important in order to satisfy consumer needs).

Try to imagine automated industrial production line in the sixties and seventies. There was always a huge electrical board for system controls, and not infrequently it covered an entire wall! Within this board there were a great number of interconnected electromechanical relays to make the whole system work. By word "connected" it was understood that electrician had to connect all relays manually using wires! An engineer would design logic for a system, and electricians would receive a schematic outline of logic that they had to implement with relays. These relay schemas often contained hundreds of relays. The plan that electrician was given was called "ladder schematic". Ladder displayed all switches, sensors, motors, valves, relays, etc. found in the system. Electrician's job was to connect them all together. One of the problems with this type of control was that it was based on mechanical relays. Mechanical instruments were usually the weakest connection in the system due to their moveable parts that could wear out. If one relay stopped working, electrician would have to examine an entire system (system would be out until a cause of the problem was found and corrected).

The other problem with this type of control was in the system's break period when a system had to be turned off, so connections could be made on the electrical board. If a firm decided to change the order of operations (make even a small change), it would turn out to be a major expense and a loss of production time until a system was functional again.

It's not hard to imagine an engineer who makes a few small errors during his project. It is also conceivable that electrician has made a few mistakes in connecting the system. Finally, you can also imagine having a few bad components. The only way to see if everything is all right is to run the system. As systems are usually not perfect with

a first try, finding errors was an arduous process. You should also keep in mind that a product could not be made during these corrections and changes in connections. System had to be literally disabled before changes were to be performed. That meant that the entire production staff in that line of production was out of work until the system was fixed up again. Only when electrician was done finding errors and repairing,, the system was ready for production. Expenditures for this kind of work were too great even for well-to-do companies.

1.2 First programmable controllers

General Motors is among the first who recognized a need to replace the system's "wired" control board. Increased competition forced automakers to improve production quality and productivity. Flexibility and fast and easy change of automated lines of production became crucial! General Motors' idea was to use for system logic one of the microcomputers (these microcomputers were as far as their strength beneath today's eight-bit microcontrollers) instead of wired relays. Computer could take place of huge, expensive, inflexible wired control boards. If changes were needed in system logic or in order of operations, program in a microcomputer could be changed instead of rewiring of relays. Imagine only what elimination of the entire period needed for changes in wiring meant then. Today, such thinking is but common, and then it was revolutionary!

Everything was well thought out, but then a new problem came up of how to make electricians accept and use a new device. Systems are often quite complex and require complex programming. It was out of question to ask electricians to learn and use computer language in addition to other job duties. General Motors Hydromatic Division of this big company recognized a need and wrote out project criteria for first programmable logic controller (there were companies which sold instruments that performed industrial control, but those were simple sequential controllers ù not PLC controllers as we know them today). Specifications required that a new device be based on electronic instead of mechanical parts, to have flexibility of a computer, to function in industrial environment (vibrations, heat, dust, etc.) and have a capability of being reprogrammed and used for other tasks. The last criterion was also the most important, and a new device had to be programmed easily and maintained by electricians and technicians. When the specification was done, General Motors looked for interested

companies, and encouraged them to develop a device that would meet the specifications for this project.

Gould Modicon developed a first device that met these specifications. The key to success with a new device was that for its programming you didn't have to learn a new programming language. It was programmed so that same language via ladder diagram, already known to technicians was used. Electricians and technicians could very easily understand these new devices because the logic looked similar to old logic that they were used to working with. Thus they didn't have to learn a new programming language which (obviously) proved to be a good move. PLC controllers were initially called PC controllers (programmable controllers). This caused a small confusion when Personal Computers appeared. To avoid confusion, a designation PC was left to computers, and programmable controllers became programmable logic controllers. First PLC controllers were simple devices. They connected inputs such as switches, digital sensors, etc., and based on internal logic they turned output devices on or off. When they first came up, they were not quite suitable for complicated controls such as temperature, position, pressure, etc. However, throughout years, makers of PLC controllers added numerous features and improvements. Today's PLC controller can handle highly complex tasks such as position control, various regulations and other complex applications. The speed of work and easiness of programming were also improved. Also, modules for special purposes were developed, like communication modules for connecting several PLC controllers to the net. Today it is difficult to imagine a task that could not be handled by a PLC.

1.3 Why use PLCs?

The softwiring advantage provided by programmable controllers is tremendous. In fact, it is one of the most important features of PLCs. Softwiring makes changes in the control system easy and cheap. If you want a device in a PLC system to behave differently or to control a different process element, all you have to do is change the control program. In a traditional system, making this type of change would involve physically changing the wiring between the devices, a costly and time-consuming endeavor.

In addition to the programming flexibility we just mentioned, PLCs offer other advantages over traditional control systems. These advantages include:

- High reliability, due to the ability of the control to process data and react quickly in real time results in consistent part production. This high speed is due in a large measure to the utilization of distributed processing concept.
- Increased control over manufacturing, due to the ready availability of all process data. A PLC can be easily connected to a host computer and its data can be made available for subsequent processing. To this end, major manufacturers offer networks that can be readily installed.
- Scalability which means that the control system can be easily and economically scaled up, or scaled down, to meet changing production and budget requirements. A large, dedicated control system once installed and operating is very difficult to modify.
- Environmental resistibility, an industrial application needs industrial solutions. The vibration resistibility, IP class of the housing and electromagnetic compatibility and wider operation temperature are essentials industrial systems. PLC's fit better to industrial environment since they are intentionally built for this purpose.
- Service and spare parts, there are still no common PLC instruction of manufacturers. PLC manufacturers provide PLC's on market until the parts are obsolete. Some new products are up compatible with the obsolete one. Mostly all manufacturers PLC programming software may convert the obsolete versions to a up-to-date -version. The up compatibility is not guaranteed and may need to be translated to new PLC instructions.
- Communication abilities with other equipment's, PLC's are designed to construct a network between them. Communication between same brand and type of PLC's is not a problem. When multiple brands and types are used to construct the application, the communications are a real problem

The comparison of the PLC with other control systems is given in the table 1.2 it is obvious that PLC's are the best for industrial control.

Table 1.2 The comparison of PLC with relays, PC, and design system

Characteristic	Relay system	Digital logic	Computer	PLC system
Price per function	Fairly low	Low	High	Low
Physical size	Bulky	Very compact	Fairly compact	Very compact
Operation speed	Slow	Very fast	Fairly fast	Fast
Electrical noise immunity	Excellent	Good	Quite good	Good
Installation	Time-consuming to design and install	Design time-consuming	Programming extremely time-consuming	Simple to program and install
Capable of complicated operation	No	Yes	Yes	Yes
Ease of changing function	Very difficult	Difficult	Quite simple	Very simple
Ease of maintenance	Poor-large number of contacts	Poor if IC s soldered	Poor-several custom boards	Good-few standard cards

1.4 PLCs – Hardware design

Programmable controllers are purpose-built computers consisting of three functional areas, which are shown, in figure 1.1, central processing unit (CPU), memory and input/output. Input conditions to the PLC are sensed then stored in memory, where the PLC performs the programmed logic instructions on these input states. Output conditions are then generated to drive associated equipment. The action taken depends totally on the control program held in memory. In smaller PLCs individual printed circuit cards within a single compact unit perform these functions, whilst larger PLCs are constructed on a modular basis with function modules slotted into the backplane connectors of the mounting rack. This allows simple expansion of the system when necessary. In both these cases the individual circuit boards are easily removed and replaced, facilitating rapid repair of the system should faults develop.

In addition a programming unit is necessary to download control programs to the PLC memory.

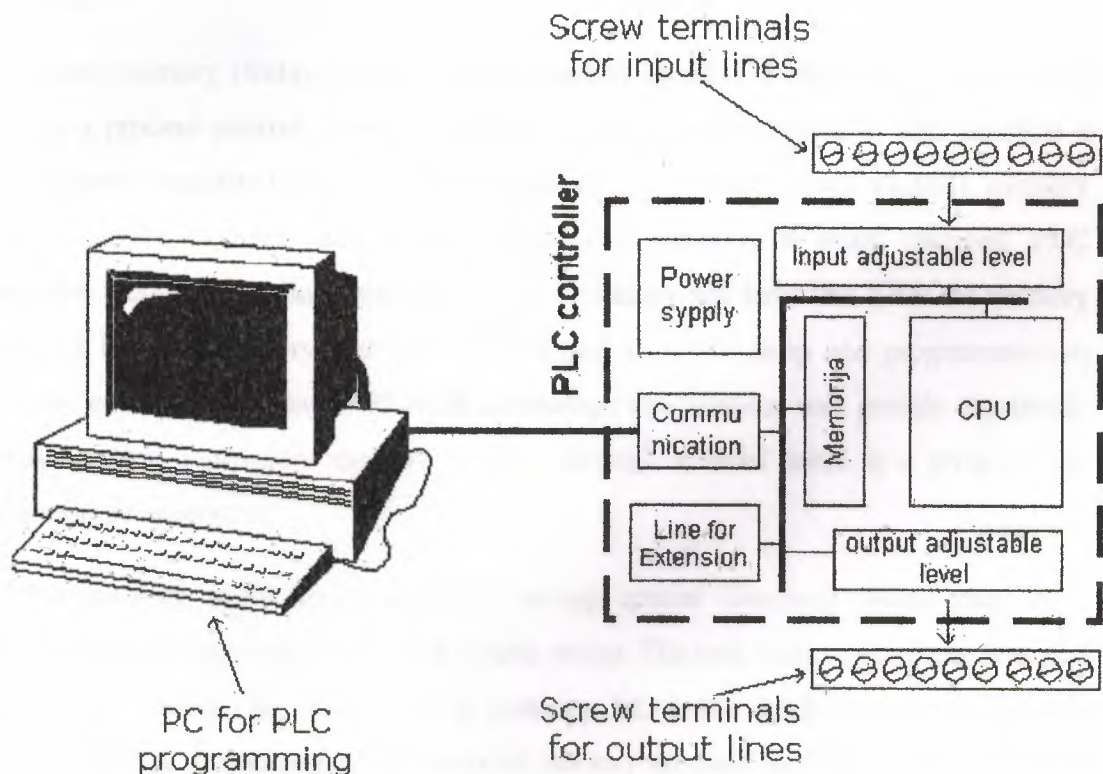


Figure 1.1 Basic element of PLC controller

1.4.1 The Central Processing Unit

Central Processing Unit (CPU) is the brain of a PLC controller. CPU itself is usually one of the microcontrollers. Aforetime these were 8-bit microcontrollers such as 8051, and now these are 16- and 32-bit microcontrollers. Unspoken rule is that you'll find mostly Hitachi and Fujicu microcontrollers in PLC controllers by Japanese makers, Siemens in European controllers, and Motorola microcontrollers in American ones. CPU also takes care of communication, interconnectedness among other parts of PLC controller, program execution, memory operation, overseeing input and setting up of an output. PLC controllers have complex routines for memory checkup in order to ensure that PLC memory was not damaged (memory checkup is done for safety reasons). Generally speaking, CPU unit makes a great number of check-ups of the PLC controller itself so eventual errors would be discovered early. You can simply look at any PLC controller and see that there are several indicators in the form of light diodes for error signalization.

1.4.2 Memory

System memory (today mostly implemented in FLASH technology) is used by a PLC for a process control system. Aside from this operating system it also contains a user program translated from a ladder diagram to a binary form. FLASH memory contents can be changed only in case where user program is being changed. PLC controllers were used earlier instead of FLASH memory and have had EPROM memory instead of FLASH memory that had to be erased with UV lamp and programmed on programmers. With the use of FLASH technology this process was greatly shortened. Reprogramming a program memory is done through a serial cable in a program for application development.

User memory is divided into blocks having special functions. Some parts of a memory are used for storing input and output status. The real status of an input is stored either as "1" or as "0" in a specific memory bit. Each input or output has one corresponding bit in memory. Other parts of memory are used to store variable contents for variables used in user program. For example, timer value, or counter value would be stored in this part of the memory. There are four types of memory, first one is random

access memory (RAM), and second one is read only memory (ROM), third one is erasable programmable read only memory (EPROM), and the last one is electrically erasable programmable read only memory (EEPROM).

1.4.3 Power supply

Electrical supply is used in bringing electrical energy to central processing unit. Most PLC controllers work either at 24 VDC or 220 VAC. On some PLC controllers you'll find electrical supply as a separate module. Those are usually bigger PLC controllers, while small and medium series already contain the supply module. User has to determine how much current to take from I/O module to ensure that electrical supply provides appropriate amount of current. Different types of modules use different amounts of electrical current. This electrical supply is usually not used to start external inputs or outputs. User has to provide separate supplies in starting PLC controller inputs or outputs because then you can ensure so called "pure" supply for the PLC controller. With pure supply we mean supply where industrial environment can not affect it damagingly. Some of the smaller PLC controllers supply their inputs with voltage from a small supply source already incorporated into a PLC.

1.4.4 PLC controller input

Intelligence of an automated system depends largely on the ability of a PLC controller to read signals from different types of sensors and input devices. Keys, keyboards and by functional switches are a basis for man versus machine relationship. On the other hand, in order to detect a working piece, view a mechanism in motion, check pressure or fluid level you need specific automatic devices such as proximity sensors, marginal switches, photoelectric sensors, level sensors, etc. Thus, input signals can be logical (on/off) or analogue. Smaller PLC controllers usually have only digital input lines while larger also accept analogue inputs through special units attached to PLC controller. One of the most frequent analogue signals are a current signal of 4 to 20 mA and millivolt voltage signal generated by various sensors. Sensors are usually used as inputs for PLCs. You can obtain sensors for different purposes. They can sense presence of some parts, measure temperature, pressure, or some other physical

dimension, etc. (ex. inductive sensors can register metal objects).

Other devices also can serve as inputs to PLC controller. Intelligent devices such as robots, video systems, etc. often are capable of sending signals to PLC controller input modules (robot, for instance, can send a signal to PLC controller input as information when it has finished moving an object from one place to the other).

- **Input adjustment interface**

Adjustment interface also called an interface is placed between input lines and a CPU unit. The purpose of adjustment interface to protect a CPU from disproportionate signals from an outside world. Input adjustment module turns a level of real logic to a level that suits CPU unit (ex. input from a sensor which works on 24 VDC must be converted to a signal of 5 VDC in order for a CPU to be able to process it). This is typically done through opto-isolation, and this function you can view in the figure 1.2.

Opto-isolation means that there is no electrical connection between external world and CPU unit. They are "optically" separated, or in other words, signal is transmitted through light. The way this works is simple. External device brings a signal which turns LED on, whose light in turn incites photo transistor which in turn starts conducting, and a CPU sees this as logic zero (supply between collector and transmitter falls under 1V). When input signal stops LED diode turns off, transistor stops conducting, collector voltage increases, and CPU receives logic 1 as information.

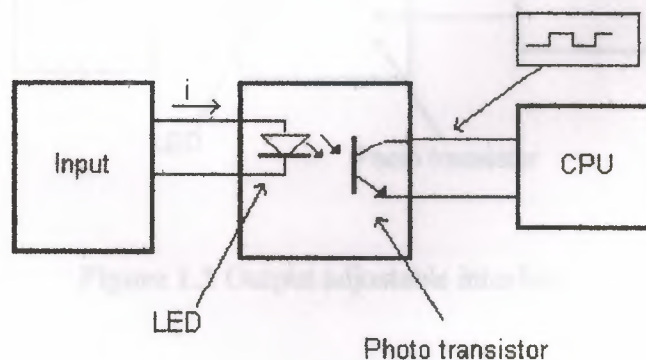


Figure 1.2 Input adjustable interface

1.4.5 PLC controller output

Automated system is incomplete if it is not connected with some output devices. Some of the most frequently used devices are motors, solenoids, relays, indicators, sound signalization and similar. By starting a motor, or a relay, PLC can manage or control a simple system such as system for sorting products all the way up to complex systems such as service system for positioning head of CNC machine. Output can be of analogue or digital type. Digital output signal works as a switch; it connects and disconnects line. Analogue output is used to generate the analogue signal (ex. motor whose speed is controlled by a voltage that corresponds to a desired speed).

- **Output adjustment interface**

Output interface is similar to input interface. CPU brings a signal to LED diode and turns it on. Light incites a photo transistor which begins to conduct electricity, and thus the voltage between collector and emitter falls to 0.7V, and a device attached to this output sees this as a logic zero. Inversely it means that a signal at the output exists and is interpreted as logic one. Phototransistor is not directly connected to a PLC controller output. Between phototransistor and an output usually there is a relay or a stronger transistor capable of interrupting stronger signals. And this function you can view in the figure 1.3.

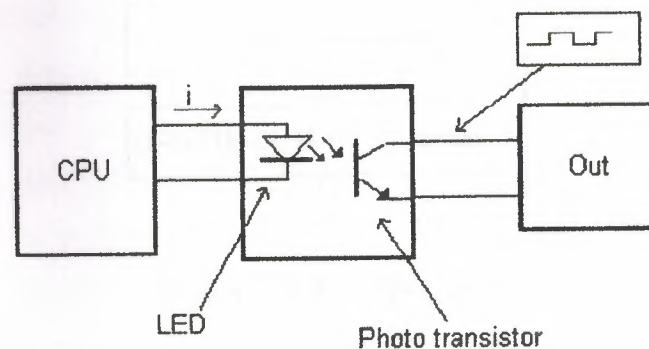


Figure 1.3 Output adjustable interface

1.4.6 Extension lines

Every PLC controller has a limited number of input/output lines. If needed, this number can be increased through certain additional modules by system extension through extension lines. Each module can contain extension both of input and output lines. Also, extension modules can have inputs and outputs of a different nature from those on the PLC controller (ex. in case relay outputs are on a controller, transistor outputs can be on an extension module).

1.5 PLC Operation

A PLC works by continually scanning a program. We can think of this scan cycle as consisting of 3 important steps which are shown in figure 1.4. There are typically more than 3 but we can focus on the important parts and not worry about the others. Typically the others are checking the system and updating the current internal counter and timer values.

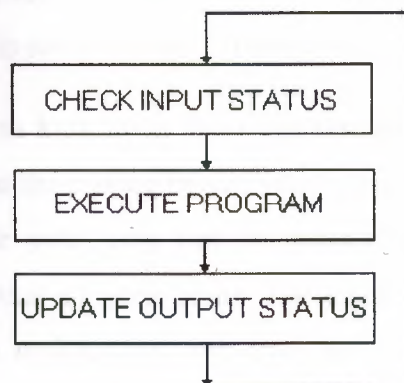


Figure 1.4 PLC operation

Step 1-CHECK INPUT STATUS-First the PLC takes a look at each input to determine if it is on or off. In other words, is the sensor connected to the first input on? How about the second input? How about the third... It records this data into its memory to be used during the next step.

Step 2-EXECUTE PROGRAM-Next the PLC executes your program one instruction at a time. Maybe your program said that if the first input was on, then it

should turn on the first output. Since it already knows which inputs are on/off from the previous step it will be able to decide whether the first output should be turned on based on the state of the first input. It will store the execution results for use later during the next step.

Step 3-UPDATE OUTPUT STATUS-Finally the PLC updates the status of the outputs. It updates the outputs based on which inputs were on during the first step and the results of executing your program during the second step. Based on the example in step 2, it would now turn on the first output because the first input was on and your program said to turn on the first output when this condition is true.

After the third step the PLC goes back to step one and repeats the steps continuously. One scan time is defined as the time it takes to execute the 3 steps listed above.

1.5.1 Response Time

The total response time of the PLC is a fact we have to consider when shopping for a PLC. Just like our brains, the PLC takes a certain amount of time to react to changes. In many applications speed is not a concern, in others though.

If you take a moment to look away from this text you might see a picture on the wall. Your eyes actually see the picture before your brain says "Oh, there's a picture on the wall". In this example your eyes can be considered the sensor. The eyes are connected to the input circuit of your brain. The input circuit of your brain takes a certain amount of time to realize that your eyes saw something. (If you have been drinking alcohol this input response time would be longer!) Eventually your brain realizes that the eyes have seen something and it processes the data. It then sends an output signal to your mouth. Your mouth receives this data and begins to respond to it. Eventually your mouth utters the words "Gee, that's a really ugly picture!". Notice in this example we had to respond to 3 things that are shown in the figure 1.5.

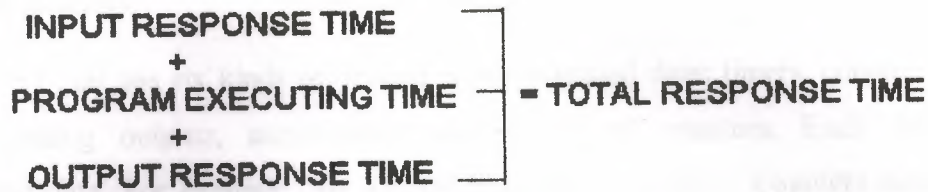


Figure1.5 Time response

- 1) INPUT- it took a certain amount of time for the brain to notice the input signal from the eyes.
- 2) EXECUTION- it took a certain amount of time to process the information received from the eyes. Consider the program to be: If the eyes see an ugly picture then output appropriate words to the mouth.
- 3) OUTPUT- The mouth receives a signal from the brain and eventually spits (no pun intended) out the words "Gee, that's a really ugly picture!"

1.5.2 Data areas (For Siemens Simatic S7-200)

Data memory contains variable memory, and output image registers, internal memory bits, and special memory bits. This memory is accessed by a byte bit convention. For example to access bit of Variable Memory byte 2 you would use the address V2.3. Table 1.2 shows the identifiers and ranges for each of the data memory types.

Table 1.2 Identifiers and ranges for of CPU 212 and CPU 214

Area identifier	Data area	CPU 212	CPU 214
I	Input	I0.0 to I7.7	I0.0 to I 7.7
Q	Output	Q0.0 to Q7.7	Q0.0 to Q7.7
M	Internal memory	M0.0 to M15.7	M0.0 to M31.7
SM	Special memory	SM0.0 to SM4507	SM0.0 to SM85.7
V	Variable memory	V0.0 to V1023.7	V0.0 to V4095.7

1.5.3 Data object

The S7-200 has six kinds of devices with associated data: timers, counters, analog inputs, analog outputs, accumulator and high-speed counters. Each device has associated data. For example, the S7-200 has counter devices. Counters have a data value that maintains the current count value. There is also a bit value, which is set when the current value is greater than or equal to the present value. Since there are multiple devices are numbered from 0 to n. The corresponding data objects and object bits are also numbered. The table 1.3 shows the identifiers and ranges for each of the data object memory types.

Table 1.3 Identifiers and ranges for CPU212 and CPU214

Object identifier	Object	CPU 212	CPU 214
T	Timers	T0 to T63	T0 to T127
C	Counters	Co to C63	C0 to C127
AI	Analog input	AIW0 to AIW30	AIW0 to AIW30
AQ	Analog output	AQW0 to AQW30	AQW0 to AQW30
AC	Accumulator	AC0 to AC3	AC0 to AC3
HC	High-speed counter	HC0	HC0 to HC2

2. PROGRAMMING THE PLC

2.1 Ladder diagram

Programmable controllers are generally programmed in ladder diagram (or “relay diagram”) which is nothing but a symbolic representation of electric circuits. Symbols were selected that actually looked similar to schematic symbols of electric devices, and this has made it much easier for electricians to switch to programming PLC controllers. Electrician who has never seen a PLC can understand a ladder diagram.

There are several languages designed for user communication with a PLC, among which ladder diagram is the most popular. Ladder diagram consists of one vertical line found on the left-hand side, and lines, which branch off to the right. Line on the left is called a “bus bar”, and lines that branch off to the right are instruction lines. Conditions, which lead to instructions positioned at the right edge of a diagram, are stored along instruction lines. Logical combination of these conditions determines when and in what way instruction on the right will execute. Basic elements of a relay diagram can be seen in figure 2.1.

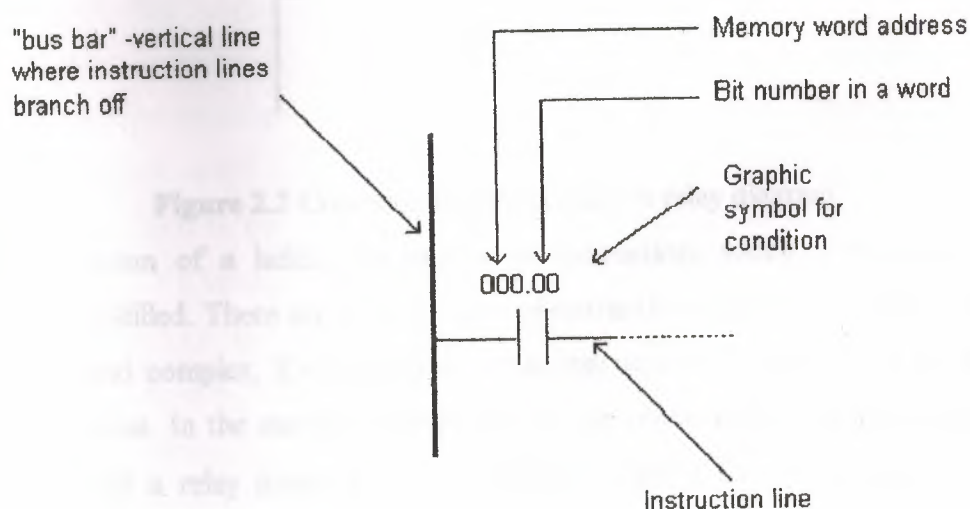


Figure 2.1 Basic elements of a relay diagram

Most instructions require at least one operand, and often more than one. Operand can be some memory location, one memory location bit, or some numeric value -

number. In the example above, operand is bit 0 of memory location IR000. In a case when we wish to proclaim a constant as an operand, designation # is used beneath the numeric writing (for a compiler to know it is a constant and not an address.)

Based on figure 2.1, one should note that a ladder diagram consists of two basic parts: left section also called conditional, and a right section, which has instructions. When a condition is fulfilled, instruction is executed, and that's all!

Figure 2.2 represents an example of a ladder diagram where relay is activated in PLC controller when signal appears at input line 00. Vertical line pairs are called conditions. Each condition in a ladder diagram has a value ON or OFF, depending on a bit status assigned to it. In this case, this bit is also physically present as an input line (screw terminal) to a PLC controller. If a key is attached to a corresponding screw terminal, you can change bit status from a logic one status to a logic zero status, and vice versa. Status of logic one is usually designated as "ON" and status of logic zero as "OFF".

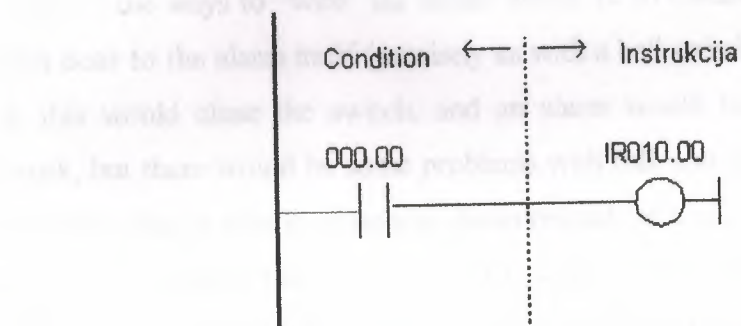


Figure 2.2 Condition and instruction in relay diagram

Right section of a ladder diagram is an instruction, which is executed if left condition is fulfilled. There are several types of instructions that could easily be divided into simple and complex. Example of a simple instruction is activation of some bit in memory location. In the example above, this bit has physical connotation because it is connected with a relay inside a PLC controller. When a CPU activates one of the leading four bits in a word IR010, relay contacts move and connect lines attached to it. In this case, these are the lines connected to a screw terminal marked as 00 and to one of COM screw terminals.

2.2 Normally open and normally closed contacts

Since we frequently meet with concepts "normally open" and "normally closed" in industrial environment, it's important to know them. Both terms apply to words such as contacts, input, output, etc. (all combinations have the same meaning whether we are talking about input, output, contact or something else).

Principle is quite simple, normally open switch won't conduct electricity until it is pressed down, and normally closed switch will conduct electricity until it is pressed. Good examples for both situations are the doorbell and a house alarm.

If a normally closed switch is selected, bell will work continually until someone pushes the switch. By pushing a switch, contacts are opened and pushing a switch interrupts the flow of electricity towards the bell. Of course, system so designed would not in any case suit the owner of the house. A better choice would certainly be a normally open switch. This way bell wouldn't work until someone pushed the switch button and thus informed of his or her presence at the entrance.

Home alarm system is an example of an application of a normally closed switch. Let's suppose that alarm system is intended for surveillance of the front door to the house. One of the ways to "wire" the house would be to install a normally open switch from each door to the alarm itself (precisely as with a bell switch). Then, if the door was opened, this would close the switch, and an alarm would be activated. This system could work, but there would be some problems with this, too. Let's suppose that switch is not working, that a wire is somehow disconnected, or a switch is broken, etc. (there are many ways in which this system could become dysfunctional). The real trouble is that a homeowner would not know that a system was out of order. A burglar could open the door, a switch would not work, and the alarm would not be activated. Obviously, this isn't a good way to set up this system. System should be set up in such a way so a burglar activates the alarm, but also by its own dysfunction or if any of the components stopped working. (A homeowner would certainly want to know if a system was dysfunctional). Having these things in mind, it is far better to use a switch with normally closed contacts, which will detect an unauthorized entrance (opened door interrupts the flow of electricity, and this signal is used to activate a sound signal), or a failure on the system such as a disconnected wire. These considerations are even more important in industrial environment where a failure could cause injury at work. One such example where outputs with normally closed contacts are used is a safety wall with

trimming machines. If the wall doors open, switch affects the output with normally closed contacts and interrupts a supply circuit. This stops the machine and prevents an injury.

Concepts normally open and normally closed can apply to sensors as well. Sensors are used to sense the presence of physical objects, measure some dimension or some amount. For instance, one type of sensors can be used to detect presence of a box on an industry transfer belt. Other types can be used to measure physical dimensions such as heat, etc. Still, most sensors are of a switch type. Their output is in status ON or OFF depending on what the sensor "feels". Let's take for instance a sensor made to feel metal when a metal object passes by the sensor. For this purpose, a sensor with a normally open or a normally closed contact at the output could be used. If it were necessary to inform a PLC each time an object passed by the sensor, a sensor with a normally open output should be selected. Sensor output would set off only if a metal object were placed right before the sensor. A sensor would turn off after the object has passed. PLC could then calculate how many times a normally open contact was set off at the sensor output, and would thus know how many metal objects passed by the sensor.

Concepts normally open and normally closed contact ought to be clarified and explained in detail in the example of a PLC controller input and output. The easiest way to explain them is in the example of a relay in the figure 2.3.

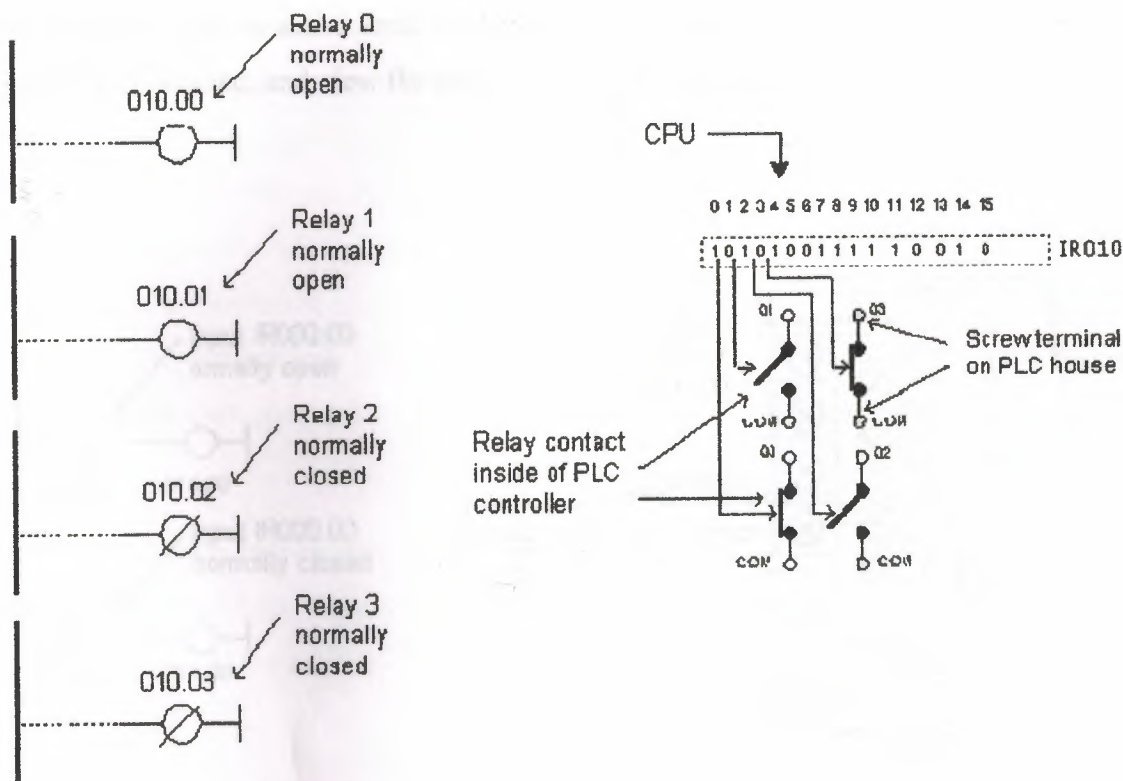


Figure 2.3 Open and closed relays diagram

Normally open contacts would represent relay contacts that would perform a connection upon receipt of a signal. Unlike open contacts, with normally closed contacts signal will interrupt a contact, or turn a relay off. Previous picture shows what this looks like in practice. First two relays are defined as normally open, and the other two as normally closed. All relays react to a signal! First relay (00) has a signal and closes its contacts. Second relay (01) does not have a signal and remains opened. Third relay (02) has a signal and opens its contacts considering it is defined as a closed contact. Fourth relay (03) does not have a signal and remains closed because it is so defined.

Concepts "normally open" and "normally closed" can also refer to input of a PLC controller. Let's use a key as an example of an input to a PLC controller. Input where a key is connected can be defined as input with open or closed contacts. If it is defined as input with normally open contact, pushing a key will set off an instruction found after the condition. In this case it will be an activation of a relay 00.

If input is defined as input with normally closed contact, pushing the key will interrupt instruction found after the condition. In this case, this will cause deactivation

of relay 00 (relay is active until the key is pressed). You can see in the figure 2.4 how keys are connected, and view the relay diagrams in both cases.

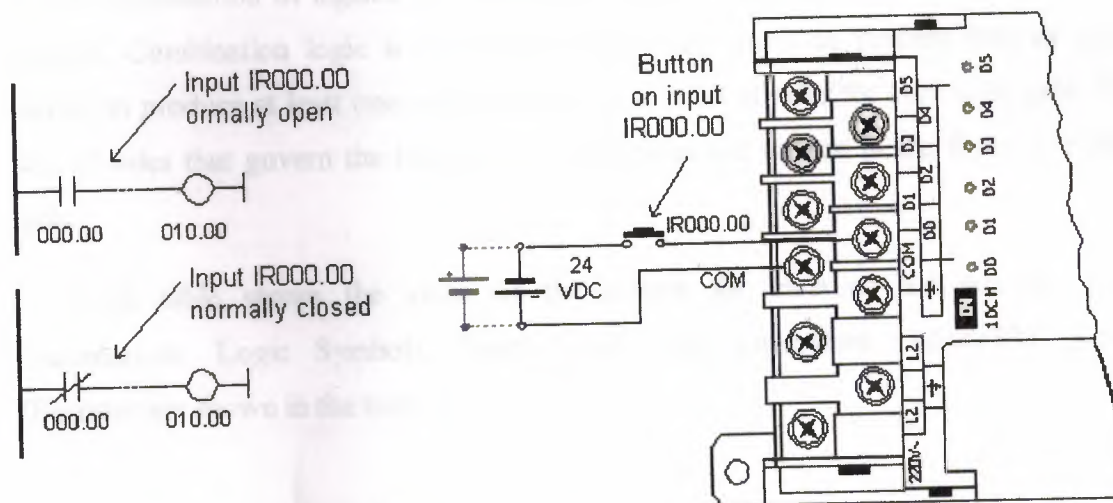


Figure 2.4 Connection of the keys and relays diagram

Normally open/closed conditions differ in a ladder diagram by a diagonal line across a symbol. What determines an execution condition for instruction is a bit status marked beneath each condition on instruction line. Normally open condition is ON if its operand bit has ON status, or its status is OFF if that is the status of its operand bit. Normally closed condition is ON when its operand bit is OFF, or it has OFF status when the status of its operand bit is ON.

When programming with a ladder diagram, logical combination of ON and OFF conditions set before the instruction determines the eventual condition under which the instruction will be, or will not be executed. This condition, which can have only ON or OFF values, is called instruction execution condition. Operand assigned to any instruction in a relay diagram can be any bit from IR, SR, HR, AR, LR or TC sector. This means that conditions in a relay diagram can be determined by a status of I/O bits, or of flags, operational bits, timers/counters, etc.

2.3 Combination Logic




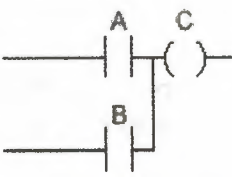
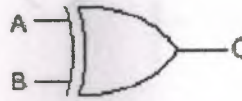
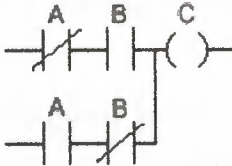
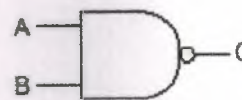
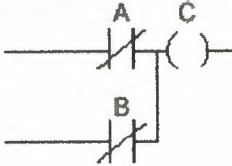
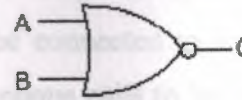
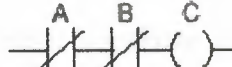
A digital logic system is concerned with just two voltage states. The signal processed by the system is either: high or low; on or off; logic 1 or logic 0. In the early

days of digital systems, circuits were built from discrete components such as resistors and transistors. Today most digital systems are built primarily from integrated circuits. At the heart of all digital systems are small switching circuits called logic gates. There are a number of different logic gates that open and give a high output signal, depending on the combination of signals present at their inputs. Logic gates are decision-making circuits. Combination logic is about combining logic gates to process two or more signals to produce at least one output according to a set of rules for each logic gate. The sets of rules that govern the behavior of logic gates are written in the form of a truth table.

Truth table shows the value of the output for each of the possible input combinations. Logic Symbols, Truth Tables, and Equivalent Ladder/PLC Logic Diagrams are shown in the table 2.1.

 <p>OR Gate</p>		
 <p>AND Gate</p>		
 <p>XOR Gate</p>		

Table 2.1 Logic Symbols, Truth Tables, and Equivalent Ladder/PLC Logic

Logic Diagram	Truth Table	Ladder Diagram															
 <p>AND Gate</p>	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>C</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	C	0	0	0	0	1	0	1	0	0	1	1	1	 <p>AND Equivalent Circuit</p>
A	B	C															
0	0	0															
0	1	0															
1	0	0															
1	1	1															
 <p>OR Gate</p>	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>C</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	C	0	0	0	0	1	1	1	0	1	1	1	1	 <p>OR Equivalent Circuit</p>
A	B	C															
0	0	0															
0	1	1															
1	0	1															
1	1	1															
 <p>Exclusive-OR Gate</p>	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>C</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	C	0	0	0	0	1	1	1	0	1	1	1	0	 <p>Exclusive-OR Equivalent Circuit</p>
A	B	C															
0	0	0															
0	1	1															
1	0	1															
1	1	0															
 <p>NAND Gate</p>	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>C</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	C	0	0	1	0	1	1	1	0	1	1	1	0	 <p>NAND Equivalent Circuit</p>
A	B	C															
0	0	1															
0	1	1															
1	0	1															
1	1	0															
 <p>NOR Gate</p>	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>C</th></tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	C	0	0	1	0	1	0	1	0	0	1	1	0	 <p>NOR Equivalent Circuit</p>
A	B	C															
0	0	1															
0	1	0															
1	0	0															
1	1	0															

2.4 Complex Gate Forms

In total there are 16 different possible types of 2-input logic gates. The simplest are AND and OR, the other gates we will refer to as complex to differentiate. The three popular complex gates that have been discussed before are NAND, NOR and EOR. All of these can be reduced to simpler forms with only ANDs and ORs that are suitable for ladder logic, as shown in Figure 2.5.

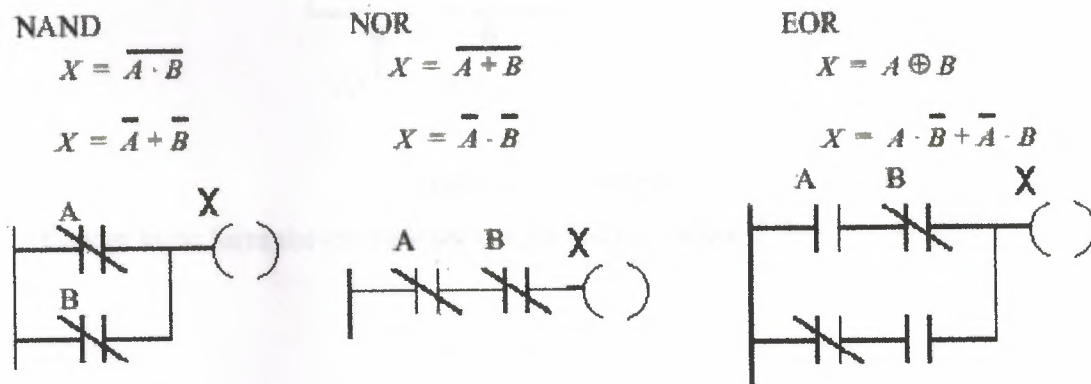


Figure 2.5 Conversion of complex logic functions

2.5 Multiplexes

Multiplexes allow multiple devices to be connected to a single device. These are very popular for telephone systems. A telephone *switch* is used to determine which telephone will be connected to a limited number of lines to other telephone switches. This allows telephone calls to be made to somebody far away without a dedicated wire to the other telephone. In older telephone switchboard operators physically connected wires by plugging them in. In modern computerized telephone switches the same thing is done, but to digital voice signals.

In Figure 2.6 a multiplexer is shown that will take one of four inputs bits D1, D2, D3 or D4 and make it the output X, depending upon the values of the address bits, A1 and A2.

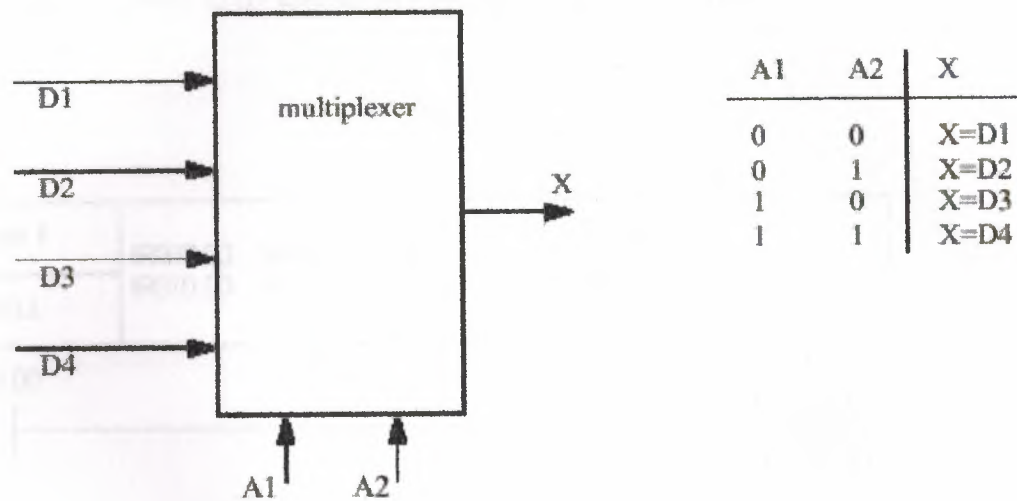


Figure 2.6 multiplexer

Ladder logic form the multiplexer can be seen in Figure 2.7.

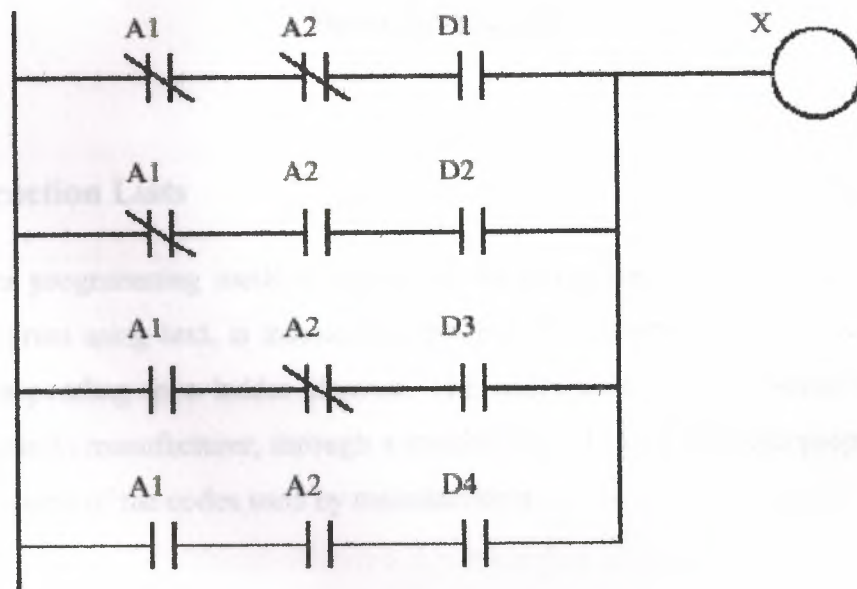


Figure 2.7 A Multiplexer in Ladder Logic

Example in figure 2.8 represents a basic program. Example consists of one input device and one output device linked to the PLC controller output. Key is an input device, and a bell is an output supplied through a relay 00 contact at the PLC controller output. Input 000.00 represents a condition in executing an instruction over 010.00 bit. Pushing the key sets off a 000.00 bit and satisfies a condition for activation of a 010.00

bit, which in turn activates the bell. For correct program function another line of program is needed with END instruction, and this ends the program.

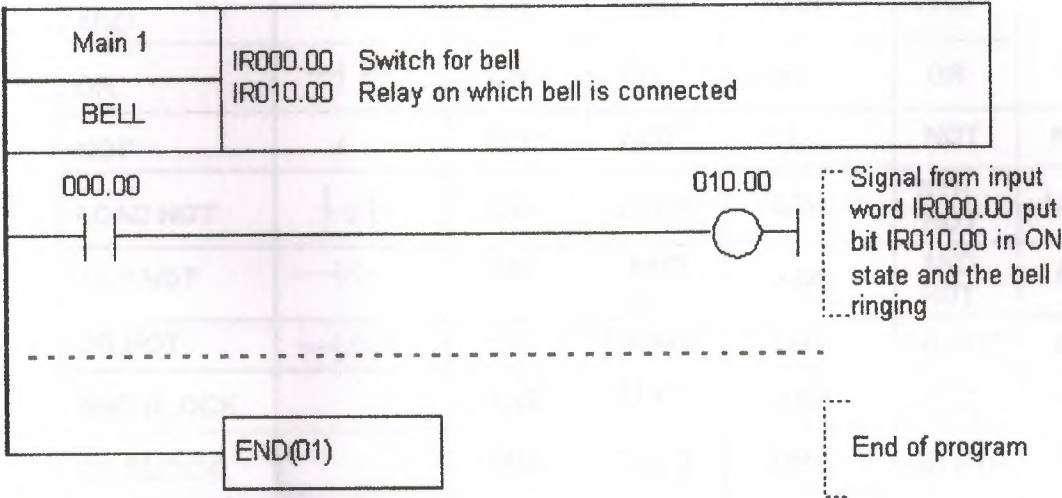









Figure 2.8 Example

2.6 Instruction Lists

Mother programming method, which can be considered to be the entering of the ladder program using text, is instruction lists. For this, mnemonic codes are used, each code corresponding to a ladder element. The codes used differ to some extent from manufacturer to manufacturer, through a standard IEC 1131-3 has been proposed. Table 2.2 shows some of the codes used by manufacturers.

Table 2.2 The codes used by manufacturers.

INSTRUCTION	LADDER SYMBOL	HITACHI	OMRON	MITSUMI.	TEXAS INST.	SIMATIC S7
LOAD		LD	LD	LD	STR	LD
AND		AND	AND	AND	AND	A
OR		OR	OR	OR	OR	O
NOT	/	NOT	NOT	I	NOT	NOT
LOAD NOT		LDI	LD NOT	LDI	STR NOT	LDN
AND NOT		ANI	AND NOT	ANI	AND NOT	AN
OR NOT		ORI	OR NOT	ORI	OR NOT	ON
AND BLOCK		ANB	ANLD	ANB	AND STR	ALD
OR BLOCK		ORB	ORLD	ORB	OR STR	OLD
OUT		OUT	OUT	OUT	OUT	=
END	END	END	END	END	END	MEND

Whenever a network is started, it must use a start a network code. In the Simatic S7, this should be LD to indicate the network is starting with open contacts, or LDN to indicate it is starting with closed contacts. Each network must end with an output. This should be =, and MEND instruction is used to complete the program.

The following shows how individual networks on a ladder are entered using the Simatic S7 for the AND gate, shown in figure 2.9. Step 0 is the start of the network with LD because it is starting with open contacts. Since the address of the input is I0.0, the instruction is LD I0.0. This is followed by another open contact input and so step 1 involves the instruction A with the address of the element, thus step 2 involves the instruction A I0.1. The network terminates with an output and so the instruction = is used with the address of the output Q0.0. Finally, the step 3 involves the instruction MEND.

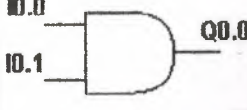
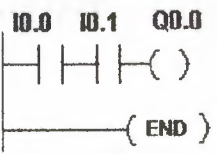
Logic Diagram	Truth Table	Ladder Diagram	Step	Instruction																										
 <p>AND Gate</p>	<table><tr><th>IO.0</th><th>IO.1</th><th>Q0.0</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	IO.0	IO.1	Q0.0	0	0	0	0	1	0	1	0	0	1	1	1	 <p>AND Equivalent Circuit</p>	<table><tr><td>0</td><td>LD</td><td>IO.0</td></tr><tr><td>1</td><td>A</td><td>IO.1</td></tr><tr><td>2</td><td>=</td><td>Q0.0</td></tr><tr><td>3</td><td>MEND</td><td></td></tr></table>	0	LD	IO.0	1	A	IO.1	2	=	Q0.0	3	MEND	
IO.0	IO.1	Q0.0																												
0	0	0																												
0	1	0																												
1	0	0																												
1	1	1																												
0	LD	IO.0																												
1	A	IO.1																												
2	=	Q0.0																												
3	MEND																													

Figure 2.9 AND gate

Consider another example, an OR gate. Figure 2.10 shows the gate with Simatic S7. The instruction for the rung start with an open contact is LD IO.0. The next item is the parallel O set of contact IO.1. Thus the next instruction is O IO.1. The next step is the output, hence = Q0.0. The last step is the end. Therefore, the instruction is MEND.

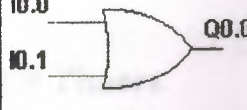
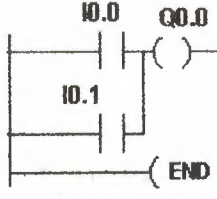
Logic Diagram	Truth Table	Ladder Diagram	Step	Instruction																										
 <p>OR Gate</p>	<table><tr><th>IO.0</th><th>IO.1</th><th>Q0.0</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	IO.0	IO.1	Q0.0	0	0	0	0	1	1	1	0	1	1	1	1	 <p>OR Equivalent Circuit</p>	<table><tr><td>0</td><td>LD</td><td>IO.0</td></tr><tr><td>1</td><td>O</td><td>IO.1</td></tr><tr><td>2</td><td>=</td><td>Q0.0</td></tr><tr><td>3</td><td>MEND</td><td></td></tr></table>	0	LD	IO.0	1	O	IO.1	2	=	Q0.0	3	MEND	
IO.0	IO.1	Q0.0																												
0	0	0																												
0	1	1																												
1	0	1																												
1	1	1																												
0	LD	IO.0																												
1	O	IO.1																												
2	=	Q0.0																												
3	MEND																													

Figure2.10 OR gate

2.7 Branch Code

The Exclusive OR (XOR) gate shown in the figure 2.11 in the Simatic S7 notation has two parallel arms with an AND situation in each arm. In such a situation Simatic S7 uses an OLD instruction to indicate OR together parallel branches. The first instruction is for a normally open contact IO.0. The next instruction is for a series set of normally closed contact IO.1, after reading the first two instructions, the third instruction starts a new line. It is recognized as a new line because it starts with LD, all new lines starting with LD or LDN. But the first line has not been ended by an output. The PLC thus recognizes that a parallel line is involved for the second line and reads together the listed elements until the OLD instruction is reached. The mnemonic OLD indicates to

the PLC that it should OR the results of steps 0 and 1 with that of the new branch with step 2 and 3. The list concludes with the output Q0.0.

Ladder Diagram	Step	Instruction
<p>Exclusive-OR Equivalent Circuit</p>	1	LD I0.0
	2	AN I0.1
	3	LDN I0.0
	4	A I0.1
	5	OLD
	6	= Q0.0
	7	MEND

Figure 2.11 XOR gate

2.8 Timers

In many control tasks there is need to control time. For example, a motor or a pump might need to be controlled to operate for a particular and interval of time, or perhaps be switched on after some time interval. PLCs thus have timers as built-in devices. Timers count fractions of seconds or seconds using the internal CPU clock.

When we look at the different kinds of timers available the fun begins. As always, different types of timers are available with different manufacturers. Here are most of them:

1) On-Delay Timer

These type of timer simply "delays turning on". In other words, after our sensor (input) turns on we wait x-seconds before activating a solenoid valve (output). This is the most common timer. It is often called TON (timer on-delay), TIM (timer) or TMR (timer).

2) Off-Delay Timer

This type of timer is the opposite of the on-delay timer listed above. This timer simply "delays turning off". After our sensor (input) sees a target we turn on a solenoid (output). When the sensor no longer sees the target we hold the solenoid on for x-seconds before turning it off. It is called a TOF (timer off-delay) and is less common than the on-delay type listed above. (i.e. few manufacturers include this type of timer)

3) Retentive or Accumulating Timer

This type of timer needs 2 inputs. One input starts the timing event (i.e. the clock starts ticking) and the other resets it. The on/off delay timers above would be reset if the input sensor wasn't on/off for the complete timer duration. This timer however holds or retains the current elapsed time when the sensor turns off in mid-stream. For example, we want to know how long a sensor is on for during a 1 hour period. If we use one of the above timers they will keep resetting when the sensor turns off/on. This timer however, will give us a total or accumulated time. It is often called an RTO (retentive timer) or TMRA (accumulating timer).

Let's now see how to use them. We typically need to know 2 things:

- 1) What will enable the timer? Typically this is one of the inputs.(a sensor connected to input 0000 for example)
- 2) How long we want to delay before we react. Let's wait 5 seconds before we turn on a solenoid, for example.

When the instructions before the timer symbol are true the timer starts "ticking". When the time elapses the timer will automatically close its contacts. When the program is running on the PLC the program typically displays the elapsed or "accumulated" time for us so we can see the current value. Typically timers can tick from 0 to 9999 or 0 to 65535 times.

Why the weird numbers? Again its because most PLCs have 16-bit timers. We'll get into what this means in a later chapter but for now suffice it to say that 0-9999 is 16-bit BCD (binary coded decimal) and that 0 to 65535 is 16-bit binary. Each tick of the clock is equal to x-seconds.

Typically each manufacturer offers several different ticks. Most manufacturers offer 10 and 100 ms increments (ticks of the clock). An "ms" is a mill-second or 1/1000th of

a second. Several manufacturers also offer 1ms as well as 1 second increments. These different increment timers work the same as above but sometimes they have different names to show their timebase. Some are TMH (high-speed timer), TMS (super high-speed timer) or TMRAF (accumulating fast timer)

Figure 2.12 shows a typical timer instruction symbol we will encounter (depending on which manufacturer we choose) and how to use it. Remember that while they may look different they are all used basically the same way. If we can setup one we can setup any of them.

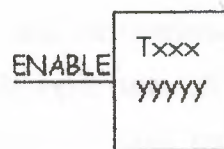


Figure 2.12 Timer

This timer is the on-delay type and is named Txxx. When the enable input is on the timer starts to tick. When it ticks yyyyy (the preset value) times, it will turn on its contacts that we will use later in the program. Remember that the duration of a tick (increment) varies with the vendor and the timebase used. (i.e. a tick might be 1ms or 1 second or...)

In diagram which is shown in figure 2.13 we wait for input 0001 to turn on. When it does, timer T000 (a 100ms-increment timer) starts ticking. It will tick 100 times. Each tick (increment) is 100ms so the timer will be a 10000ms (i.e. 10 second) timer. $100\text{ticks} \times 100\text{ms} = 10,000\text{ms}$. When 10 seconds have elapsed, the T000 contacts close and 500 turns on. When input 0001 turns off(false) the timer T000 will reset back to 0 causing its contacts to turn off(become false) thereby making output 500 turn back off.

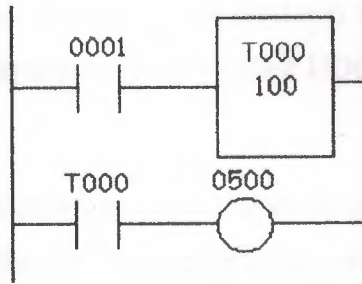


Figure 2.13 Ladder diagram

The timer, which is shown in figure 2.14, is named Txxx. When the enable input is on the timer starts to tick. When it ticks yyyyy (the preset value) times, it will turn on its contacts that we will use later in the program. Remember that the duration of a tick (increment) varies with the vendor and the timebase used. (i.e. a tick might be 1ms or 1 second or...) If however, the enable input turns off before the timer has completed, the current value will be retained. When the input turns back on, the timer will continue from where it left off. The only way to force the timer back to its preset value to start again is to turn on the reset input.

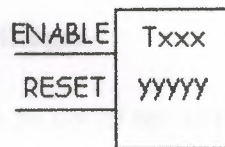


Figure 2.14 Timer

2.9 Counters

A counter is a simple device intended to do one simple thing - count. Using them, however, can sometimes be a challenge because every manufacturer (for whatever reason) seems to use them a different way. Rest assured that the following information will let you simply and easily programs anybody's counters.

What kinds of counters are there? Well, there are up-counters (they only count up 1,2,3...). These are called CTU,(count up) CNT,C, or CTR. There are down counters (they only count down 9,8,7,...). These are typically called CTD (count down) when

they are a separate instruction. There are also up-down counters (they count up and/or down 1,2,3,4,3,2,3,4,5,...) These are typically called UDC(up-down counter) when they are separate instructions.

Many manufacturers have only one or two types of counters but they can be used to count up, down or both. To further confuse the issue, most manufacturers also include a limited number of high-speed counters. These are commonly called HSC (high-speed counter), CTH (Counter High-speed?) or whatever.

Typically a high-speed counter is a "*hardware*" device. The normal counters listed above are typically "*software*" counters. In other words they don't physically exist in the PLC but rather they are simulated in software. Hardware counters do exist in the PLC and they are not dependent on scan time.

A good rule of thumb is simply to always use the normal (software) counters unless the pulses you are counting will arrive faster than 2X the scan time. (i.e. if the scan time is 2ms and pulses will be arriving for counting every 4ms or longer then use a software counter. If they arrive faster than every 4ms (3ms for example) then use the hardware (high-speed) counters. ($2 \times \text{scan time} = 2 \times 2\text{ms} = 4\text{ms}$)

To use them we must know 3 things:

- 1) Where the pulses that we want to count are coming from. Typically this is from one of the inputs. (A sensor connected to input 0000 for example)
- 2) How many pulses we want to count before we react. Let's count 5 widgets before we box them, for example.
- 3) When and how we will reset the counter so it can count again. After we count 5 widgets let's reset the counter, for example.

When the program is running on the PLC the program typically displays the current or "accumulated" value for us so we can see the current count value.

Typically counters can count from 0 to 9999, -32,768 to +32,767 or 0 to 65535. Why the weird numbers? Because of most PLCs have 16-bit counters. We'll get into what this means in a later chapter but for now suffice it to say that 0-9999 is 16-bit BCD (binary coded decimal) and that -32,768 to 32767 and 0 to 65535 is 16-bit binary.

Here are some of the instruction symbols we will encounter (depending on which manufacturer we choose) and how to use them. Remember that while they may look different they are all used basically the same way. If we can setup one we can setup any of them.

In the counter, which is shown in the figure 2.15, we need 2 inputs. One goes before the reset line. When this input turns on the current (accumulated) count value will return to zero. The second input is the address where the pulses we are counting are coming from.

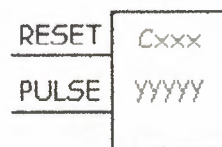


Figure 2.15 Counter

For example, if we are counting how many widgets pass in front of the sensor that is physically connected to input 0001 then we would put normally open contacts with the address 0001 in front of the pulse line. Cxxx is the name of the counter. If we want to call it counter 000 then we would put "C000" here. yyyyy is the number of pulses we want to count before doing something. If we want to count 5 widgets before turning on a physical output to box them we would put 5 here. If we wanted to count 100 widgets then we would put 100 here, etc. When the counter is finished (i.e. we counted yyyyy widgets) it will turn on a separate set of contacts that we also label Cxxx.

Note that the counter accumulated value ONLY changes at the off to on transition of the pulse input.

The ladder diagram in the figure 2.16 is showing how we set up a counter (we'll name it counter 000) to count 100 widgets from input 0001 before turning on output 500. Sensor 0002 resets the counter.

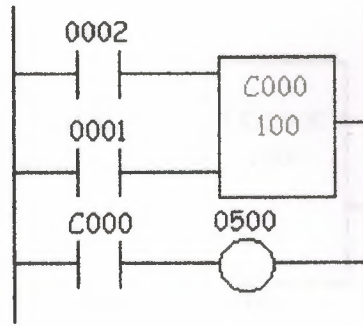


Figure 2.16 Ladder diagram

Figure 2.17 is one symbol we may encounter for an up-down counter. We'll use the same abbreviation as we did for the example above. (i.e. UDCxxx and yyyyy)

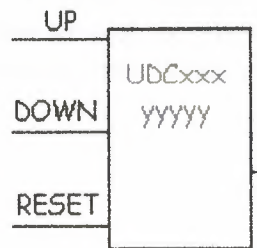


Figure 2.17 Up-down counter

In this up-down counter we need to assign 3 inputs. The reset input has the same function as above. However, instead of having only one input for the pulse counting we now have 2. One is for counting up and the other is for counting down. In this example we will call the counter UDC000 and we will give it a preset value of 1000. (We'll count 1000 total pulses) For inputs we'll use a sensor which will turn on input 0001 when it sees a target and another sensor at input 0003 would also turn on when it sees a target. When input 0001 turns on we count up and when input 0003 turns on we count down. When we reach 1000 pulses we will turn on output 500. Again note that the counter accumulated value only changes at the off to on transition of the pulse input. The ladder diagram is shown in figure 2.18.

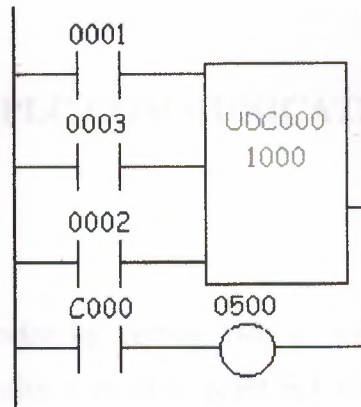


Figure 2.18 Ladder diagram

One important thing to note is that counters and timers can't have the same name (in most PLCs). This is because they typically use the same registers. Well, the counters above might seem difficult to understand but they're actually quite easy once we get used to using them. They certainly are an essential tool. They are also one of the least "standardized" basic instructions that we will see. However, always remember that the theory is the same from manufacturer to manufacturer!

3. PLC COMMUNICATION

3.1 Introduction

Information can be transfer by getting two or more pieces of equipment. The information transfer may involve a point-to-point link such as a computer to PLC or a network of various types of devices. All communication interfaces are either parallel or serial.

3.2 Parallel communication

Parallel communication interfaces use a parallel bus (usually 8-bits wide) to transmit data. They allow data to be transmitted over short distances at high speed. Two common standard parallel communication interfaces are the centronics and IEEE-488. The centronics interface is used for connecting printers. The IEE-488 is mainly used for connecting laboratory instruments to computer.

3.3 Serial Communications

A serial interface transmits and receives data one bit at a time. This means that a data byte has to be separated into component bits for transmission and reassembled back again when received. Serial communication interfaces are used for transmitting data over long distances.

3.3.1 RS232

The most common standard serial communication interface is the RS232 which is also called V24 and EIA. The usual RS232 connector is the 25-pin D connector as shown in the figure 3.1. A minimum cabling configuration will use pins 2, 3 and 7. The connecting lines to pins 2 and 3 normally have to be crossed over that each device

transmits data to a receive pin. A -12V and logic 0 represent logic 1 by +12V. The transmission distance is about 15 m.

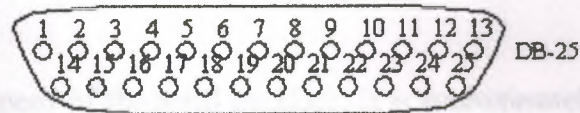


Figure 3.1 RS232 D connector

The main pin assignments for the DB-25 are,

1. 1 - GND (chassis ground)
2. 2 - TXD (transmit data)
3. 3 - RXD (receive data)
4. 4 - RTS (request to send)
5. 5 - CTS (clear to send)
6. 6 - DSR (data set ready)
7. 7 - COM (common)
8. 8 - DCD ()
9. 20 - DTR (data terminal ready)
2. Other pins
 1. 9 - Positive Voltage
 2. 10 - Negative Voltage
 3. 11 - not used
 4. 12 - Secondary Received Line Signal Detector
 5. 13 - Secondary Clear to Send
 6. 14 - Secondary Transmitted Data
 7. 15 - Transmission Signal Element Timing (DCE)
 8. 16 - Secondary Received Data
 9. 17 - Receiver Signal Element Timing (DCE)
 10. 18 - not used
 11. 19 - Secondary Request to Send
 12. 21 - Signal Quality Detector
 13. 22 - Ring Indicator (RI)

14. 23 - Data Signal Rate Selector (DTE/DCE)

15. 24 - Transmit Signal Element Timing (DTE)

16. 25 - Busy

With RS232 the user may set several options within the communication process. Both communicating devices must agree on these options, which are the follows.

1. Baud rate

This is the operation speed of the serial interface. It is approximately the number of bits transmitted or received per second. Standard baud rates are 75, 110, 150, 300, 600, 1200, 2900, 4800, and 9600 baud.

2. Number of bits

This is the length of the data to be communicated. If data bytes are to be transferred then the number of bits is 8. ASCII codes use 7 bits. Teletype equipment uses 5 bits.

3. Parity

Parity is an optional bit added to the data and provides a way of checking whether data has been corrupted. Even parity is when logic 1 is added to the data to make the number of logic is an even number. Odd parity is when logic 1 is added to the data to make the number of logic is an add number. Space parity is when the parity bit is fixed at logic 0. Mark parity is when the parity bit is fixed at logic 1.

4. Stop bits

Bits added to the end of the data are called stop bits. One or two stop bits may be chosen.

5. Duplex

Full duplex requests that a communication device echo backs what it receive. Half duplex tells the communicating device not is echo back what it receives.

6. Flow control

The simplest way of controlling the flow of data between two pieces of equipment is to set the baud rate to that of the slowest link. Alternatively hardware and software methods may be used. The hardware method required that the flow control lines 4, 5, 6, 20 of the two RS232 connectors are wired straight through and crossed 4 to 5, 5 to 4, 6 to 20 and 20 to 6. The software methods called XON-XOFF and ETX-ACK use control characters to regulate the flow of data and so do not require pins 4, 5, 6, and 20 to be connected.

A typical RS232 signal is illustrated in figure 3.2. The data is transmitted at 600 baud. This means that each bit is present for 1.66 ms, which equates to $1/600$. The first bit is the start bit. The next seven bits represent the data. The least significant bit (LSB) appears first and the most significant bit (MSB) last. The following bit is the parity bit. Even parity is used as logic 1 is added to the end of the data to make the number of logic is an even number. The last bit is the stop bit.

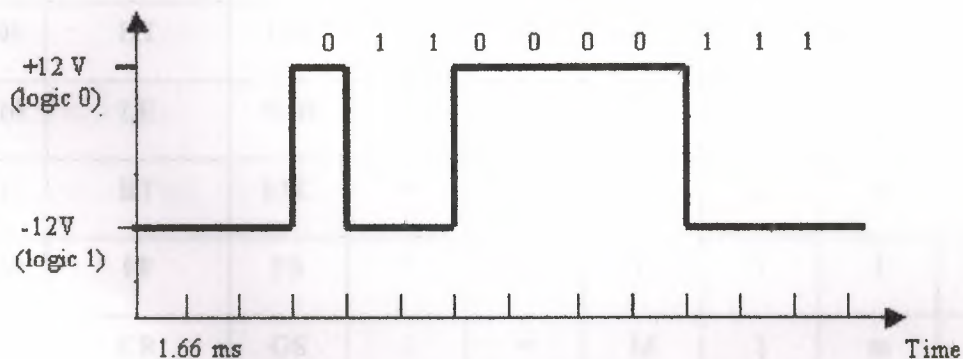


Figure 3.2 RS232 data line at 600 baud

Data transmitted via an RS232 interface is coded into ASCII codes. ASCII stands for American Standard Code for Information Interchange and is a way of coding characters into 7-bits form. ASCII codes are listed in the table 3.1. The table 3.1 shows that the data 1000011 transmitted is represented the ASCII code for the capital letter C. And table3.2 shows the control code.

Table 3.1 ASCII character set (7-bit code)

LS	MS	0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P		P
1	0001	SOH	DC1	!	1	A	Q	a	Q
2	0010	STX	DC2	”	2	B	R	b	R
3	0011	ETX	DC3	#	3	C	S	c	S
4	0100	EOT	DC4	\$	4	D	T	d	T
5	0101	ENQ	NAK	%	5	E	U	e	U
6	0110	ACK	SYN	&	6	F	V	f	V
7	0111	BEL	ETB	‘	7	G	W	g	W
8	1000	BS	CAN	(8	H	X	h	Y
9	1001	HT	EM)	9	I	Y	i	X
A	1010	LE	SUB	*	:	J	Z	j	Z
B	1011	BT	ESC	+	;	K	[k	{
C	1100	FF	FS	,	<	L	\	l	
D	1101	CR	GS	-	=	M]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	US	/	?	O	—	o	DEL

Table 3.2 Control code

NUL	NULL	SI	SHIFT IN
SOH	START OF HEADER	DLE	DALATA LINE SCAPE
STX	START OF TEXT	DC	DEVICE CONTROL
ETX	END OF TEXT	NAK	NEGATIVE ACKNOWLEDGE
EOT	END OF TRANSMISSION	SYN	SYNCHRONOUS IDLE
ENQ	ENQUIRY	ETB	END OF TRASNMISSION BLOCK
ACK	ACKNOLELEDGE	CAN	CANCEL
BEL	BELL	EM	END IF MEDIUM
BS	BACKSPACE	SUM	SUBSTITUTE
HT	HORIZANTAL TAB	ESC	ESCAPE
LF	LINE FEED	FS	FILE SEPARATOR
VT	VERTICAL TAB	GS	GROUP SEPARATOR
FF	FORM FEED	RS	RECORD SEPARATOR
CR	CARRIAGE RETURN	US	UNIT SEPARATOR
SO	SHIFT OUT	SP	SPACE

A computer, which is used as a programming terminal, is normally connected to a PLC via an RS22 link. If programming is done using the computer while the PLC is running and controlling outputs it is terminated on-line programming. If the programming is done using the computer with the PLC not controlling outputs it is termed off-line programming.

3.3.2 RS322, RS422 and RS485

Standards such as RS422 and RS485 are similar to RS232 although voltage level for the states 1 and 0 differs. An RS485 port is set up in similar way to an RS232 (i.e. baud rate, data bits, stop bits and parity must be agreed). Standards such as RS422 have been developed to improve the speed of data transfer.

3.4 Local Area Network (LAN)

A local area network allows a set of PLCs and other devices to be connected together so that they can exchange information. The term local is used because the hardwired link has a limited range. Usually the range is large enough to service a medium-sized factory (500 to 1000 m). Networks, which are used for long-distance communication, are called wide area networks or WANNS.

A network consists of a number of active points (e.g. PLCs), which are called nodes. There are depending upon whether the network uses a series of point links, a central cable with spurs, or links, which make up a ring. Signals transmitted may be any of the following.

1) Base Band

Base band systems simply send a digital signal along the connection cable. Noise immunity can be poor.

2) Signal Channel Modulated

The signal is superimposed on top of a high-frequency sine wave signal called the carrier. The process of altering a wave so that it carries information is called modulation.

3) Broadband

Broadband systems modulate carries signals so that information is carried in separate frequency bands called channels.

The layout of atypical PLC network is illustrated in the figure 3.4. It consists of a central network cable with the PLC connected to it by spurs. The network cable may be

fiber optic, coaxial cable or a preferred because electromagnetic interference and other types of the noise do not effect it generated in a factory. Repeaters are used throughout the network to boost signal levels.

The entire networks use a protocol, which allows nodes (i.e. PLCs) to communicate without crosstalk. Many PLC manufacturers have developed their own network protocol for their own equipment. These are called proprietary networks. Generally the user of a PLC network system does not need to be connected with the technical details of the network interface or protocol.

In the figure 3.3 the computer is linked to the network via a communications converter. This converts an RS232 signal into a network signal.

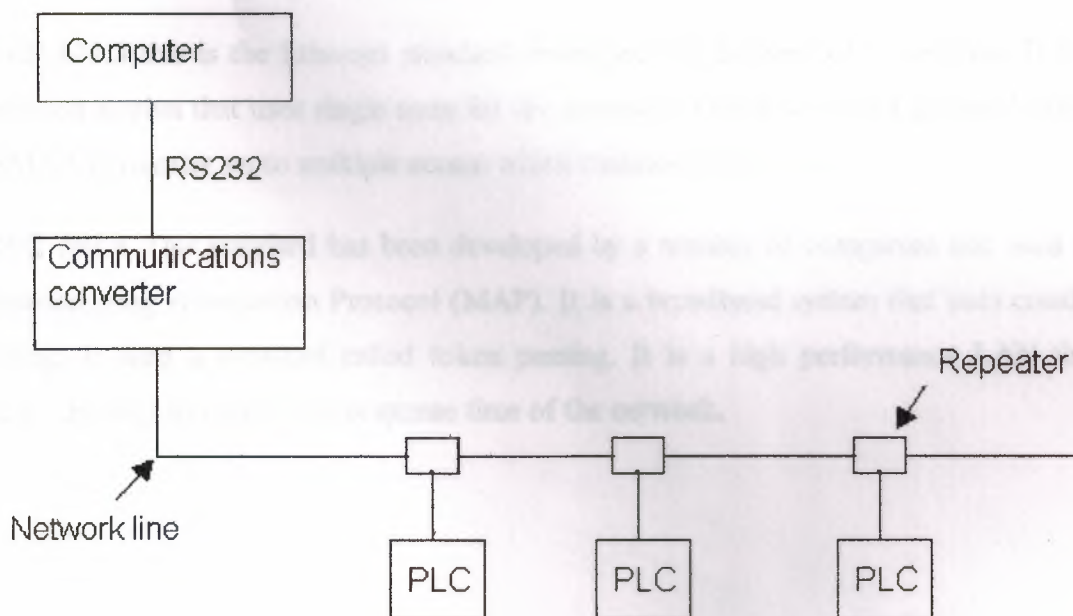


Figure 3.3 PLC network

3.4.1 Response Time of Network

The response time of a network (also called the access time) is the time taken for two nodes to communicate. It effectively gives a measure of how fast data can be transferred from one PLC to another. The response time increases as the number of

active nodes or PLCs on the network increases. Typically the response time is about 10 ms.

It is essential to know a precise value for the response time if two or more PLCs are to work in unison on a time-critical control application. If the network operates too slowly the control action will fail. Emergency stop signals should never be sent on a network but should be hardwired.

3.4.2 Network Standards

Most PLC network uses a data protocol system developed by the PLC manufacture. This means that two PLCs from different manufactures cannot be networked together. Various standards have been suggested which, if adopted by different manufacture, would overcome this problem. IEEE (Institute of Electrical and Electronic Engineers) now accepts two standards. These are as follows.

IEEE 802.3 this is the Ethernet standard developed by the Xerox Corporation. It is a baseband system that uses single coax for the connection cable. It uses a protocol called CSMA/CD (carrier sense multiple access with collision detection).

IEEE 802.4 This standard has been developed by a number of companies and used as Manufacturing Automation Protocol (MAP). It is a broadband system that uses coaxial cabling. It uses a protocol called token passing. It is a high performance LAN that allows the user to predict the response time of the network.

4. Automation of product packaging

4.1. Introduction

Product packaging is one of the most frequent cases for automation in industry. It can be encountered with small machines (ex. packaging grain like food products) and large systems such as machines for packaging medications. The project we are showing in figure 4.1 solves the classic packaging problem with few elements of automation. Small number of needed inputs and outputs provides for the use of PLC controller, which represents simple and economical solution.

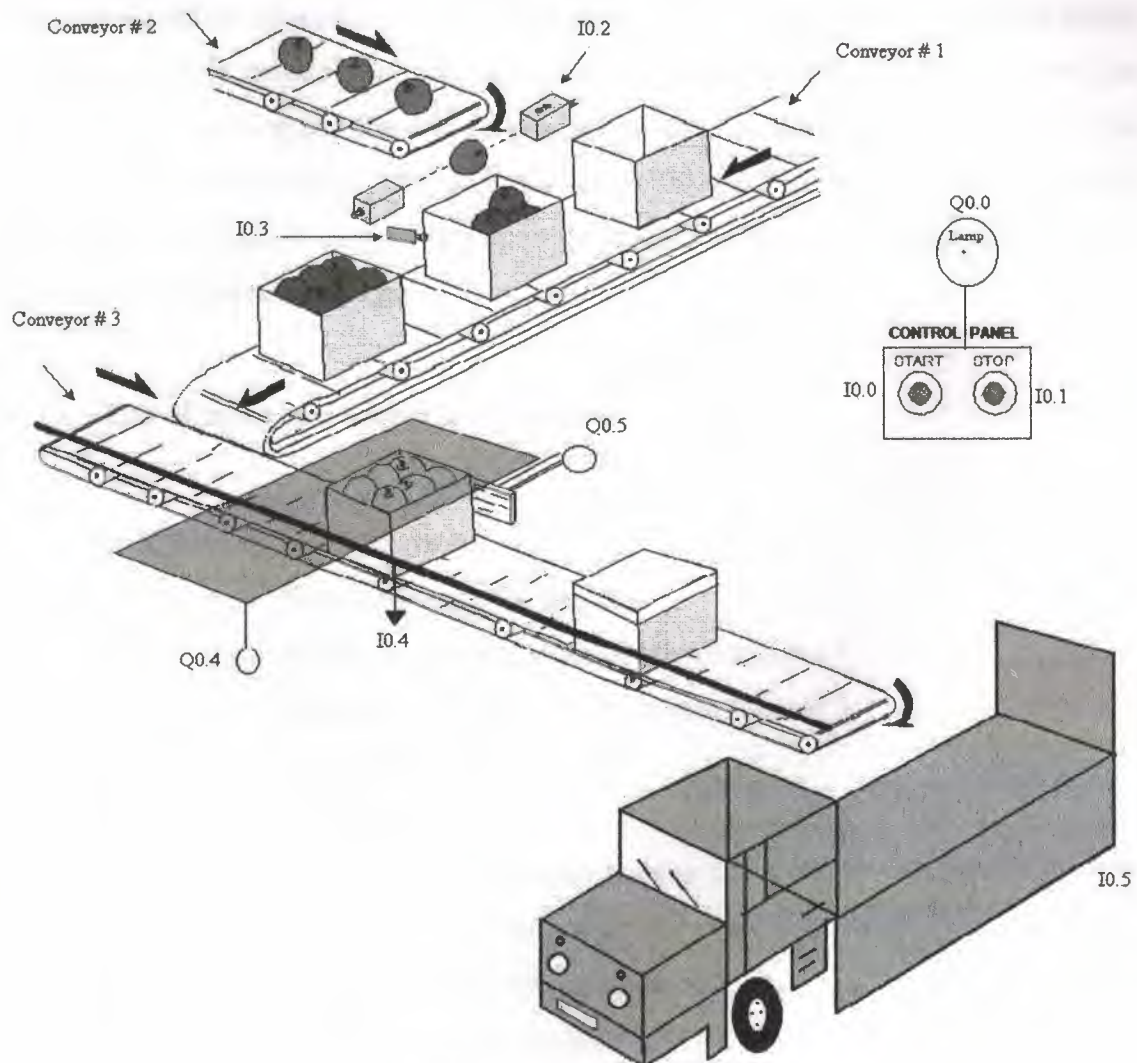


Figure 4.1 The project

4.2. Working process

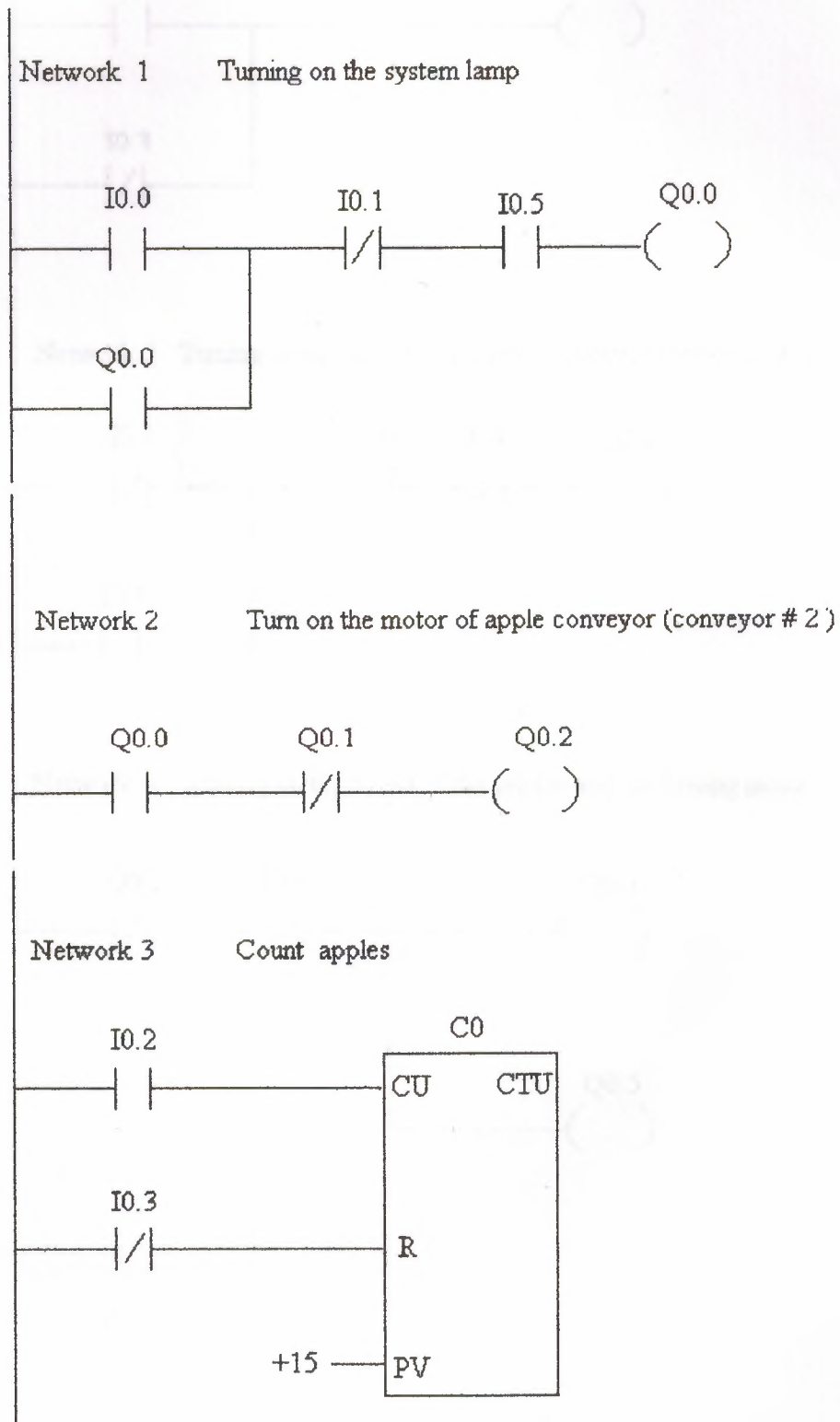
By pushing start key we activate the lamp to show that the system is working and by pushing stop key we deactivate the lamp which means that the system does not working. When started, motor of a conveyor for boxes is activated. The conveyor takes a box up to the limit switch, and motor stops. Condition for starting a conveyor with apples is actually a limit switch for a box. When a box is detected, a conveyor with apples starts moving. Presence of the box allows counter to count 10 apples through a sensor used for apples and to generate counter C0 which is a condition for new activation of a conveyor with boxes. When the conveyor with boxes has been activated, limit switch resets counter which is again ready to count 10 apples.

After filling the box, the box passes on the other conveyor (Q0.3). When the sensor (I0.4) is activated, the motor of a conveyor (Q0.3) stops and at the same time the motor of closing (Q0.4) starts closing the box and the motor of printer (Q0.5) starts print the name of the company and the data of product on the box. Moreover, the motor of the printer activates the timer (T 37) that after (200x100 ms) T37 will reactivate the motor of the conveyor (Q0.3) to take a box up to the track. The operations repeat until stop key is pressed, or if there is no track.

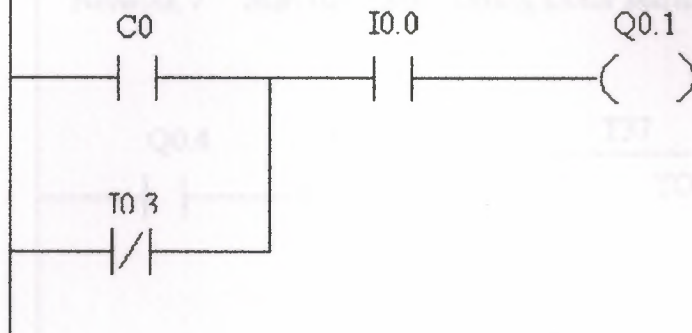
- **List of input and output assignments**

I0.0	Start the system
I0.1	Stop the system
I0.2	Sensor to count the apples
I0.3	Sensor for the box in the conveyor number 1
I0.4	Sensor for the box in the conveyor number 3
I0.5	Sensor for the truck
Q0.0	The lamp
Q0.1	Motor for box conveyor number 1
Q0.2	Motor for apple conveyor number 2
Q0.3	Motor for box conveyor number 3
Q0.4	Motor for closing the box
Q0.5	Motor for the printer

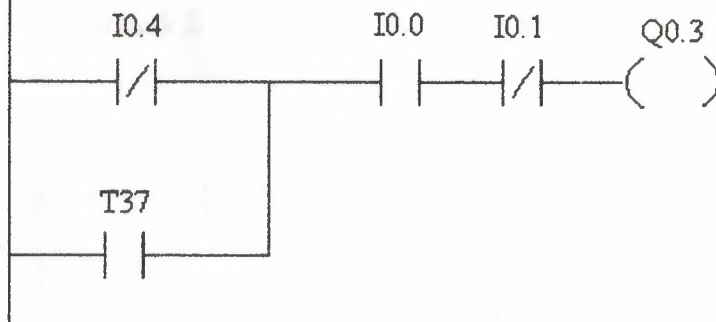
4.3. Ladder diagram of the system



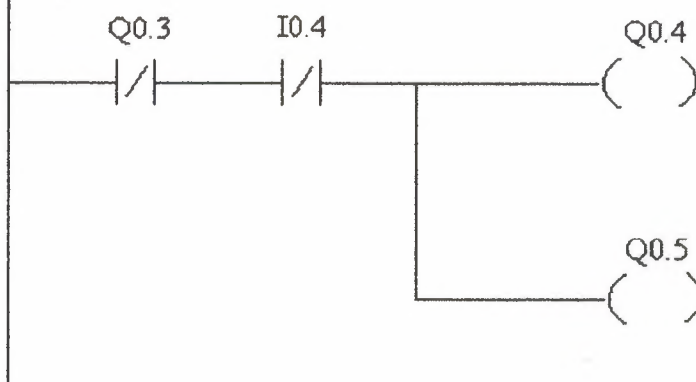
Network 4 Turning on the motor of the boxes conveyor (conveyor # 1)



Network 5 Turning on the motor of the bxs conveyor (conveyor # 3)



Network 6 Turning on the motor of the printer and the closing motor



4.4. Statement list of the system

NETWORK 1 : Turning on the system lights

L3: I0.0

Q: Q0.0

AN: I0.1

A: I0.2

T: T0.0

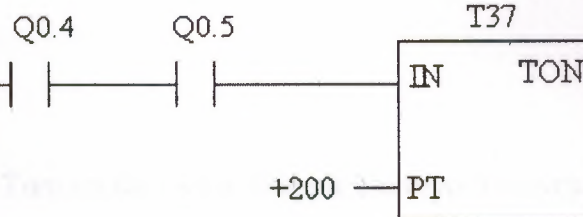
NETWORK 2 : Turning on the motor of the printer

LD: Q0.0

AN: Q0.1

Q0.2

Network 7 Start timer when closing motor and motor of printer are activated



Network 8 End the main program

NETWORK 9 : Counting the number of times the printer is used

LD: I0.2

LDN: I0.3

CTU: C0, +15

(END)

NETWORK 4 : Turning on the motor of the printer when the printer is not in use

LD: I0.0

AN: I0.1

Q: Q0.0

Q0.1

NETWORK 5 : Turning on the motor of the printer when the printer is not in use

LD: I0.0

AN: I0.1

Q: Q0.0

Q0.1

Q0.2

4.4. Statement list of the system

NETWORK 1 // Turning on the system light.

```
LD      I0.0
O        Q0.0
AN      I0.1
A        I0.5
=        Q0.0
```

NETWORK 2 // Turn on the motor of apple conveyor (conveyor # 2).

```
LD      Q0.0
AN      Q0.1
=        Q0.2
```

NETWORK 3 // Count apple.

```
LD      I0.2
LDN     I0.3
CTU     C0, +15
```

NETWORK 4 // Turning on the motor of the boxes conveyor (conveyor # 1).

```
LD      C0
ON      I0.3
A        I0.0
=        Q0.1
```

NETWORK 5 // Turning on the motor of the boxes conveyor (conveyor # 3).

```
LDN     I0.4
O        T37
A        I0.0
AN      I0.1
=        Q0.3
```



NETWORK 6 // Turning on the motor of printer and closing motor.

LDN Q0.3

AN I0.4

= Q0.4

= Q0.5

NETWORK 7 // Start timer when closing motor and motor of the printer are activated.

LD Q0.4

A Q0.5

TON T37, +200

NETWORK 8 // End the main program.

MEND

Conclusion

PLCs can seem a little daunting at first, but there's no need to panic. Just remember that all PLCs follow the basic rules of operation we've just discussed. All PLCs have a CPU and an input/output system. They also all use a control program, instructions, and addressing to make the equipment in the control system do what it's supposed to do.

And no matter how many bells and whistles you add to it, every PLC does the same three things: (1) examines its input devices, (2) executes its control program, and (3) updates its output devices accordingly. So in reality, understanding PLCs is as simple as 1-2-3!

[5] <http://www.rockwell.com>

REFERENCES

- [1] Siemens, Simatic S7-200 Programmable Controller System, 1977.
- [2] Mr. Özgür Cemal ÖZERDEM, Programmable Logic Controllers and Programming. Near East University. Lefkoşa 2001.
- [3] Alan J. Crispin, Programmable Logic Controller and their Engineering Application, Mc Graw-Hill Inc., New York NY, 1996.
- [4] <http://www.sea.siemens.com/controls/product/s7200/CNs7200.htm>
- [5] <http://www.plc.net>.