NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

Department of Electrical and Electronic Engineering

PROGRAMMABLE LOGIC CONTROLLER (PLC)

Graduation Project

Student : RAMAZAN DEMIRSOY (20000431)

Supervisior : Özgür C. Özerdem

Lefkoşa-2005

ACKNOWLEDGMENTS

First I want to thank Mr. Ozgur Cemal Ozerdem to be my advisor.I learned a lot of things with his helps.When I have any problem he always help me.I asked many questions about this project and other courses and he always answered my question .I believe that I will be succesfull in the future with his advises.

I also want to thank my friends in NEU.Serkan,Halil,,Serhat and other friends.We always have been together for 4 years and will be continious to be together.

Finally I want to thank my familly,especcially my parents and older sister and brother for their endless support.And also wants to thank my maternal familly.

ABSTRACT

The main objective of this project is learning the PLC structure and working of PLC programming.

Mechanical parts of this project were prepared before by one student who graduated from Near East University.But in mechanical parts there were untidy and broken devices.These all untidy and broken devices required the modification.

First of all for modification broken devices repaired and put on the box.And also untidy cabels were placed.In this way I provided the tidy.In addition I put on the inputs and outputs connector to the box to connection the PLC easly and practically.

And finally after modification PLC program were prepared which is the main objective of this project.

TABLE OF CONTENTS

INTRODUCTION

With developing technology day by day control systems are developing as well.PLCs are the good example for these developed control system.Usually PLCs are used in the industry rather than other control systems.

Before PLCs relay systems were used.In some industries relay system is still used.However,day by day this systems is leaved.Relay systems and other control systems require the supplement and develop in time.In addition they causes the new expenses.It's possible to make program easly in PLC with minimum expences,effort and time.These features are the most important differences between the PLC and relay sytem and other control systems.In addition,old programs can be saved in the PLC and finding faulty controlling, maintanence can make easly and speedly.

Input output number ,memory capacity,type and number of timer and counter,working speed are very important to choose PLC. PLCs are can be found compact and modular form.In compact PLC all parts placed in a box including the power supply,I/O units,memory,CPU etc.But modular PLCs are manifactured as separete card.The PLC can be mounted on a rock depending upon the desiret operation.

Nowadays there are many brands of PLc manifactured.these are Siemens,Mitsubishi,Omron,Texas etc. Siemens is used commonly in Turkey industry and also we used Siemens S7 200 in our project.

CHAPTER 1

1.1. PLC HISTORY

In the late 1960's PLCs were first introduced. The primary reason for designing such a device was eliminating the large cost involved in replacing the complicated relay based machine control systems. Bedford Associates (Bedford, MA) proposed something called a Modular Digital Controller (MODICON) to a major US car manufacturer. Other companies at the time proposed computer based schemes, one of which was based upon the PDP-8. The MODICON 084 brought the world's first PLC into commercial production.

When production requirements changed so did the control system. This becomes very expensive when the change is frequent. Since relays are mechanical devices they also have a limited lifetime which required strict adhesion to maintenance schedules. Troubleshooting was also quite tedious when so many relays are involved. Now picture a machine control panel that included many, possibly hundreds or thousands, of individual relays. The size could be mind boggling. How about the complicated initial wiring of so many individual devices! These relays would be individually wired together in a manner that would yield the desired outcome.

These "new controllers" also had to be easily programmed by maintenance and plant engineers. The lifetime had to be long and programming changes easily performed. They also had to survive the harsh industrial environment. The answers were to use a programming technique most people were already familiar with and replace mechanical parts with solid-state ones.

In the mid70's the dominant PLC technologies were sequencer state-machines and the bit-slice based CPU. The AMD 2901 and 2903 were quite popular in Modicon and A-B PLCs. Conventional microprocessors lacked the power to quickly solve PLC logic in all but the smallest PLCs. As conventional microprocessors evolved, larger and larger PLCs were being based upon them. However, even today some are still based upon the 2903. (ref A-B's PLC-3) Modicon has yet to build a faster PLC than their 984A/B/X which was based upon the 2901.

Communications abilities began to appear in approximately 1973. The first such system was Modicon's Modbus. The PLC could now talk to other PLCs and they could be far away from the actual machine they were controlling. They could also now be used to

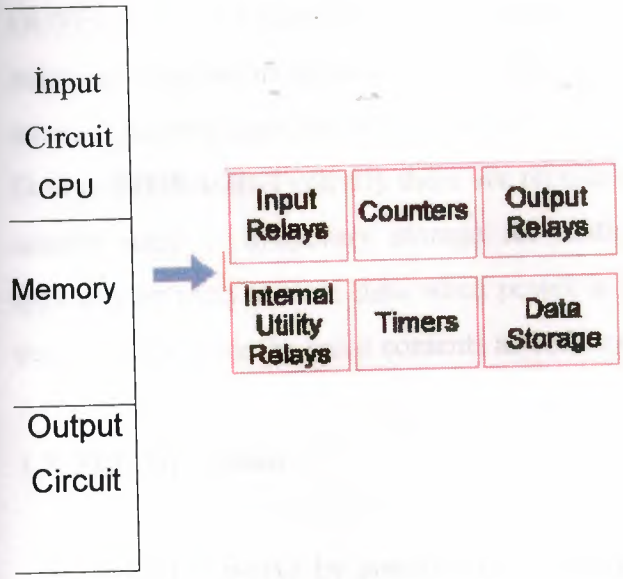
send and receive varying voltages to allow them to enter the analog world. Unfortunately, the lack of standardization coupled with continually changing technology has made PLC communications a nightmare of incompatible protocols and physical networks.

The 80's saw an attempt to standardize communications with General Motor's manufacturing automation protocol(MAP). It was also a time for reducing the size of the PLC and making them software programmable through symbolic programming on personal computers instead of dedicated programming terminals or handheld programmers. Today the world's smallest PLC is about the size of a single control relay!

The 90's have seen a gradual reduction in the introduction of new protocols, and the modernization of the physical layers of some of the more popular protocols that survived the 1980's. The latest standard (IEC 1131-3) has tried to merge PLC programming languages under one international standard. We now have PLCs that are programmable in function block diagrams, instruction lists, C and structured text all at the same time! PC's are also being used to replace PLCs in some applications. The original company who commissioned the MODICON 084 has actually switched to a PC based control system.

1.2. PLC Inside

The PLC mainly consists of a CPU, memory areas, and appropriate circuits to receive input/output data. We can actually consider the PLC to be a box full of hundreds or thousands of separate relays, counters, timers and data storage locations. Do these counters, timers, etc. really exist? No, they don't "physically" exist but rather they are simulated and can be considered software counters, timers, etc. These internal relays are simulated through bit locations in registers. (more on that later)



What does each part do?

INPUT RELAYS-(contacts)These are connected to the outside world. They physically exist and receive signals from switches, sensors, etc. Typically they are not relays but rather they are transistors.

INTERNAL UTILITY RELAYS-(contacts) These do not receive signals from the outside world nor do they physically exist. They are simulated relays and are what enables a PLC to eliminate external relays. There are also some special relays that are dedicated to performing only one task. Some are always on while some are always off. Some are on only once during power-on and are typically used for initializing data that was stored.

COUNTERS-These again do not physically exist. They are simulated counters and they can be programmed to count pulses. Typically these counters can count up, down or both up and down. Since they are simulated they are limited in their counting speed. Some manufacturers also include high-speed counters that are hardware based. We can think of these as physically existing. Most times these counters can count up, down or up and down.

TIMERS-These also do not physically exist. They come in many varieties and increments. The most common type is an on-delay type. Others include off-delay and both retentive and nonretentive types. Increments vary from 1 ms through 1 s.

OUTPUT RELAYS-(coils)These are connected to the outside world. They physically exist and send on/off signals to solenoids, lights, etc. They can be transistors, relays, or triacs depending upon the model chosen.

DATA STORAGE-Typically there are registers assigned to simply store data. They are usually used as temporary storage for math or data manipulation. They can also typically be used to store data when power is removed from the PLC. Upon power-up they will still have the same contents as before power was removed.

1.3. PLC Operation

A PLC works by continually scanning a program. We can think of this scan cycle as consisting of 3 important steps. 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.

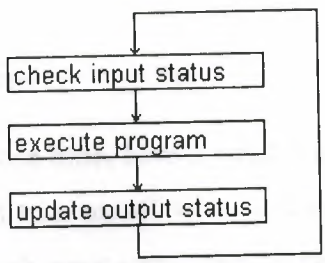


Figure-1.2.

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 ofthe 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.

1.4. Response Time

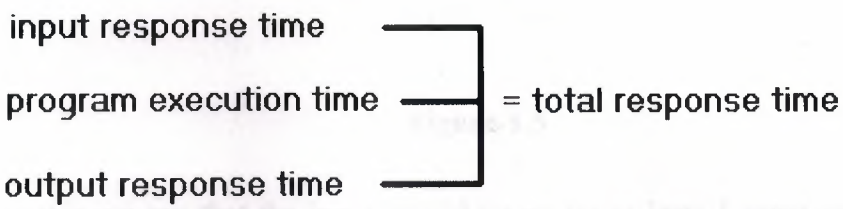


figure-1.3

1.4.1 Response Time Concerns

Now that we know about response time, here's what it really means to the application. The PLC can only see an input turn on/off when it's looking. In other words, it only looks at its inputs during the check input status part of the scan.

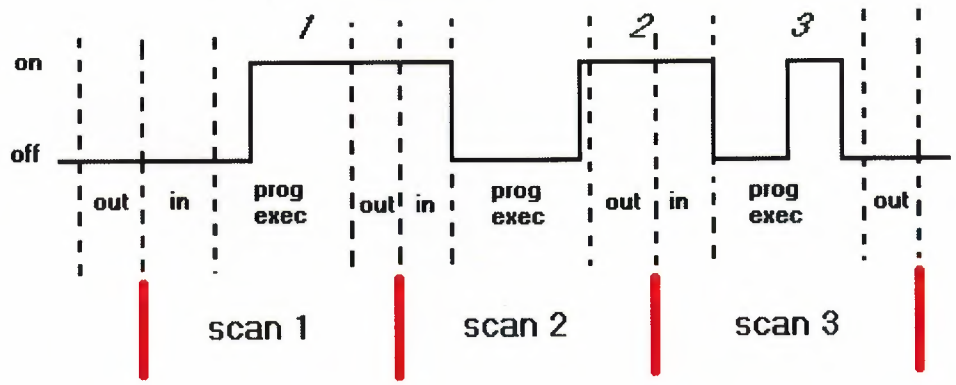


Figure.1.4

In the diagram, input 1 is not seen until scan 2. This is because when input 1 turned on, scan 1 had already finished looking at the inputs.

Input 2 is not seen until scan 3. This is also because when the input turned on scan 2 had

already finished looking at the inputs.

Input 3 is never seen. This is because when scan 3 was looking at the inputs, signal 3 was not on yet. It turns off before scan 4 looks at the inputs. Therefore signal 3 is never seen by the PLC.

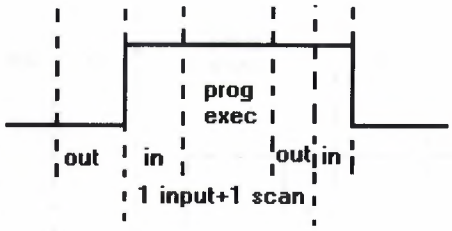


Figure-1.5

To avoid this we say that the input should be on for at least 1 input delay time + one scan time.

But what if it was not possible for the input to be on this long? Then the PLC doesn't see the input turn on. Therefore it becomes a paper weight! Not true... of course there must be a way to get around this. Actually there are 2 ways.

Pulse stretch function. This function extends the length of the input signal until the PLC looks at the inputs during the next scan. (i.e. it stretches the duration of the pulse.)

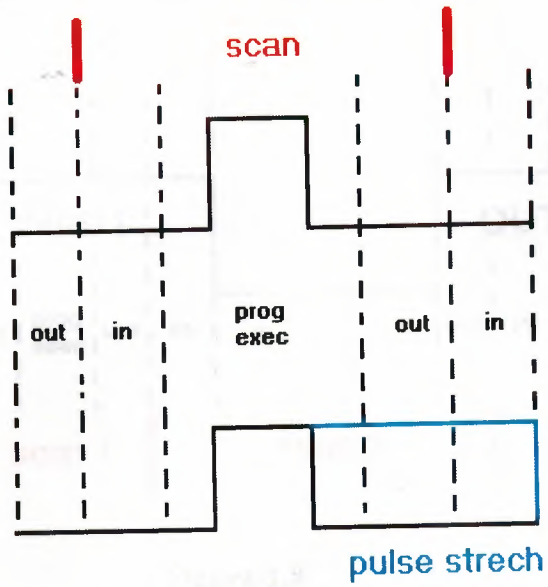


Figure-1.6

1.4.2 **Interrupt function.**

This function interrupts the scan to process a special routine that you have written. İ.e. As soon as the input turns on, regardless of where the scan currently is, the PLC immediately stops what its doing and executes an interrupt routine. (A routine can be thought of as a mini program outside of the main program.) After its done executing the interrupt routine, it goes back to the point it left off at and continues on with the normal scan process.

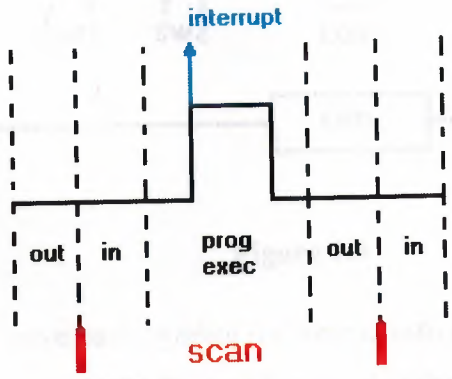


Figure 1.7

Now let's consider the longest time for an output to actually turn on. Let's assume that when a switch turns on we need to turn on a load connected to the PLC output. The diagram below shows the longest delay (worst case because the input is not seen until scan 2) for the output to turn on after the input has turned on.

The maximum delay is thus 2 scan cycles - 1 input delay time.

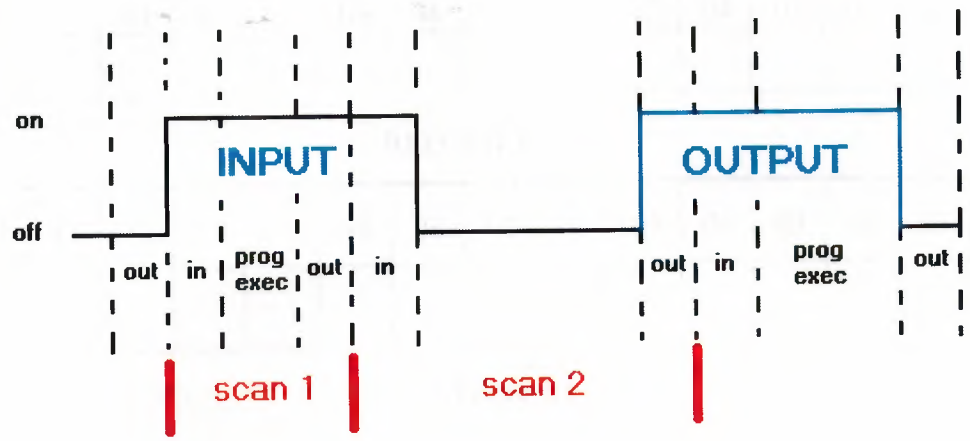


Figure-1.8

1.4.3 PLC Registers

We'll now take the previous example and change switch 2 (SW2) to a normally closed symbol (load bar instruction). SW1 will be physically OFF and SW2 will be physically ON initially. The ladder diagram now looks like

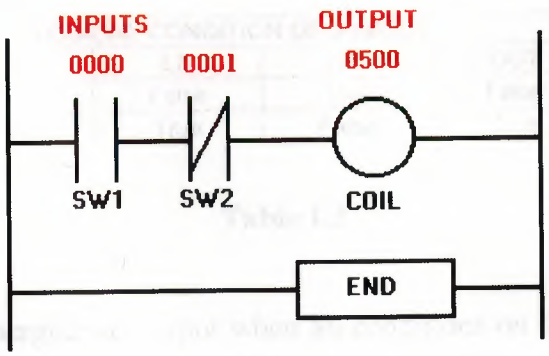


Figure 1.9

Notice also that we now gave each symbol (or instruction) an address. This address sets aside a certain storage area in the PLCs data files so that the status of the instruction (i.e. true/false) can be stored. Many PLCs use 16 slot or bit storage locations. In the example above we are using two different storage locations or registers.

REGISTER 00															
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
														1	0

REGISTER 05															
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
															0

Table 1.1

In the tables above we can see that in register 00, bit 00 (i.e. input 0000) was a logic 0 and bit 01 (i.e. input 0001) was a logic 1. Register 05 shows that bit 00 (i.e. output 0500) was a logic O. The logic 0 or 1 indicates whether an instruction is False or True. *Although most of the items in the register tables above are empty, they should each contain a O. They were left blank to emphasize the locations we were concerned with.

LOGICAL CONDITION OF SYMBOL			
LOGIC BITS	LD	LDB	OUT
Logic 0	False	True	False
Logic 1	True	False	True

Table 1.2

The PLC will only energize an output when all conditions on the rung are TRUE. So, looking at the table above, we see that in the previous example SW1 has to be logic 1 and SW2 must be logic O. Then and ONLY then will the coil be true (i.e. energized). If any of the instructions on the rung before the output (coil) are false then the output (coil) will be false (not energized).

Let's now look at a truth table of our previous program to further illustrate this important point. Our truth table will show ALL possible combinations of the status of the two inputs.

Inputs		Outputs	Register Logic Bits		
SW1 (LD)	SW2(LDB)	COIL(OUT)	SW1 (LD)	SW2(LDB)	COIL(OUT)
False	True	False	0	0	0
False	False	False	0	1	0
True	True	True	1	0	1
True	False	False	1	1	0

Table 1.3

Notice from the chart that as the inputs change their states over time, so will the output. The output is only true (energized) when all preceding instructions on the rung are true.

1.5 Level Application Example

Now that we've seen how registers work, let's process a program like PLCs do to enhance our understanding of how the program gets *scanned.*

Let's consider the following application:
We are controlling lubricating oil being dispensed from a tank. This is possible by using two sensors. We put one near the bottom and one near the top, as shown in the picture below

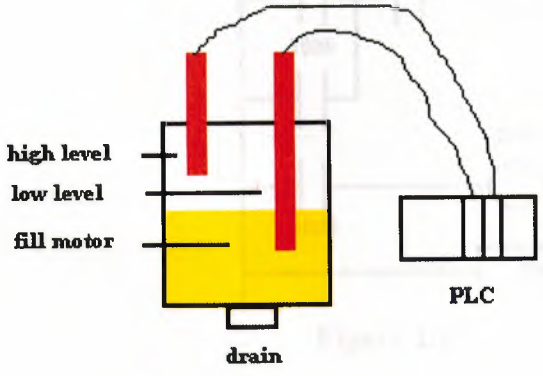


Figure 1.10 Dispensing oil from a tank

Here, we want the fill motor to pump lubricating oil into the tank until the high level sensor turns on. At that point we want to turn off the motor until the level falls below the low level sensor. Then we should turn on the fill motor and repeat the process.

Here we have a need for 3 I/0 (i.e. Inputs/Outputs). 2 are inputs (the sensors) and 1 is an

output (the fill motor). Both of our inputs will be NC (normally closed) fiber-optic level sensors. When they are NOT immersed in liquid they will be ON. When they are immersed in liquid they will be OFF.

We will give each input and output device an address. This lets the PLC know where they are physically connected. The addresses are shown in the following tables:

Inputs	Address	Output	Address	Internal Utility Relay
Low	0000	Motor	0500	1000
High	0001			

Table 1.4

Below is what the ladder diagram will actually look like. Notice that we are using an internal utility relay in this example. You can use the contacts of these relays as many times as required. Here they are used twice to simulate a relay with 2 sets of contacts. Remember, these relays DO NOT physically exist in the PLC but rather they are bits in a register that you can use to SIMULATE a relay.

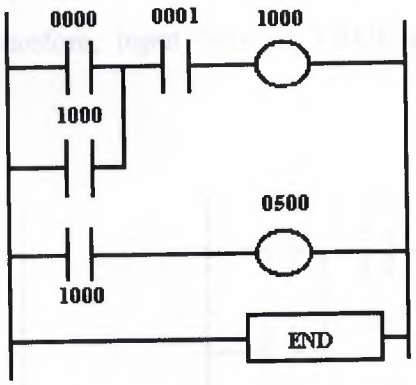


Figure 1.11

We should always remember that the most common reason for using PLCs in our applications is for replacing real-world relays. The internal utility relays make this action possible. It's impossible to indicate how many internal relays are included with each brand of PLC. Some include 100's while other include 1000's while still others include 10's of 1000's! Typically, PLC size (not physical size but rather I/O size) is the deciding factor. If we are using a micro-PLC with a few I/O we don't need many internal relays. If however, we are using a large PLC with 100's or 1000's of 110 we'll

certainly need many more internal relays.

If ever there is a question as to whether or not the manufacturer supplies enough internal relays, consult their specification sheets. In all but the largest of large applications, the supplied amount should be MORE than enough.

1.5.1 The Program Scan

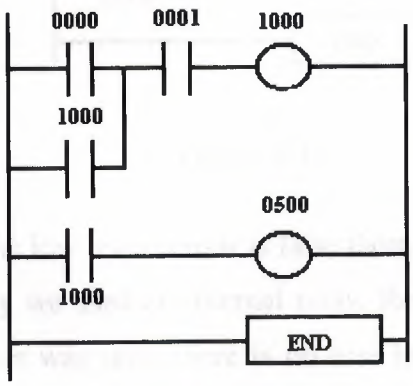


Figure 1.12

Let's watch what happens in this program scan

Initially the tank is empty. Therefore, input 0000 is TRUE and input 0001 is also TRUE.

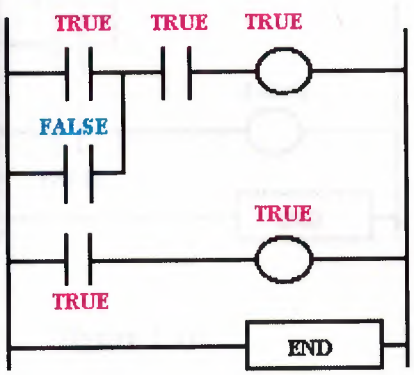


Figure 1.13

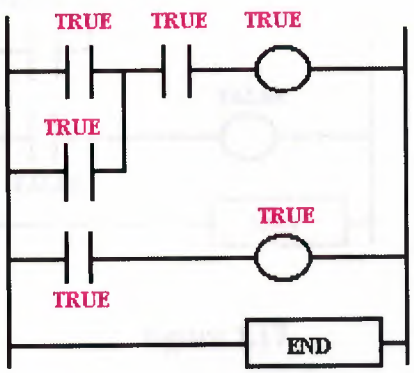


Figure 1.14

Gradually the tank fills because 500(fill motor) is on.

After 100 scans the oil level rises above the low level sensor and it becomes open. (i.e. FALSE)

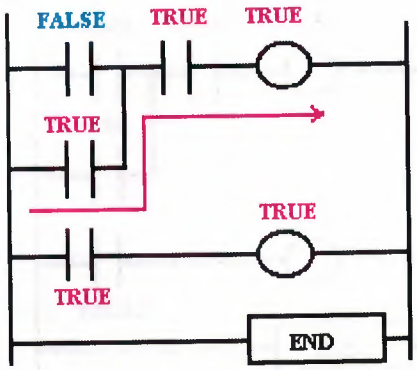


Figure 1.15

Notice that even when the low level sensor is false there is still a path of true logic from left to right. This is why we used an internal relay. Relay 1000 is latching the output (500) on. It will stay this way until there is no true logic path from left to right.(i.e. when 0001 becomes false)

After 1000 scans the oil level rises above the high level sensor at it also becomes open (i.e. false)

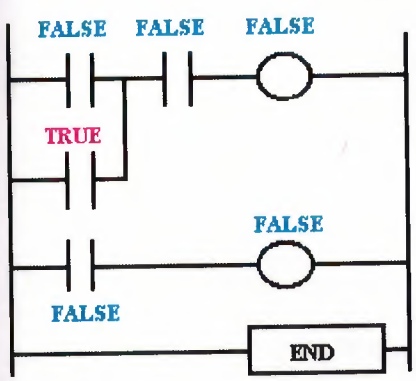


figure 1.16

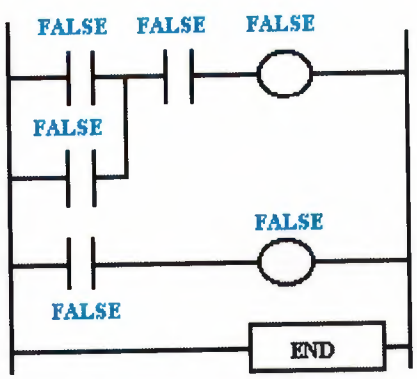


figure 1.17

Since there is no more true logic path, output 500 is no longer energized (true) and therefore the motor turns off.

After 1050 scans the oil level falls below the high level sensor and it will become true again.

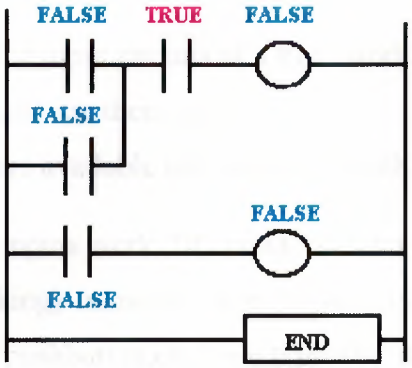


figure 1.18

Notice that even though the high level sensor became true there still is NO continuous true logic path and therefore coil 1000 remains false!

After 2000 scans the oil level falls below the low level sensor and it will also become true again. At this point the logic will appear the same as SCAN 1 above and the logic will repeat as illustrated above.

CHAPTER 2

2.1 DC INPUTS

Let's now take a look at how the input circuits of a PLC work. This will give us a better understanding of how we should wire them up.
Typically, DC input modules are available that will work with 5, 12,24, and 48 volts.

We'll first look at how the dc inputs work. DC input modules allow us to connect either PNP (sourcing) or NPN (sinking) transistor type devices to them. If we are using a regular switch (i.e. toggle or pushbutton,etc.) we typically don't have to worry about whether we wire it as NPN or PNP. We should note that most PLCs won't let us mix NPN and PNP devices on the same module. When we are using a sensor (photoeye, proximity sensor, etc.) we do, however, have to worry about its output configuration.

The difference between the two types is whether the load (in our case, the PLC is the load) is switched to ground or positive voltage. An NPN type sensor has the load switched to ground whereas a PNP device has the load switched to positive voltage.
Below is what the outputs look like for NPN and PNP sensors.

NPN(SINKING)SENSOR

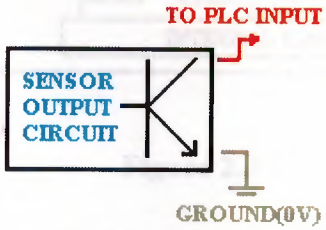


Figure 2.1

On the NPN sensor we connect one output to the PLCs input and the other output to the power supply ground. If the sensor is not powered from the same supply as the PLC, we should connect both grounds together. Engineers will say that PNP is better (i.e. safer) because the load is switched to ground.

On the PNP sensor we connect one output to positive voltage and the other output to the PLCs input. If the sensor is not powered from the same supply as the PLC, we should connect both V+'s together. PNP sensors are most commonly used in Europe.

PNP(SOURCING)SENSOR

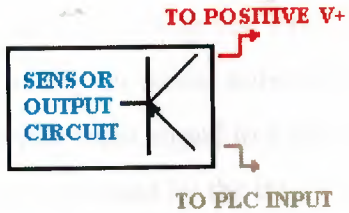


Figure 2.2

Inside the sensor, the transistor is just acting as a switch. The sensors internal circuit tells the output transistor to turn on when a target is present. The transistor then closes the circuit between the 2 connections shown above. (V+ and PLC input).

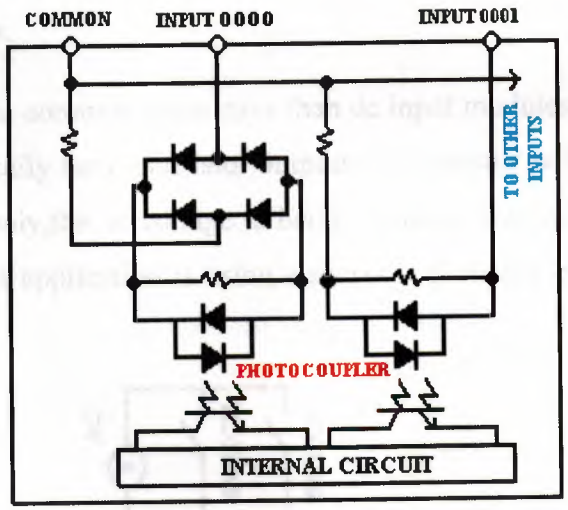


Figure 2.3

The only things accessible to the user are the terminals labeled COMMON, INPUT 0000, INPUT 0001, INPUTxxxx… The common terminal either gets connected to V+ or ground. Where its connected depends upon the type of sensor used. When using an NPN sensor this terminal is connected to V+. When using a PNP sensor this terminal is connected to 0V (ground).

A common switch (i.e. limit switch, pushbutton, toggle, etc.) would be connected to the inputs in a similar fashion. One side of the switch would be connected directly to V+. The other end goes to the PLC input terminal. This assumes the common terminal is connected to 0V (ground). If the common is connected to V+ then simply connect one

end of the switch to 0V (ground) and the other end to the PLC input terminal.

The optocouplers are used to isolate the PLCs internal circuit from the inputs. This eliminates the chance of any electrical noise entering the internal circuitry. They work by converting the electrical input signal to light and then by converting the light back to an electrical signal to be processed by the internal circuit.

2.2 AC INPUTS

Now that we understand how dc inputs work, let's take a close look at ac inputs. An ac voltage is nonpolarized. Typically, AC input modules are available that will work with 24, 48, 110, and 220 volts.

AC input modules are less common these days than dc input modules. The reason being that today's sensors typically have transistor outputs. A transistor will not work with an ac voltage. Most commonly,the ac voltage is being switched through a limit switch or other switch type. If your applicatipn is using a sensor it probably is operating on a dc voltage.

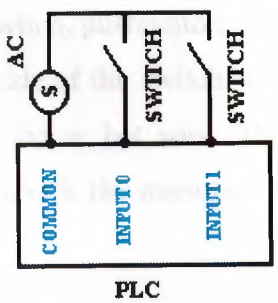


Figure 2.4

We typically connect an ac device to our input module as shown above. Commonly the ac "hot" wire is connected to the switch while the "neutral" goes to the PLC common. The ac ground (3rd wire where applicable) should be connected to the frame ground terminal of the PLC.(not shown) As is true with dc, ac connections are typically color coded so that the individual wiring the device knows which wire is which. This coding varies from country to country but in the US is commonly white (neutral), black (hot) and green (3rd wire ground when applicable). Outside the US it's commonly coded as

brown (hot), blue (neutral) and green with a yellow stripe (3rd wire ground where applicable).

The PLCs ac input module circuit typically looks like this:

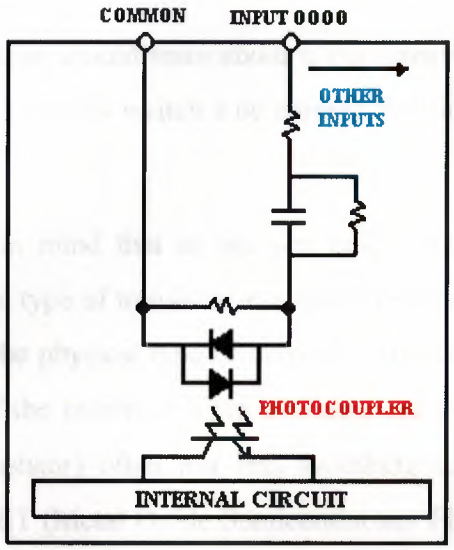


Figure 2.5

The only things accessible to the user are the terminals labeled COMMON, INPUT 0000, INPUTxxxx... The common terminal gets connected to the neutral wire.

A common switch (i.e. limit switch, pushbutton, toggle, etc.) would be connected to the input terminals directly. One side of the switch would be connected directly to INPUT XXX. The other end goes to the ac hot wire. This assumes the common terminal is connected to neutral. Always check the manufacturers specifications before wiring, to be sure AND SAFE.

The optocouplers are used to isolate the PLCs internal circuit from the inputs. This eliminates the chance of any electrical noise entering the internal circuitry. They work by converting the electrical input signal to light and then by converting the light back to an electrical signal to be processed by the internal circuit.

One last note, typically an ac input takes longer than a dc input for the PLC to see. In most cases it doesn't matter to the programmer because an ac input device is typically a mechanical switch and mechanical devices are slow. It's quite common for a PLC to require that the input be on for 25 or more milliseconds before it's seen. This delay is required because of the filtering, which is needed by the PLC internal circuit.

2.3 Transistor Outputs

The next type of output we should learn about is our transistor type outputs. It is important to note that a transistor can only switch a dc current. For this reason it cannot be used with an AC voltage.

We should also keep in mind that as we saw before with the input circuits, there are generally more than one type of transistor available. Typically a PLC will have either NPN or PNP type outputs. The physical type of transistor used also varies from manufacturer to manufacturer. Some of the common types available are BJT and MOSFET. A BJT type (Bipolar Junction Transistor) often has less switching capacity (i.e. it can switch less current) than a MOS-FET (Metal Oxide Semiconductor- Field Effect Transistor) type. The BJT also has a slightly faster switching time.

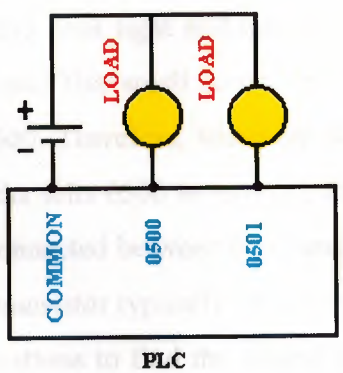


Figure 2.6

Shown above is how we typically connect our output device to the transistor output. Please note that this is an NPN type transistor. If it were a PNP type, the common terminal would most likely be connected to V+ and V- would connect to one end of our load. Note that since this is a DC type output we must always observe proper polarity for the output. One end of the load is connected directly to V+ as shown above.

Let's take a moment and see what happens inside the output circuit. Shown below is a typical output circuit diagram for an NPN type output.

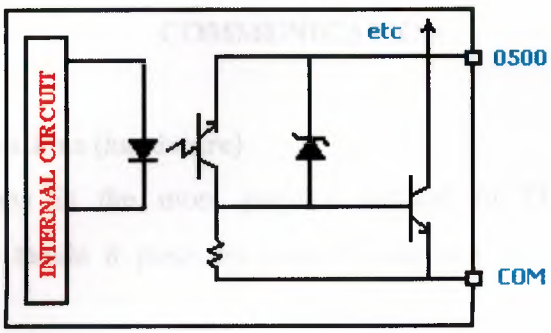


Figure 2.7

Notice that as we saw with the transistor type inputs, there is a photocoupler isolating the "real world' from the internal circuit. When the ladder diagram calls for it, the internal circuit turns on the photocoupler by applying a small voltage to the LED side of the photocoupler. This makes the LED emit light and the receiving part of the photocoupler will see it and allow current to flow. This small current will turn on the base of the output transistor connected to output 0500. Therefore, whatever is connected between COM and 0500 will turn on. When the ladder tells 0500 to turn off, the LED will stop emitting light and hence the output transistor connected between 0500 and COM will turn off. One other important thing to note is that a transistor typically cannot switch as large a load as a relay. Check the manufacturers specifications to find the largest load it can safely switch. If the load current you need to switch exceeds the specification of the output, you can connect the PLC output to an external relay. Then connect the relay to the large load. You may be thinking, "why not just use a relay in the first place"? The answer is because a relay is not always the correct choice for every output. A transistor gives you the opportunity to use external relays when and only when necessary.

In summary, a transistor is fast, switches a small current, has a long lifetime and works with dc only. Whereas a relay is slow, can switch a large current, has a shorter lifetime and works with ac or dc.

CHAPTER 3

COMMUNICATION

3.1 RS-232 Communications (hardware)

RS-232 communications is the most popular method of PLC to external device communications. Let's tackle it piece by piece to see how simple it can be when we understand it.

RS-232 is an asynchronous (a marching band must be "in sync" with each other so that when one steps they all step. They are asynchronous in that they follow the band leader to keep their timing) communications method. We use a binary system (1's and 0's) to transmit our data in the ASCII format. (American Standard Code for Information Interchange- pronounced ASS-KEY) This code translates human readable code (letters/numbers) into "computer readable" code (1's and 0's). Our pies serial port is used for transmission/reception of the data. It works by sending/receiving a voltage. A positive voltage is called a MARK and a negative voltage is called a SPACE. Typically, the PLC works with +/-15volts. The voltage between +/- 3 volts is generally not used and is considered noise.

There are 2 types of RS-232 devices. The first is called a DTE device. This means Data Terminal Equipment and a common example is a computer. The other type iDCE device. DCE means Data Communications Equipment and a common example is a modem. Your PLC may be either a DTE or DCE device. Check your documentations

The PLC serial port works by turning some pins on while turning other off. These pins each are dedicated to a specific purpose. The serial port comes in 2 flavors a 25-pin type and a 9-pin type. The pins and their purposes are shown below. (This chart assumes your PLC is DTE device)

9-PIN	25-PIN	PURPOSE
1	1	frame ground
2	3	receive data (RD)
3	2	transmit data (TO)
4	20	data terminal ready (DTR)
5	7	signal ground
6	6	data set ready (DSR)
7	4	request to send (RTS)
8	5	clear to send (CTS)
9	22	ring indicator (RI) *only for modems*

Table 3.1

Each pins purpose in detail:

· frame ground- This pin should be internally connected to the chassis of the device.

· receive data- This pin is where the data from the external device enters the PLC serial port.

· transmit data- This pin is where the data from the PLC serial port leaves the PLC enroute to the external device.

· data terminal ready- This pin is a master control for the external device. When this pin is 1 the external device will not transmit or receive data.

· signal ground- Since data is sent as + or - voltage, this pin is the ground that is referenced.

· data set ready- Usually external devices have this pin as a permanent 0 and the PLC basically uses it to determine that the external device is powered up and ready.

· request to send- This is part of hardware handshaking. When the PLC wants to send data to the external device it sets this pin to a 0. In other words, it sets the pin to a 0 and basically says "I want to send you data. Is it ok?" The external device says it's OK to send data by setting its clear to send pin to 0. The PLC then sends the data.

. clear to send- This is the other half of hardware handshaking. As noted above, the external device sets this pin to 0 when it is ready to receive data from the PLC.

. ring indicator- only used when the PLC is connected to a modem.

What happens when your PLC and external device are both DTE (or both DCE) devices? They can't talk to each other, that's what happens. The picture below shows why 2 same type devices can't communicate with each other.

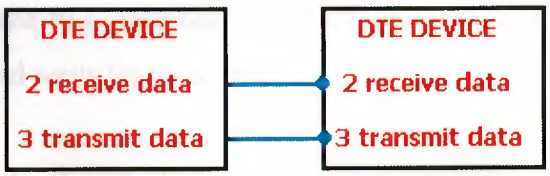


Figure 3.1

Notice that in the picture above, the receive data line (pin2) of the first device is connected to the receive data line of the second device. And the transmit data line (pin3) of the first device is connected to the transmit data of the second device. Ifs like talking through a phone with the wires reversed. (i.e. your mouth piece is connected directly to the other parties mouthpiece and your ear piece is connected directly to the other parties earpiece.) Obviously, this won't work well!

The solution is to use a null-modem connection as shown below. This is typically done by using a reverse (null-modem) cable to connect the devices.

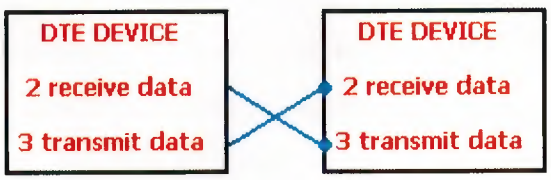


Figure 3.2

To summarize everything, here's a typical communications session. Both devices are powered up. the PLC is DTE and the external device is DCE.

The external device turns on DSR which tells the PLC that's it's powered up and "there".

The PLC turns on RTS which is like asking the external device "are you ready to receive some data?" The external device responds by turning on it's CTS which says it's ok to for the PLC to send data. The PLC sends the data on its TD terminal and the external device receives it on its RD terminal. Some data is sent and received. After a while, the external device can't process the data quick enough. So, it turns off its CTS terminal and the PLC pauses sending data. The external device catches up and then turns its CTS terminal back on. The PLC again starts sending data on its TD terminal and the external device receives it on its RD terminal. The PLC runs out of data to send and turns off its RTS terminal. The external device sits and waits for more data.

3.2 RS-232 Communications (software)

Now that we understand the hardware part of the picture, let's dive right into the software part. We'll define a few of the common terms.

ASCII is a human-readable to computer-readable translation code. (i.e. each letter/number is translated to 1's and 0's) It's a 7-bit (a bit is a 1 or a 0) code, so we can translate 128 characters. (2^7 is 128) Character sets that use the 8^{th} bit do exist but they are not true ASCII. Below is an ASCII chart showing its "human-readable" representation. We typically refer to the characters by using hexadecimal terminology. "0" is 30h, "5" is 35h, "E" is 45h, etc. ("h" means hexadecimal)

	Most significant bits								
Least Sign. Bits		0	1	2	3	4	5	6	7
	0			space	0	@	P	`	p
	1		XON	!	1	A	Q	a	q
	2	STX		"	2	B	R	b	r
	3	ETX	XOFF	#	3	C	S	c	s
	4			$	4	D	T	d	t
	5		NAK	%	5	E	U	e	u
	6	ACK		&	6	F	V	f	v
	7			'	7	G	W	g	w
	8			(	8	H	X	h	x
	9			)	9	I	Y	i	y
	A	LF		*	:	J	Z	j	z
	B			+	;	K	[	k	{
	C			,	<	L	\	l	|
	D	CR		-	=	M	]	m	}
	E			.	>	N	^	n	~
	F			/	?	O	_	o	

Table 3.2

- **Start bit**-In RS-232 the first thing we send is called a start bit.This start bit(invented during WW1 by Kleinschmidt) is a synchronizing bit added just before each character we are sending.This is considered a SPACE or negative voltage or a 0.
- **Stop bit**- The last thing we send is called a stop bit.This stop bit tells us that the last character was just sent.Thing of it as an end-of-character bit.This is considered a MARK or positive voltage or a 1.The start and stop bits are commonly called framing bits because they surround the character we are sending.
- **Parity bit**-Since most PLCs/external equipment are byte-oriented(8 bits=1 byte)it seems natural to handle data as a byte.Althought ASCII is a 7-bit code it is rarely transmitted that way.Typically,the 8^{th} bit is used as a parity bit for error checking.This method of error checking gets its name from the math idea of parity.In simple terms,prity means that all characters will either have an odd number of 1's or an even number of 1's.

Common forms of parity are None,Even, and Odd.(Mark and Space aren't very common so I won't discuss them).Consider these examples;

Send "E" (45h or 1000101(binary)).

In parity of None,the parity bit is always 0 so we send 10001010.

In parity of Even we must have an Even number of 1's in our total character so the original character currentky has 3 1's(1000101)therefore our parity bit we will add must be a 1.(10001011)now we have an evev number of 1's.

In odd paity we need an odd number of 1's.Since our original character already has an odd number of 1's (3 is an odd number) our parity bit will be a 0.(10001010)

During transmission, the sender calculates the parity bit and sends it. The receiver calculates parity for the 7 -bit character and compares the result to the parity bit received. If the calculated and real parity bits don't match, an error occurred an we act appropriately It's strange that this parity method is so popular. The reason is because it's only effective half the time. That is, parity checking can only find errors that effect an odd number of bits. If the error affected 2 or 4 or 6 bits the method is useless. Typically, errors are caused by noise which comes in bursts and rarely effects 1 bit. Block redundancy checks are used in other communication methods to prevent this.

- **Baud rate-** I'll perpetuate the incorrect meaning since it's most commonly used incorrectly. Think of baud rate as referring to the number of bits per second that are being transmitted. So 1200 means 1200 bits per second are being sent and 9600 means 9600 bits are being transmitted every second. Common values (speeds) are 1200, 2400, 4800, 9600, 19200, and 38400.

- **RS232 data format-** (baud rate-data bits-parity-stop bits) This is the way the data format is typically specified. For example, 9600-8-N-1 means a baud rate of 9600,

8 data bits, parity of None, and 1 stop bit.

The picture below shows how data leaves the serial port for the character "E" (45h 100 0101b) and Even parity .

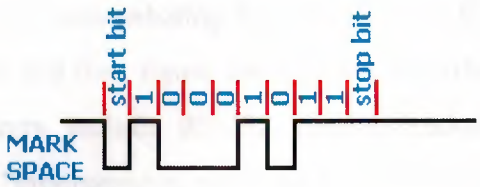


Figure 3.3

Another important thing that is sometimes used is called software handshaking (flow control). Like the hardware handshaking we saw in the previous chapter, software handshaking is used to make sure both devices are ready to send/receive data. The most popular "character flow control" is called XON/XOFF. It's very simple to understand. Simply put, the receiver sends the XOFF character when it wants the transmitter to pause sending data. When it's ready to receive data again, it sends the transmitter the XON character. XOFF is sometimes referred to as the hold off character and XON as the release character. The last thing we should know about is delimiters. A delimiter is simply added to the end of a message to tell the receiver to process the data it has received. The most common is the CR or the CR and LF pair. The PLC/external device receives this and knows to take the data from its buffer (where the data is stored temporarily before being processed). An LF (line feed) is also sometimes sent with the CR character. If viewed on a computer screen this would look like what happens on the typewriter when the carriage is returned and the page moves down a line so you don't type over what you just typed.
Sometimes an STX and ETX pair is used for transmission/reception as well. STX is "start of text" and ETX is "end of text". The STX is sent before the data and tells the external device that data is coming. After all the data has been sent, an ETX character is sent.

Finally, we might also come across an ACK/NAK pair. This is rarely used but it should be noted as well. Essentially, the transmitter sends its data. If the receiver gets it without error, it sends back an ACK character. If there was an error, the receiver sends back a NAK character and the transmitter resends the data.

3.3 Using RS-232 with Ladder Logic

Now that we understand what RS-232 is/means let's see how to use it with our PLC.

We should start out as always, remembering that a PLC is a PLC is a PLC... In other words, understand the theory first and then figure out how our manufacturer of choice "makes it work". Some manufacturers include RS-232 communication capability in the main processor. Some use the "programming port" for this. Others require you to purchase a module to "talk RS-232" with an external device.

To communicate via RS-232 we have to setup a few things. Ask yourself the following questions:

- Where, in data memory, will we store the data to be sent? Essentially we have to store the data we will send. Where, in data memory, will we put the data we receive from the external device?

- How will we tell the PLC when ifs time to send our data (the data we stored in data memory) out

the serial port?

- How will we know when we have received data from our external device?

If you know the above, then the rest is easy. If you don't know the above, then make something up and now the rest is easy. Huh??? Simple, pick a memory area to work with and figure out if we can choose the internal relays to use to send and receive data or if the PLC has ones that are dedicated to this

purpose.

1. We assign memory locations DM100 through DM102 to be where we'll put our data before we send it out the serial port. Many PLCs have dedicated areas of memory for this and only this purpose.

2. We'll assign internal relay 1000 to be our send relay. In other words, when we turn on 1000 the PLC will send the data in DM100-DM102 out the serial port to our external device. Note againMany PLCs have dedicated relays (special utility relays) for this and only this purpose. Ifs great when the manufacturer makes our life easy!

We'll send the string "air" out the PLC serial port to an operator interface when our temp sensor input turns on. This means our oven has become too hot. When the operator interface receives this string it will displayed an alarm message for the operator to see. Look back on the ASCII chart and you'll see that "air" is hexadecimal 61, 6C, 72. (a=61, I=6C, r=72) We'll write these ASCII characters (in hexadecimal form) into the individual data memory locations. We'll use DM100-102. Remember the LDA or MOV instruction? We'll turn on our send relay (1000) when our temperature sensor (0000) turns on. The ladder is shown below.

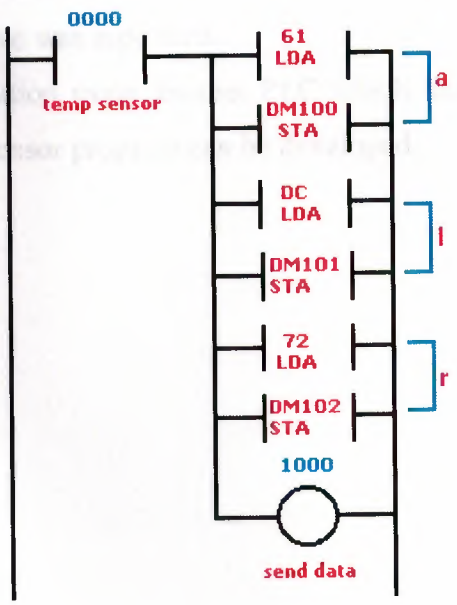


Figure 3.4

Some PLCs may not have dedicated internal relays that send out our data through the RS-232 port. We may have to assign them manually. Further, some pies will have a special instruction to tell us where the data is stored and when to send the data. This instruction is commonly called AWT (ASCII Write) or RS.

Put the data in a memory location and then turn on a relay to send the data.

CONCLUSION

The aim of this project is the learning PLC structure and working of PLC programming.But to see the working of PLC programming we should have devices which works steady.In this project mechanical were prepared before by an other student who graduated from NEU.So there were some broken devices and also motos and electromagnets were not work.Due to these problems first of all mechanical parts repaired to see PLC working clearly.

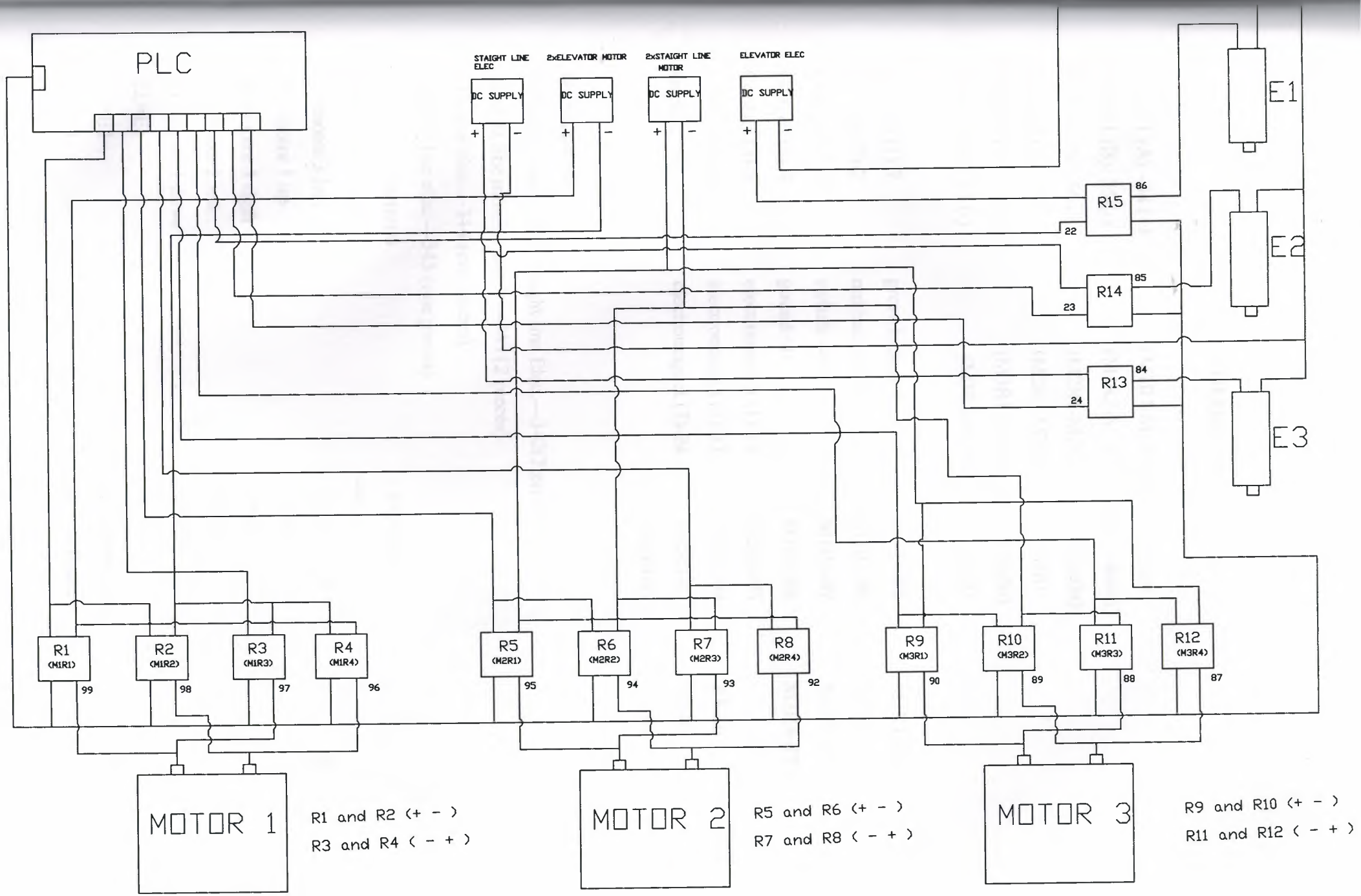
After repairing mechanical parts were modified.And finally PLC programme prepared.But we could not active every step.Because our PLC outputs was not enough to active.So the program were was seperated.

In the future application using another PLC which have more output and also using extra devices like a sensor program can be developed.

REFERENCES

1. SIEMENS cataloque.
2. Benjamin C. Kuo. Automatic Control Systems.
3. Brogan William L. Modern Control Theory.
4. SIEMENS PLC handbook.
5. www.odevim.com

APPENDIX


PLC
STAIGHT LINE ELEC
2xELEVATOR MOTOR
2xSTAIGHT LINE MOTOR
ELEVATOR ELEC
DC SUPPLY
DC SUPPLY
DC SUPPLY
DC SUPPLY
+
−
+
−
+
−
+
−
E1
E2
E3
R15
86
22
R14
85
23
R13
84
24
R1 (M1R1)
R2 (M1R2)
R3 (M1R3)
R4 (M1R4)
R5 (M2R1)
R6 (M2R2)
R7 (M2R3)
R8 (M2R4)
R9 (M3R1)
R10 (M3R2)
R11 (M3R3)
R12 (M3R4)
99
98
97
96
95
94
93
92
90
89
88
87
MOTOR 1
R1 and R2 (+ −)
R3 and R4 (− +)
MOTOR 2
R5 and R6 (+ −)
R7 and R8 (− +)
MOTOR 3
R9 and R10 (+ −)
R11 and R12 (− +)

COLOR CODES

Motor 1 (A) –34 (+)	(M1R1-M1R2)-72 = M1 (up)
Motor 1 (B)- 35 (-)	(M1R3-M1R4)-73= M1 (down)
Motor 2 (A)- 56 (+)	(M2R1-M2R2)- 74=M2 (right)
Motor 2 (B)- 57 (-)	(M2R3-M2R4)-75=M2 (left)
Motor 3 (A)- 62 (+)	(M3R1-M3R2)-76=M3 (right)
Motor 3 (B)- 63 (-)	(M3R3-M3R4)-77=M3 (left)

Contact (1)-7	green button-42	M1R1-99	M3R1-90
Contact (2)-2	red button- 43	M1R2-98	M3R2-89
Contact (3)-3	switch –44	M1R3-97	M3R3-88
Contact (4)-4	gound =0	M1R4-96	M3R4-87
Contact (5)-5	electromagnet (1)-22	M2R1-95	
Contact (6)-6	electromagnet (2)-23	M2R2-94	E1-86
Groung =0	electromagnet (3)-24	M2R3-93	E2-85
		M2R4-92	E3-84

Dc suppliers

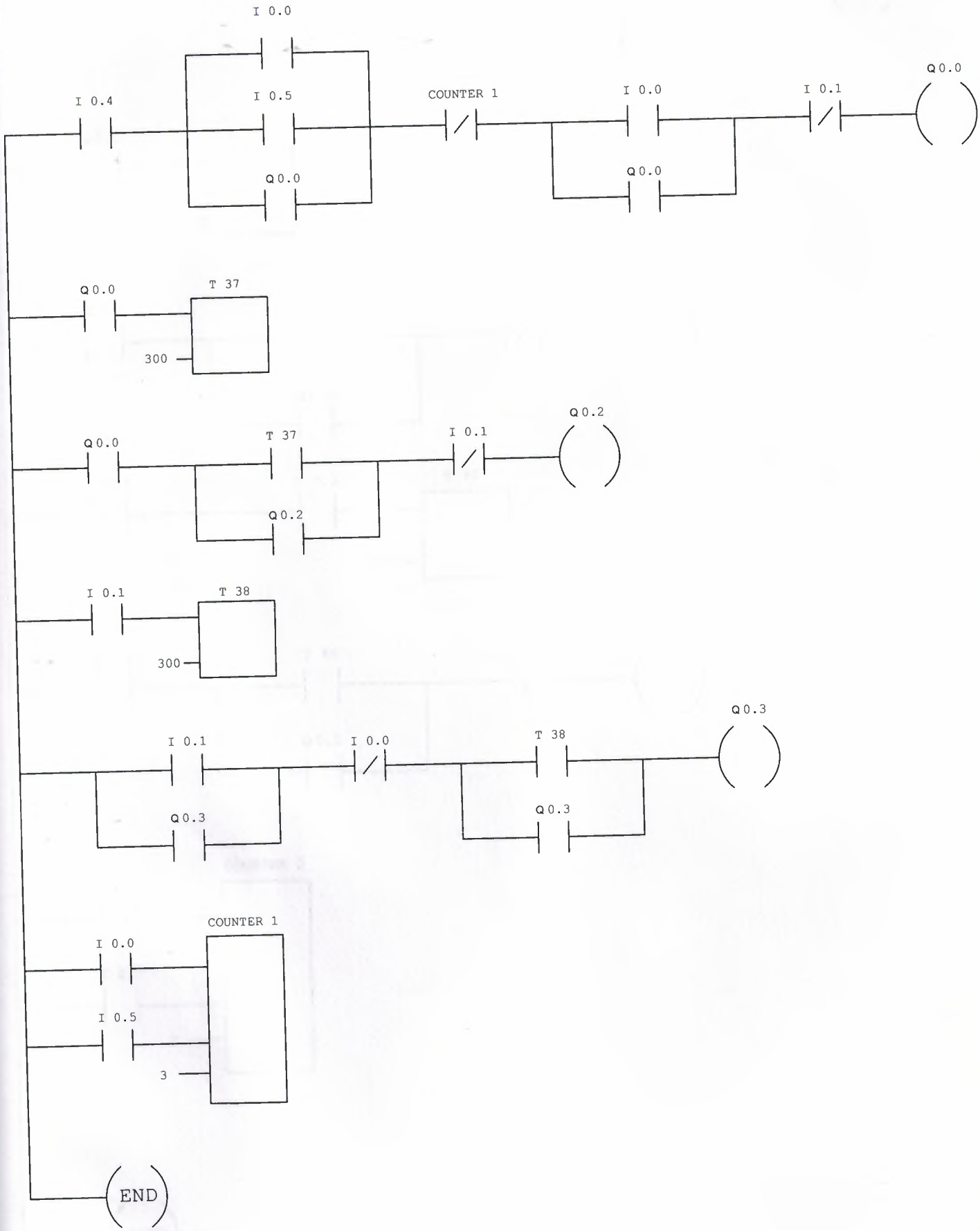
Elevator motor 1 and straight line Elec.---342(2 pieces)

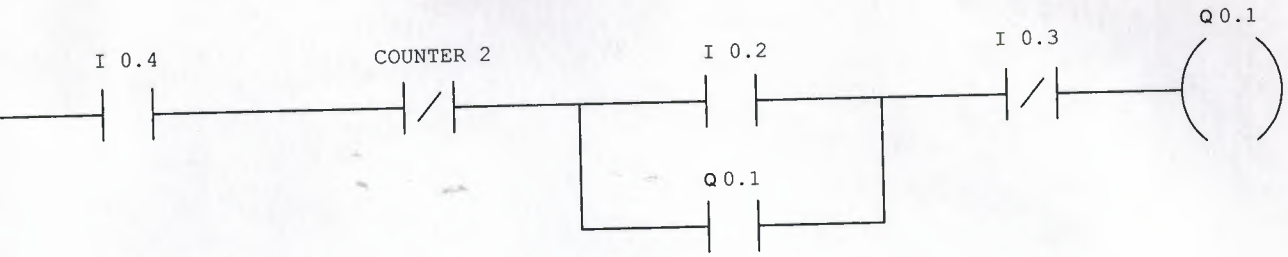
Straight line motor 2 and 3 ---343 (2 pieces)

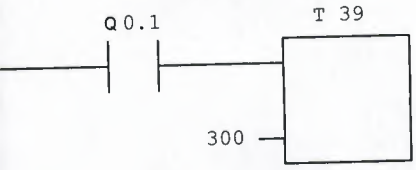
Elevator elec.---344 (one pieces)

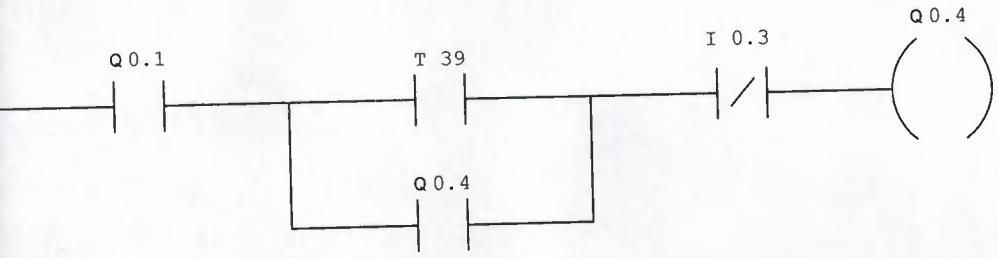
Straight line elec.---345 (one pieces)

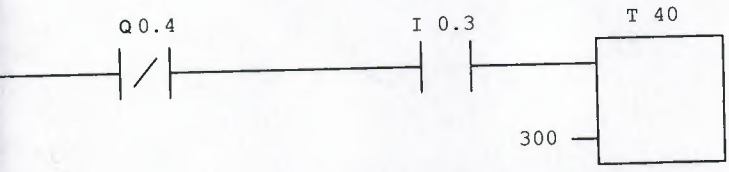
Outputs	**Inputs**
74=motor 2 rigth	44=switch
77=motor 3 left	7=C1
72 =motor 1 up	2=C2
76 =motor 3 rigth	3=C3
75 =motor 2 left	4=C4
73 =motor 1 down	5= C5
22 =E1	6 =C6
23 =E2	42 =green button
24 =E3	43 = redbutton

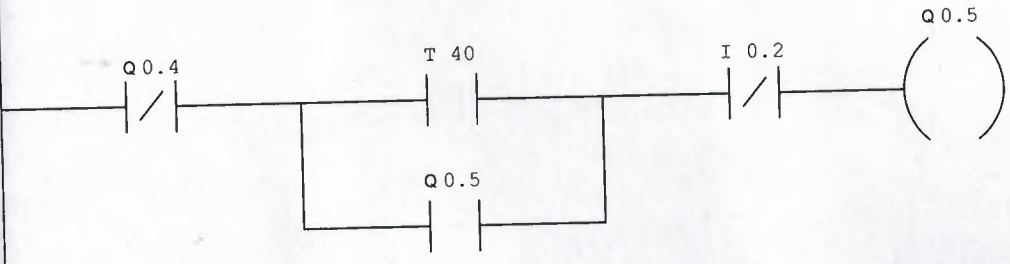

I 0.0
I 0.4
I 0.5
Q0.0
COUNTER 1
I 0.0
Q0.0
I 0.1
Q0.0
Q0.0
T 37
300
Q0.0
T 37
I 0.1
Q0.2
Q0.2
I 0.1
T 38
300
Q0.3
I 0.1
I 0.0
T 38
Q0.3
Q0.3
COUNTER 1
I 0.0
I 0.5
3
END

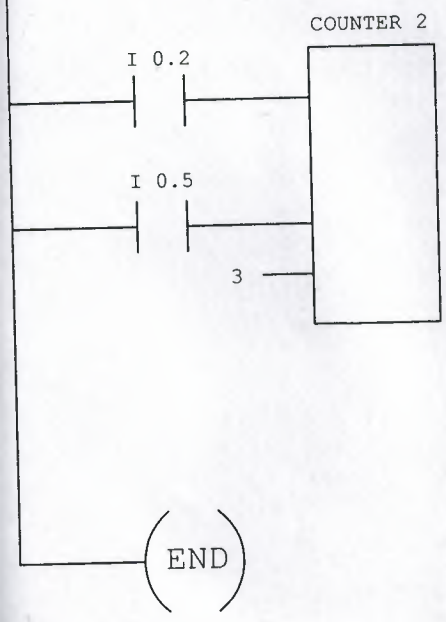

I 0.4
COUNTER 2
I 0.2
I 0.3
Q 0.1
Q 0.1


Q 0.1
T 39
300


Q 0.1
T 39
I 0.3
Q 0.4
Q 0.4


Q 0.4
I 0.3
T 40
300


Q 0.4
T 40
I 0.2
Q 0.5
Q 0.5


COUNTER 2
I 0.2
I 0.5
3
END